TRƯỜNG ĐẠI HỌC ĐÀ LẠT
KHOA TOÁN –TIN

Trương Chí Tín

GIÁO TRÌNH

NHẬP MÔN TRÍ TUỆ NHÂN TẠO

Đà Lạt, 04 - 2009

LỜI MỞ ĐẦU

Giáo trình "Nhập môn Trí tuệ nhân tạo" được viết dành cho sinh viên ngành Toán – Tin, Tin học và Công nghệ thông tin.

Để đọc giáo trình này, sinh viên cần có kiến thức cơ bản về lôgic, cấu trúc dữ liệu và thuật toán. Nội dung giáo trình này gồm 4 chương:

Chương 1: Khái niệm về trí tuệ nhân tạo
Chương 2: Các phương pháp giải quyết vấn đề
Chương 3: Biểu diễn và xử lý tri thức
Chương 4: Lập trình lôgic

Chương 1 giới thiệu tóm tắt lịch sử hình thành và phát triển cũng như các khái niệm chung nhất, các lĩnh vực nghiên cứu và ứng dụng chính của trí tuệ nhân tạo. Chương 2 trình bày các phương pháp biểu diễn và giải quyết vấn đề cơ bản: biểu diễn vấn đề trong không gian trạng thái bằng đồ thị thông thường, đồ thị VÀ/HOẶC, các phương pháp xác định trực tiếp lời giải, các phương pháp thử – sai (trong đó trình bày các phương pháp tìm kiếm theo chiều rộng, chiều sâu, theo hướng cực tiểu giá thành trên cây và đồ thị, thuật giải di truyền, phương pháp GPS, …) và các kỹ thuật heuristic. Chương 3 đề cập đến các phương pháp biểu diễn tri thức bằng: lôgic, luật sinh, mạng ngữ nghĩa, khung và các phương pháp xử lý tri thức bằng suy diễn dựa trên lôgic tất định và bất định. Chương 4 giới thiệu kỹ thuật lập trình lôgic thông qua ngôn ngữ lập tình Prolog.

Cuối mỗi chương có phần bài tập nhằm củng cố chắc hơn kiến thức lý thuyết và rèn luyện kỹ năng thực hành cho học viên. Các phần được in chữ nhỏ dành cho học viên đọc thêm.

Chắc chắn tài liệu này không tránh khỏi sơ suất, tác giả rất mong nhận được và chân thành biết ơn các ý kiến đóng góp quí báu của các bạn đồng nghiệp và độc giả nhằm làm cho giáo trình hoàn chỉnh hơn trong lần tái bản sau.

Đà lạt, 04 - 2009
Tác giả

MỤC LỤC

***CHƯƠNG IV*. LẬP TRÌNH LÔGIC**

Chương I

KHÁI NIỆM VỀ TRÍ TUỆ NHÂN TẠO

I.1. Lược sử hình thành và phát triển

* Trí tuệ nhân tạo (TTNT hay AI – Artificial Intelligence) là một trong những ngành mới trong lĩnh vực công nghệ thông tin. Có *nhiều quan điểm về trí tuệ nhân tạo*.

- Năm 1950, *Alan Turing đã đưa ra các "trắc nghiệm thông minh"* để nhận biết máy tính có thông minh hay không. Tuy vậy, cũng theo ông ta, tuy máy tính có thể thất bại trong các trắc nghiệm thông minh nhưng nó vẫn có thể thông minh.

- Theo quan điểm của *Minsky*, trí tuệ nhân tạo là một ngành khoa học nhằm *nghiên cứu, mô phỏng trên máy tính các hành vi và tư duy thông minh tương tự như con người*. Nó giúp máy tính có khả năng nhận thức, suy luận và phản ứng. Có *hai hướng tiếp cận trí tuệ nhân tạo*: dùng máy tính để bắt chước quá trình xử lý của con người và thiết kế những máy tính thông minh độc lập với cách suy nghĩ của con người.

- *Từ điển bách khoa toàn thư Webster* thì định nghĩa: "Trí tuệ là khả năng:
  1. *Phản ứng một cách thích hợp với những tình huống mới* thông qua hiệu chỉnh hành vi một cách thích đáng;
  2. *Hiểu rõ những mối liên hệ qua lại giữa các sự kiện* của thế giới bên ngoài nhằm đưa ra những hành động phù hợp để đạt tới một mục đích nào đó".

- Theo những *nhà tâm lý học nhận thức* thì *quá trình hoạt động trí tuệ của con người bao gồm 4 thao tác cơ bản:*
  1. Xác định tập đích (goal) cần đạt tới;
  2. Thu thập các sự kiện (facts) và các luật suy diễn (inference rules) để đạt tới tập đích đặt ra;
  3. Thu gọn (prunning) quá trình suy luận nhằm xác định một cách nhanh chóng tập các luật suy diễn có thể sử dụng được để đạt tới một đích trung gian nào đó;
  4. Áp dụng các cơ chế suy diễn (tiến hoặc lùi) cụ thể (inference mechanisms), dựa trên các thao tác thu gọn quá trình suy luận và những sự kiện trung gian mới được tạo ra, để dẫn dắt từ những sự kiện ban đầu đến những đích đã đặt ra.

** TTNT ra đời* dựa trên các thành quả của các ngành *tâm lý học nhận thức, lôgic hình thức*, … *Từ trên 2000 năm trước*, các nhà *triết học và tâm lý*

học đã cố gắng tìm hiểu *cách thức, cơ chế của quá trình nhớ, học tập, nhận thức và suy lý*.

- Vào đầu những *năm 50* của thế kỷ XX, nhờ sự *ra đời và cải tiến liên tục về hiệu suất hoạt động của máy tính,* đã xuất hiện xu hướng không chỉ nghiên cứu trí tuệ về mặt lý thuyết mà còn kiểm nghiệm các kết quả lý thuyết thông minh đó trên máy tính. Trong thời gian đầu mới hình thành, *nhiều công trình lý thuyết về TTNT vẫn chưa được kiểm nghiệm và triển khai trên thực tế* do chưa có ngôn ngữ lập trình đặc trưng cho TTNT, do hạn chế về kỹ thuật máy tính, giới hạn về bộ nhớ đặc biệt là tốc độ thực hiện và do vấn đề bùng nổ tổ hợp nảy sinh trong những thuật toán tìm kiếm lời giải cho các bài toán khó trong TTNT.

- Dựa trên các *thành quả về kỹ thuật phần cứng*, cùng với sự xuất hiện các ngôn ngữ lập trình đặc thù cho TTNT, *chuyên xử lý ký hiệu hình thức phục vụ cho lập trình lôgic* như IPL.V, *LISP* (viết tắt của LISt Processing, do Mc Cathy tại đại học MIT đề xuất năm 1960), PLANNER, *PROLOG* (viết tắt của PROgramming in LOGic, do Alain Colmerauer và nhóm công sự của ông tại đại học Marseilles xây dựng năm 1972), *nhiều giả thuyết hay kết quả thú vị về lý thuyết trong TTNT có điều kiện được kiểm nghiệm và trở thành các sản phẩm tin học cụ thể trên thị trường mang tính thông minh, hoạt động như các nhóm chuyên gia* nhiều kinh nghiệm trong từng lĩnh vực hẹp nào đó như y học, địa chất, dạy – học, chơi cờ,... Chẳng hạn các sản phẩm, chương trình: dẫn xuất kết luận trong hệ hình thức, chứng minh các định lý hình học phẳng, tính tích phân bất định, giải phương trình đại số sơ cấp, chơi cờ (Samuel), phân tích và chữa bệnh tâm lý (ELIZA), chuyên gia về y khoa (MYCIN ở đại học Stanford), phân tích và tổng hợp tiếng nói, điều khiển Robot theo đồ án "Mắt - tay", thăm dò khoáng sản (PROSPECTOR)...

  Khi sử dụng những sản phẩm chuyên dụng thông minh này, đặc biệt là lần đầu tiên, ta không khỏi ngạc nhiên về tính "thông minh" đến mức đôi khi ta có cảm giác chúng vượt trội hẳn khả năng của những người không chuyên nghiên cứu về lĩnh vực đặc thù đó.

- Trong những năm *1990*, ngành TTNT càng phát triển mạnh hơn nữa theo các hướng: *cơ sở tri thức và hệ chuyên gia, xử lý ngôn ngữ tự nhiên, lý thuyết nhận dạng hình ảnh, tiếng nói* và ứng dụng vào các kỹ thuật đa phương tiện, siêu văn bản, *mạng nơron, máy học, lý thuyết mờ trong lập luận xấp xỉ, lập trình tiến hoá, khai thác tri thức từ dữ liệu, ...*

* Có *vài dấu hiệu quan trọng của trí tuệ máy là các khả năng*: *học; mô phỏng các hành vi sáng tạo của con người; trừu tượng hóa, tổng quát hóa và suy diễn; tự giải thích hành vi; thích nghi với tình huống mới gồm khả năng thu*

nạp dữ liệu tích hợp, rút tri thức từ dữ liệu; *xử lý các biểu diễn hình thức (các ký hiệu tượng trưng, danh sách); vận dụng các tri thức heuristics sẵn có; xử lý các thông tin bất định, không đầy đủ, không chính xác, ...Trí tuệ máy khác trí tuệ người* ở chỗ nó *không thể nhìn trước được một phần hay toàn thể quá trình giải trong những tình huống mới* và *không tự sinh ra được các heuristics của chính bản thân chúng.*

* TTNT gồm *các phương pháp và kỹ thuật cơ bản* sau: phương pháp biểu diễn và giải quyết vấn đề; kỹ thuật heuristics; phương pháp biểu diễn và xử lý tri thức; phương pháp học và nhận dạng, xử lý ngôn ngữ tự nhiên và các ngôn ngữ lập trình cho TTNT. TTNT vẫn kế thừa các kỹ thuật cơ bản của tin học truyền thống như: xử lý danh sách, kỹ thuật đệ qui và quay lui, cú pháp hình thức, ... Trong bất kỳ một hệ thống TTNT nào cũng đều có *2 thành phần cơ bản liên quan mật thiết với nhau*: các phương pháp biểu diễn vấn đề và tri thức, các phương pháp tìm kiếm trong không gian bài toán, các chiến lược thu hẹp không gian lời giải và suy diễn.

I.2. Những lĩnh vực nghiên cứu của trí tuệ nhân tạo

I.2.1. Từ thuật toán đến thuật giải

* Đặc trưng của *thuật toán (Algorithm)*: yêu cầu *thỏa mãn nghiêm ngặt 3 tính chất: xác định, hữu hạn, đúng đắn. Ưu điểm*: những bài toán giải được bằng thuật toán có độ phức tạp không quá đa thức được áp dụng tốt trong thực tế. *Nhược điểm*: những thuật toán có phức tạp trên đa thức chỉ được áp dụng với không gian bài toán nhỏ; trên thực tế, lớp các bài toán khó chưa có thuật toán giải hoặc chưa biết được thuật toán giải hiệu quả rộng hơn rất nhiều.

Một hướng để giải quyết khó khăn đó là *mở rộng tính xác định, tính đúng* và *đưa vào thêm các thông tin đặc trưng về bài toán*, đưa vào máy tính một kiểu kinh nghiệm và "tư duy" của con người là sự *ước lượng*, để thu được các thuật giải heuristic.

* Đặc trưng của *thuật giải heuristic*: *độ phức tạp bé*, cho phép nhanh chóng *tìm ra các lời giải*, nhưng không phải luôn luôn tìm ra mà *có thể tìm thấy lời giải chỉ trong đa số trường hợp*; vả lại, các lời giải này *chưa chắc luôn đúng* hay *tối ưu* mà thường *gần đúng* hay *gần tối ưu*.

I.2.2. Phân loại các phương pháp giải quyết vấn đề

* *Biểu diễn vấn đề*: Xét vấn đề: A $\rightarrow$ B.

- Dạng *chuẩn*: Cho A, B tìm $\rightarrow$ (hai loại thuật toán, chương trình: thuật toán cần xác định trước chính là $\rightarrow$ và thuật toán tổng quát để tìm ra $\rightarrow$).
- Cho A, $\rightarrow$, tìm B (suy diễn *tiến*).
- Cho B, $\rightarrow$, tìm A (suy diễn *lùi*).

* *Nhóm các phương pháp* ***xác định trực tiếp lời giải***: phương pháp chính xác, phương pháp xấp xỉ gần đúng, phương pháp không tường minh, đệ qui, nguyên lý qui hoạch động

* *Nhóm các phương pháp* ***xác định gián tiếp lời giải*** *hoặc* ***tìm kiếm lời giải****:*

- *Phương pháp thử – sai*: vét cạn, nguyên lý mắt lưới, phương pháp nhánh cận, sinh và thử lời giải, phương pháp ngẫu nhiên (phương pháp Monte – Carlo, thuật giải di truyền GA), nguyên lý mê cung (dạng đệ qui), vét cạn dần (dưới dạng lặp) bằng cách quay lui và xác định dần thông tin về bài toán trong quá trình giải thông qua các cấu trúc không tuyến tính (chẳng hạn: cây, đồ thị hoặc đồ thị VÀ/HOẶC như các phương pháp tìm kiếm: theo chiều rộng, sâu, sâu dần, cực tiểu A^T), phương pháp GPS, …

- *Phương pháp heuristic trong trí tuệ nhân tạo*: là hướng tiếp cận quan trọng để xây dựng các hệ thống TTNT. Nó bao gồm các phương pháp và kỹ thuật tìm kiếm có sử dụng các tri thức đặc biệt từ chính bản thân lớp bài toán cần giải nhằm rút ngắn quá trình giải và nhanh chóng đi đến kết quả mong muốn mặc dù có thể không chắc chắn đó là cách giải quyết tối ưu nhưng lại có tính khả thi trong điều kiện thiết bị hiện có và thời gian yêu cầu. Trong kỹ thuật này người ta thường sử dụng kỹ thuật heuristis định lượng thông qua các hàm đánh giá. Chúng ta sẽ minh họa các phương pháp heuristics thông qua các phương pháp vét cạn thông minh (tìm kiếm tối ưu được bổ sung bằng tri thức đặc trưng về bài toán trên cây hoặc đồ thị tổng quát: A^{KT}, A*), nguyên lý tham lam, nguyên lý hướng đích (thuật giải leo núi), nguyên lý sắp thứ tự, nguyên lý khớp nhất, …

Những thông tin heuristic này vẫn *được gián tiếp đưa vào máy* tính thông qua con người. Vậy máy tính có thể *tự tạo ra các "tri thức", biết suy luận, chứng minh, tự học qua kinh nghiệm, máy có khả năng rút ra tri thức và vận dụng chúng vào việc giải quyết bài toán* hay không ?

Các *phương pháp trong trí tuệ nhân tạo* đã giúp máy tính thực hiện được trong một chừng mực nào đó các vấn đề đặt ra ở trên: các phương pháp biểu diễn

và xử lý tri thức, lập trình tiến hoá, mạng neuron nhân tạo, máy học, khai thác tri thức từ dữ liệu, …

I.2.3. Biểu diễn và xử lý tri thức

Có 4 phương pháp cơ bản biểu diễn và xử lý tri thức - dữ liệu tích hợp: phương pháp hình thức sử dụng cách tiếp cận logic (lôgic cổ điển - tất định: lôgic mệnh đề, lôgic vị từ; lôgic bất định: lôgic xác suất, lôgic khả xuất, lôgic mờ), các luật sinh (thường dùng trong các hệ chuyên gia), mạng ngữ nghĩa, bộ ba liên hợp OAV, cách biểu diễn bằng khung (hay dàn - Frame),... Các hệ chuyên gia là những thể hiện của việc kết hợp của các phương pháp biểu diễn và phương pháp xử lý tri thức (ví dụ: DENDRAL, MOLGEN, PROSPECTOR, MYCIN, ...).

I.2.4. Xử lý ngôn ngữ tự nhiên, các ngôn ngữ lập trình dựa trên việc xử lý danh sách, ký hiệu và lập trình logic: các ngôn ngữ lập trình LISP, PROLOG có hạn chế là chi phí lớn và khó phát triển hệ thống; còn CLIPS nhằm biểu diễn tri thức theo hướng đối tượng và xử lý các luật suy dẫn. Ta có thể thấy sự khác nhau cơ bản giữa lập trình truyền thống và lập trình xử lý ký hiệu trong TTNT qua bảng so sánh I.1.

Lập trình truyền thống	**Lập trình xử lý ký hiệu và lôgic**
- Xử lý dữ liệu	- Xử lý tri thức - dữ liệu tích hợp
- Dữ liệu trong bộ nhớ được đánh địa chỉ số	- Tri thức được cấu trúc trong bộ nhớ làm việc theo ký hiệu
- Xử lý theo các thuật toán	- Xử lý theo các thuật giải heuristics và cơ chế lập luận
- Định hướng xử lý các đại lượng định lượng số	- Định hướng xử lý các đại lượng định tính, logic, ký hiệu tượng trưng, danh sách
- Xử lý tuần tự hoặc theo lô	- Xử lý theo chế độ tương tác cao (hội thoại, theo ngôn ngữ tự nhiên,...)
- Không giải thích trong quá trình thực hiện	- Có thể tự giải thích hành vi hệ thống trong quá trình thực hiện

Bảng I.1

I.2.5. Lý thuyết nhận dạng *theo hướng thống kê, cấu trúc, đại số và heuristics* gồm: nhận dạng *hình ảnh và âm thanh* (HEARSAY-II, …).

I.2.6. Lập trình tiến hoá, mạng nơron, máy học, khai thác dữ liệu

- *Lập trình tiến hóa (Revolution Programming)* sử dụng ý tưởng *qui luật tiến hoá và học thuyết di truyền* của ngành sinh học: những gì hợp lý, *thích*

nghi tốt với môi trường sẽ có khả năng tồn tại lâu dài hơn trong quá trình đào thải, sinh tồn; một số đặc điểm của thế hệ trước sẽ di truyền, ảnh hưởng đến thế hệ sau thông qua *lai chéo*; thỉnh thoảng vẫn xuất hiện vài cá thể có đặc điểm khác hẳn (hoặc nổi trội lại các đặc điểm tiềm tàng) với thế hệ trước của chúng thông qua *đột biến*, … Khởi điểm của hướng nghiên cứu này là thuật giải di truyền (*GA - Genetic Algorithm*). *Ưu điểm* của các thuật giải GA là *có thể áp dụng đối với các bài toán chưa biết thuật toán nào hay chưa có thuật giải nào khả dĩ, hiệu quả* để giải. Có thể xem GA thuộc vào lớp các *thuật toán ngẫu nhiên* thông qua việc tạo ngẫu nhiên quần thể ban đầu cũng như các vị trí lai chéo hay tỉ lệ lai chéo và đột biến của chúng.

- *Mạng nơron nhân tạo* (hay vắn tắt hơn là mạng nơron, *ANN - Artificial Neural Networks)* mô phỏng mô hình và cơ chế hoạt động hưng phấn và ức chế thần kinh để điều khiển hoạt động của con người. Mô hình ANN có thể gồm nhiều lớp, trong đó ít nhất phải có lớp nhập (input layer) và lớp xuất (output layer), ngoài ra có thể có nhiều lớp ẩn (hidden layers) trung gian. Mỗi lớp gồm nhiều nút. Các kích thích từ môi trường ngoài được truyền vào mạng thông qua các nút của lớp nhập. Tổng hợp các kích thích này (phụ thuộc các trọng số mà mạng này cần học), nếu vượt quá một ngưỡng nào đó, sẽ gây kích thích (hưng phấn hay ức chế) đến các nút của lớp kế tiếp. Cứ thế, quá trình tiếp tục lan truyền đến lớp xuất. Một mô hình ANN thường được ứng dụng nhiều trong thực tế là mạng nơron lan truyền ngược. Lặp lại quá trình này, dựa trên việc cập nhật trọng số qua mỗi thế hệ sao cho giảm dần sai số giữa giá trị thật và giá trị do mạng kết xuất. Qua một số thế hệ luyện, mạng sẽ học được bộ trọng số thích hợp. *Ưu điểm* của các phương pháp luyện mạng ANN là *có thể áp dụng đối với các bài toán chưa biết thuật toán nào hay chưa có thuật giải nào khả dĩ, hiệu quả* để giải.

- *Máy học (LM - Learning Machine)* là quá trình rút ra qui luật từ dữ liệu, chẳng hạn học thông qua lôgic, học bằng quan sát dựa trên các độ đo phù hợp, học dựa trên cây định danh thông qua độ đo hỗn loạn thông tin trung bình, … Ta có thể dùng ANN để ứng dụng vào máy học.

- *Khai thác dữ liệu (DM - Data Mining)* nhằm rút ra tri thức từ dữ liệu thô, chẳng hạn đo độ phụ thuộc của một lớp các thuộc tính xác định vào lớp các thuộc tính phổ biến khác trong tập dữ liệu thô cho trước thông qua luật kết hợp, ...

I.3. Những ứng dụng của TTNT

- Điều khiển học, Robotic, giao diện người máy thông minh
- Trò chơi máy tính
- Thiết bị điện tử thông minh nhờ sử dụng lôgic mờ
- Hệ chuyên gia trong: giáo dục, y khoa, địa chất, quản lý, …
- Xử lý ngôn ngữ tự nhiên

- Nhận dạng hình ảnh, âm thanh, …
- Các hệ thống xử lý tri thức và dữ liệu tích hợp: cho phép xử lý đồng thời tri thức và dữ liệu (cơ sở dữ liệu suy diễn, biểu diễn luật đối tượng, hệ hỗ trợ quyết định)
- Mô hình hoá các giải pháp giải bài toán

Chương II
CÁC PHƯƠNG PHÁP GIẢI QUYẾT VẤN ĐỀ

II.1. Các phương pháp xác định trực tiếp lời giải

Đặc điểm của các phương pháp này là xác định *trực tiếp* được lời giải thông qua một *thủ tục tính toán* hoặc *các bước căn bản* để có được lời giải. Có ba loại phương pháp chính để xác định trực tiếp lời giải. Loại thứ nhất được áp dụng để giải các bài toán đã biết cách giải bằng các *công thức chính xác* (như công thức toán học). Loại thứ hai được dùng cho các bài toán đã biết cách giải bằng các *công thức xấp xỉ* (như các công thức xấp xỉ trong phương pháp tính). Loại thứ ba được áp dụng vào các bài toán đã biết *cách giải không tường minh thông qua các hệ thức truy hồi hay kỹ thuật đệ qui*.

II.1.1. Phương pháp giải chính xác: thông qua các công thức giải chính xác. Chẳng hạn, thuật toán giải phương trình bậc hai.

II.1.2. Phương pháp giải xấp xỉ: thông qua các công thức giải gần đúng. Chẳng hạn, phương pháp lặp tính tích phân xác định theo công thức hình thang trong học phần *"Phương pháp tính"*.

II.1.3. Phương pháp giải không tường minh: thông qua các hệ thức truy hồi hoặc kỹ thuật đệ qui.

* <u>*Thao tác đệ qui*</u> *F(x)* trên 1 đối tượng $x \in D$ nào đó. Xét hai trường hợp:

- Nếu đối tượng x thuộc một tập đặc biệt X_0 nào đó ($X_0 \subset D$) mà đã biết cách giải đơn giản thì thực hiện các thao tác sơ cấp tương ứng;

- Ngược lại, trước hết có thể thực hiện một thao tác G(x) nào đó, biến đổi x thành x'= H(x) $\in$ D rồi thực hiện thao tác tương tự F(x') trên x', sau đó có thể thực hiện thêm một thao tác K(x) nào đó trên x, sao cho sau một số hữu hạn bước này, các điểm $x^{n(*)}$ sẽ rơi vào tập X_0.

```
F(x) //x ∈ D
{
  if (x ∈ X0) ThaoTácSơCấp(x);    // điều kiện dừng
  else { G(x);
         x' = H(x);                //H(x) ∈ D
```

```
        F(x');                    // lời gọi đệ qui
        K(x);
      }
}
```

Các thao tác đệ qui thường gặp trong tin học là: *định nghĩa đệ qui, hàm hoặc thủ tục đệ qui, thuật toán đệ qui.*

* <u>*Chú ý*</u>: Khi thiết kế một thao tác đệ qui, ta cần có hai phần:

- *Phần cơ sở* (phần neo hay *điều kiện dừng*): thao tác sơ cấp đã biết cách thực hiện ngay trên tập con $X_0 \subset D$.
- *Phần gọi đệ qui* F(x'): cần phải bảo đảm *sau một số hữu hạn bước biến đổi x thì ta sẽ gặp điều kiện dừng: $H(H(\ldots H(x))) = x_0 \in X_0$.*
- Trên đây, ta xét đệ qui đuôi trực tiếp. Các trường hợp phức tạp hơn một chút như đệ qui nhánh trực tiếp và đệ qui gián tiếp (hay đệ qui hỗ tương) được xét tương tự.

* <u>*Ví dụ 1a*</u> (dãy số *Fibonacci*): Ở đầu tháng thứ 1 có 1 cặp thỏ con mới ra đời (F(0) = 0, F(1) = 1). Giả sử:
- Cứ sau mỗi tháng một cặp thỏ (từ sau hai tháng tuổi) sẽ sinh thêm một cặp thỏ con;
- Các con thỏ không bao giờ chết.

Hỏi số cặp thỏ F(n) sau n tháng là bao nhiêu?

Ta có công thức truy hồi để tính F(n) như sau:

$F(0) = 0; F(1) = 1$ ($X_0 = \{0; 1\}$)

$F(n) = F(n-1) + F(n-2), \forall n \geq 2$

Để tính F(n), ta có thể thực hiện theo các cách sau:

- *Thuật toán đệ qui* sau đây có độ phức tạp thuật toán với số phép cộng là $O(F(n)) = O(((1+\sqrt{5})/2)^n)$: độ phức tạp mũ, quá lớn, không khả thi !

```
Nguyên Fibonaci_De_Qui(n)
{       if (n ≤ 1) return n;
        else return (Fibonaci_De_Qui(n-1) + Fibonaci_De_Qui(n-2));
}
```

- *Thuật toán lặp* sau đây có độ phức tạp thuật toán với số phép cộng là O(n): hiệu quả hơn nhiều ! Ta có thể *khử đệ qui* bằng cách *dùng vòng lặp và vài biến phụ hoặc dùng cơ chế ngăn xếp*.

```
Nguyên Fibonacci_Lặp(n)
{  if (n==0 or n==1) return n;
   else { j = 1;
          Truoc = 0;
          HienTai = 1;
          while (j < n) do
          {     Sau = Truoc + HienTai;
                Truoc = HienTai;
```

```
                HienTai = Sau;
                j = j + 1;
            }
            return Sau;
        }
}
```

- *Tìm công thức tường minh từ hệ thức truy hồi*

$$F(n) = \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^n + \left(\frac{1-\sqrt{5}}{2}\right)^n \right]$$

* Phương pháp tổng quát tìm công thức tường minh cho F_n từ hệ thức truy hồi tuyến tính (hệ số hằng):

$\boldsymbol{F_n = b_1 F_{n-1} + b_2 F_{n-2}}$, $\forall n \geq 2$ với các trị $\{F_0, F_1\}$ cho trước.

Gọi $\{\Phi_1, \Phi_2\}$ là 2 nghiệm của *đa thức đặc trưng* tương ứng:

$$\Phi^2 - b_1\Phi - b_2 = 0.$$

Khi đó: $\mathbf{F_n = c_1 \Phi_1^n + c_2 \Phi_2^n}$, (c_1, c_2 sẽ được xác định từ các điều kiện đầu của F_0, F_1).

* <u>*Nhận xét*</u>: Trong nhiều bài toán khó, ta có thể dùng chiến lược "***Chia để trị***" để *tách nó thành nhiều bài toán con có cùng cách giải* như bài toán ban đầu thông qua *kỹ thuật đệ qui*.

- <u>*Ví dụ 1b*</u>: Tráo đổi hai phần *a[1..k]* và *a[k+1..n]* của mảng *a* gồm *n* phần tử (không nhất thiết có độ dài bằng nhau) mà không dùng mảng phụ.

Nếu k ≤ n-k thì trước tiên trao đổi hai phần bằng nhau a[1..k] và a[n-k..n]; sau đó trong mảng con a[1..n-k], ta chỉ cần tráo đổi *k* phần tử đầu với phần còn lại. Trường hợp k ≥ n-k, giải tương tự, ...

```
TraoHaiPhanBangNhau(a, Tu1, Tu2, SoPTuTrao)
{  for (i=1; i ≤ SoPTuTrao; i++)
        DoiCho(a[Tu1+i], a[Tu2+i]);
}

TraoHaiPhanBatKy(a, n, k) // k >= 0
{ i = 1; j = n;
  while (k >= i)
  { if (k < (i+j)/2)
        { TraoHaiPhanBangNhau(a, i, i+j-k, k-i+1);
          j = i + j - k - 1;
        }
    else { TraoHaiPhanBangNhau(a, i, k+1, j-k);
          i = i + j - k;
        }
  }
}
```

II.1.4. Phương pháp qui hoạch động:

* *Ý tưởng của nguyên lý qui hoạch động*: nghiệm của một bài toán con (của một bài toán) là sự kết hợp các nghiệm của các bài toán con nhỏ hơn của nó (trong trường hợp rời rạc có tính đệ qui thì *nghiệm trong n bước sẽ có được từ lời giải của k bước trước và lời giải trong n-k bước)*. Ta thường dùng phương pháp này để giải các bài toán tối ưu mà thỏa mãn nguyên lý trên.

Có thể xem nguyên lý tối ưu là một sự thể hiện tốt của phương pháp chia để trị trong việc giải quyết vấn đề. Khi thực hiện các tính toán trong phương pháp qui hoạch động, để thực hiện tính toán tại bước thứ n, nên *tận dụng các kết quả đã tính ở các bước trước* thông qua các hệ thức truy hồi và một vài biến phụ để lưu các kết quả trung gian trước đó (chẳng hạn, xét bài tập: tính tất cả các số tổ hợp C^k_n, với mọi k: $0 \le k \le n$).

* *Mô hình toán học của nguyên lý tối ưu*

- *Định nghĩa II.1 (hàm phân tích được):* Cho hàm f: $D \to R$, $D \subset R^n$,
$D_1 = \{x_1 \in R^1: \exists y \in R^{n-1}$ & $(x_1,y) \in D\}$, $D(x_1) = \{y \in R^{n-1} : (x_1, y) \in D\}$ $\forall x_1 \in D_1$.
Ta nói hàm f là *phân tích được* nếu tồn tại hai hàm g: $R^2 \to R$ và h: $R^{n-1} \to R^1$ sao cho:

. $f(x_1, y) = g(x_1, h(y))$, $\forall x = (x_1, y) \in D$, $x_1 \in D_1$, $y \in D(x_1)$

. Hàm *g đơn điệu không giảm theo biến thứ hai*:

$\forall x_1 \in D_1$, $\forall z_1, z_2 \in R^1$: $z_1 \ge z_2 \Rightarrow g(x_1, z_1) \ge g(x_1, z_2)$

(thật ra chỉ cần: $\forall x_1 \in D_1$, $\forall z_1, z_2 \in R^1$, $\exists y_1, y_2 \in D(x_1)$: $z_1 = h(y_1)$, $z_2 = h(y_2)$,
$z_1 \ge z_2 \Rightarrow g(x_1, z_1) \ge g(x_1, z_2)$)

- *Mệnh đề II.1:* Cho f là hàm phân tích được (trong định nghĩa II.1). Khi đó, ta có:

$$\underset{x \in D}{opt}\, f(x) = \underset{x_1 \in D_1}{opt}[g(x_1, \underset{y \in D(x_1)}{opt}[h(y)])]$$

- *Nhận xét:*

. Kết quả của mệnh đề trên cho phép ta *đưa việc tối ưu hàm nhiều biến về tối ưu các hàm theo các biến thành phần có số chiều bé hơn.*

. Ta thường gặp trường hợp *hàm g có dạng tuyến tính theo biến thứ hai:*

$$g(x_1, z) = r(x_1) + z$$

và *f có dạng cộng tính theo từng thành phần:*

$$f(x_1, x_2, \ldots, x_n) = r(x_1) + r(x_2) + \ldots + r(x_n)$$

* *Ví dụ 2* (bài toán người giao hàng *Salesman*): Hàng ngày, người giao hàng phải chuyển hàng qua n địa điểm, mỗi địa điểm đúng một lần, rồi quay lại địa điểm xuất phát. Bài toán đặt ra là: làm thế nào để anh ta có được một hành trình với đường đi ngắn nhất.

Ta biểu diễn bài toán bằng đồ thị định hướng G = (V, A), với V={1, 2, …, n} và độ dài cung C(i,j) > 0, nếu (i,j) ∈ A và C(i,j) = ∞, nếu (i,j) ∉ A. Không mất tính tổng quát, ta có thể giả sử đường đi của anh ta xuất phát từ đỉnh 1. Bất kỳ đường đi nào (chấp nhận được) của người giao hàng cũng có thể phân thành: cung (1,k) với k ∈ V\{1} và đường đi từ k tới 1 qua mỗi đỉnh thuộc V\{1} đúng một lần. Nếu đường đi của anh ta ngắn nhất thì đường đi từ k tới đỉnh 1 qua các đỉnh thuộc V\{1, k} phải ngắn nhất. Do đó

nguyên lý tối ưu được thỏa mãn. Gọi d(j, S) là độ dài đường đi ngắn nhất từ j đến đỉnh 1 qua mỗi đỉnh k ∈ S (∀S ⊂ V và S ≠ ∅) đúng một lần. Ta có công thức truy hồi:

Nghiệm tối ưu cần tìm là:

Rõ ràng, d(j, ∅) = C(j,1), ∀ j ∈ [2, n]. Từ công thức truy hồi trên, ta tính được d(j, S) với mọi S chỉ chứa 1 đỉnh. Từ đó, ta tính được d(j, S) với mọi S chỉ chứa 2 đỉnh, ... Cứ thế tiếp tục, ta tính được d(k, S) với mọi S = V\{1,k}, ∀ k ∈ [2, n]. Từ đó, ta tìm được

$$d(1, V \setminus \{1\}) = \min_{2 \le k \le n}(C(1,k) + d(k, V \setminus \{1,k\}))$$

$$d(j, S) = \min_{k \in S}(C(j,k) + d(k, S \setminus \{k\}))$$

nghiệm tối ưu.

- Cụ thể, xét đồ thị định hướng có 4 đỉnh và độ dài các cung được cho bởi ma trận C như sau:

$$\begin{pmatrix} 0 & 10 & 15 & 20 \\ 5 & 0 & 9 & 10 \\ 6 & 13 & 0 & 12 \\ 8 & 8 & 9 & 0 \end{pmatrix}$$

Ta có: d(2, ∅) = C(2,1) = 5
d(3, ∅) = C(3,1) = 6
d(4, ∅) = C(4,1) = 8

Từ công thức truy hồi, ta tính được:

d(2, {3}) = C(2,3) + d(3, ∅) = 15
d(2, {4}) = C(2,4) + d(4, ∅) = 18
d(3, {2}) = C(3,2) + d(2, ∅) = 18
d(3, {4}) = C(3,4) + d(4, ∅) = 20
d(4, {2}) = C(4,2) + d(2, ∅) = 13
d(4, {3}) = C(4,3) + d(3, ∅) = 15

Tương tự:

d(2, {3, 4}) = min{C(2,3) + d(3, {4}), C(2,4) + d(4, {3})}
= min {29, 25} = 25

d(3, {2, 4}) = min{C(3,2) + d(2, {4}), C(3,4) + d(4, {2})}
= min {31, 25} = 25

d(4, {2, 3}) = min{C(4,2) + d(2, {3}), C(4,3) + d(3, {2})}
= min {23, 27} = 23

Cuối cùng, ta có:

d(1, {2, 3, 4}) = min{ C(1,2) + d(2, {3, 4}), C(1,3) + d(3, {2, 4}),
C(1,4) + d(4, {2, 3})}
= min {35, 40, 43} = 35

Để tìm được hành trình ngắn nhất, gọi K(j,S) là đỉnh k, tại đó nó đạt min của công thức truy hồi để tính d(j,S). Từ đó, ta có:

K(1, {2, 3, 4}) = 2
K(2, {3, 4}) = 4
K(4, {3}) = 3

Vậy đường đi {1, 2, 4, 3, 1} là ngắn nhất và đạt giá trị 35.

Tương tự, có thể áp dụng nguyên lý qui hoạch động để giải bài toán sau.

- *Ví dụ 3* (*bài toán sắp ba lô)*: một chiếc ba lô có thể chứa được một khối lượng w. Có n loại đồ vật được đánh số 1, 2, …, n. Mỗi đồ vật loại i có khối lượng a_i và có giá trị c_i (các trị w, a_i, c_i đều nguyên dương, i = 1, 2, …, n). Cần sắp xếp các đồ vật vào ba lô để ba lô có giá trị lớn nhất có thể được. Giả sử rằng mỗi loại đồ vật có đủ nhiều để xếp (*bài tập*).

II.2. Các phương pháp thử - sai

II.2.1. Phương pháp vét cạn, nguyên lý mắt lưới, phương pháp sinh và thử, phương pháp nhánh cận

**** Bài toán 1**: Tìm tập LờiGiải = {x ∈ D: tính chất P(x) đúng}.* ***(BT1)***

a. Phương pháp vét cạn

* *Thuật toán vét cạn V1.1*: giải BT1

```
{   B1: LờiGiải = ∅;
    B2: Trong khi D ≠ ∅ thực hiện:
            x ← get(D);  //lấy khỏi D một phần tử x
            if (P(x)) LờiGiải = LờiGiải ∪ {x};
    B3: if (LờiGiải == ∅) write("Không có lời giải");
        else XuấtLờiGiải(LờiGiải);
}
```

* *Chú ý*: - Nếu hạn chế miền D càng bé thì thuật toán chạy càng nhanh.

* *Ví dụ 4*: Cho trước các số M, K nguyên dương. Tìm các bộ số nguyên dương x, y, z sao cho:

Thay vì chọn D = {(x,y,z) ∈ $\mathbf{N}^3$ ∩ $[1;M-2]^3$} (**N** là tập các số tự nhiên), ta lấy: D = $\begin{cases} x+y+z=M \\ x^3+y^3+z^3=K \end{cases}$

{(x,y,z) ∈ $\mathbf{N}^3$ ∩ $[1; \min\{M-2; \sqrt[3]{K-2}\}]^3$}.

Khi lập trình, ta thể hiện miền D bởi 3 vòng for theo x, y, z:

```
Tìm_xyz(M, K)
{ Max = min(M-2, pow(K-2,1.0/3));
  for (x = 1; x ≤ Max; x++)
    for (y = 1; y ≤ Max; y++)
      for (z = 1; z ≤ Max; z++)
        if (x+y+z==M && pow(x,3)+ pow(y,3)+ pow(z,3) == K)
            XuấtLờiGiải(x,y,z);
}
```

Thật ra, vẫn có thể cải tiến chương trình trên để thu hẹp miền D hơn nữa (*bài tập*) !

- Dựa vào thuật toán trên, ta có thể sửa đổi chút ít để giải bài toán tìm kiếm chỉ một lời giải sau:

**** Bài toán 2**: Tìm một lời giải $x_0 \in D$: mệnh đề $P(x_0)$ đúng.* ***(BT2)***

Thuật toán tìm một lời giải V1.2: giải BT2

```
{   Trong khi D ≠ ∅ thực hiện:
    {   x ← get(D);  //lấy khỏi D một phần tử x
        if (P(x))
            { XuấtLờiGiải(x);
              Dừng;      // điểm chính khác với thuật toán V1
            }
    }
    write("Không có lời giải");
}
```

* Đối với một lớp bài toán nào đó mà có thể tìm được một *điều kiện đủ Q(x) cho lời giải x*, ta có thể dùng *thuật giải* sau: *với mỗi x ∈ D mà Q(x) đúng thì xuất lời giải*. Chú ý rằng, ngoài các lời giải trên, trong D còn *có thể chứa các lời giải khác mà không thỏa điều kiện đủ này* ! Nguyên lý mắt lưới sau đây là một thể hiện của ý tưởng trên.

b. Nguyên lý mắt lưới:

* *Ý tưởng*: Những con cá lớn hơn kích thước mắt lưới lớn nhất sẽ còn lại trong lưới !

Để giải bài toán (1), nếu chứng minh được:

> "nếu tìm được điều kiện Q(x) đúng với một vài x thuộc một miền con của D có kích thước nhỏ hơn ε thì P(y) đúng với mọi y ∈ miền con đó"

thì: Chia lưới D thành n miền con D_i (mỗi miền D_i có kích thước nhỏ hơn ε). Với mỗi i = 1 .. n, xét nếu tồn tại $x \in D_i$ mà Q(x) đúng thì P(x) đúng.

* *Ví dụ 5*: Tìm nghiệm gần đúng (với độ chính xác eps > 0) của phương trình f(x) = 0 trên miền [a, b], với f là hàm liên tục.

Ta đã biết nếu $f(x_i).f(x_{i+1}) < 0$, với $[x_i, x_{i+1}] \subset [a, b]$ và $abs(x_{i+1}-x_i) < eps$ thì $x_k = (x_{i+1}+x_i)/2$ sai khác với một nghiệm chính xác của phương trình f(x) = 0 không quá eps/2 (*một điều kiện đủ để tìm nghiệm* của phương trình liên tục).

- *Thuật giải tìm nghiệm gần đúng*:

```
Nghiem_Gan_Dung(a, b, eps)
{ Sai_so = 1e-3;
  n = (b-a)/eps;
  xi = a;
```

```
  for  i=1 to n do
  { xi_1 = xi + eps;
    if (f(xi)*f(xi_1) < - Sai_so)  writeln((xi + xi_1)/2);
    xi = xi_1;
  }
}
```

c. ***Phương pháp sinh và thử***

* *Ý tưởng: Sinh dữ liệu* (*cấu trúc của lời giải* có thể không xác định trước khi giải mà *chỉ được tạo ra dần* trong quá trình tìm kiếm lời giải*), sau đó kiểm tra nó có thỏa điều kiện dừng hay không ?*

- *Thuật toán sinh và thử*:

```
Sinh_Thử
{ Init;                              // khởi tạo dữ liệu xuất phát
  Stop = False;
  While not Stop do
  { if (Accept(C)) Show(C); // nếu phương án C là chấp nhận thì xuất C
    if (Generate(C)=False) Stop = True;
        // nếu không sinh được thêm phương án C nào nữa thì dừng
  }
}
```

* *Ví dụ 6*: bài toán chỉnh hợp lặp chập k của n phần tử $X = \{0,1, \ldots,n-1\}$: $\{x_1, x_2, \ldots, x_k\}$, với $x_i \in X$, $1 \le i \le k$.

Thủ tục *Init* sẽ khởi tạo 0 cho vectơ lời giải $x = \{x_1, x_2, \ldots, x_k\}$. Trong ví dụ này không cần đến điều kiện kiểm tra *Accept*: ta luôn cho nó nhận trị đúng. Thủ tục *Generate*(x, k) sẽ tăng dần x[j] một đơn vị, j bắt đầu từ k đến 1, cho đến khi x[j]=n-1 thì khởi động lại 0 cho x[j], rồi giảm j đi một. Khi nào j=0 thì dừng việc sinh dữ liệu.

```
Boolean Generate(x, k)
{ j = k;  // sinh chỉnh hợp kế tiếp
  while (j>0 and x[j] == n-1) do
  { x[j] = 0;
    j = j-1;
  }
   if (j==0) return False;
   else { x[j] = x[j]+1;
          return True;
        }
}
```

(Ngoài ra, ta có thể giải bài toán này bằng cách này sử dụng thuật toán *đệ qui TryRờiRạc*(i) trong II.2.3 với điều kiện kết thúc nghiệm là i = k).

Để các thuật toán vét cạn *không bị bùng nổ tổ hợp về thời gian và không gian* nhớ, ta cần: ***giảm độ phức tạp tính toán*** (*không tính lại các hằng trong vòng lặp, cần tận dụng lại các kết quả tính toán ở các bước trước, dùng kỹ thuật lính canh* để đơn giản các biểu thức điều kiện của vòng lặp, …); ***thu gọn không gian tìm kiếm***.

* ***Chiến lược thu hẹp không gian tìm kiếm***: Trong các thuật toán vét cạn để tìm kiếm lời giải trong không gian D, đối với một lớp các bài toán nào đó, dựa trên các *đánh giá toàn cục* (ví dụ: duyệt các bộ tổ hợp), *cục bộ* (ví dụ: bài toán sắp ba-lô), nếu ta *tìm ra được các* ***điều kiện cần*** *cho lời giải,* khi đó ta có thể *cải tiến thuật toán bằng cách loại bỏ ngay các phương án trong D không thỏa điều kiện cần này !* Khi đó:

. hoặc *xét tập D* (nếu có thể) *chỉ chứa những trạng thái thoả mãn điều kiện cần* cho lời giải mà thôi, do đó *D được thu hẹp ngay*;

. hoặc xét tập D như thông thường, nhưng trong các thuật toán vét cạn, ta *hiểu get(D) là lấy ra khỏi D phần tử x: nếu x không thỏa điều kiện cần cho lời giải thì loại ngay nó (và thực hiện việc này càng sớm tới mức có thể để tránh các thao tác thừa) rồi lấy ngay phần tử x tiếp theo của D,* ngược lại mới kiểm tra tính chất P(x).

Phương pháp nhánh cận là một thể hiện của chiến lược này.

d. Phương pháp nhánh cận

* *Ý tưởng*: *nhánh có chứa quả phải nặng hơn trọng lượng quả*. Khi xây dựng thêm thành phần cho lời giải, dùng các phép kiểm tra đơn giản để *xác định chi phí tối thiểu đến lời giải (điều kiện cần cho lời giải)*. Loại *bỏ ngay các hướng đi tiếp theo mà chi phí tối thiểu này còn lớn hơn cả chi phí thấp nhất hiện thời (hướng không thỏa điều kiện cần).*

* *Ví dụ 7* (Bài toán người du lịch): Có n thành phố (được đánh số từ 1 đến n). Một người du lịch xuất phát từ một thành phố, muốn đi thăm các thành phố khác, mỗi thành phố đúng một lần rồi lại quay về nơi xuất phát. Giả thiết giữa hai thành phố j, k khác nhau bất kỳ đều có đường đi với chi phí c(j,k). Hãy tìm một hành trình có tổng chi phí nhỏ nhất.

Một hành trình x[1], x[2], …, x[n] là một hoán vị của {1, 2, …, n}. Dùng mảng lôgic ChưaĐến để đánh dấu các thành phố đã đi qua: ChưaĐến[k] = True nghĩa là người du lịch chưa đến thành phố k. Nếu việc đến thành phố k có tổng chi phí dự đoán thấp nhất để hoàn thành toàn bộ hành trình lớn hơn chi phí thấp nhất hiện thời thì ta không chọn đi tiếp k, ngược lại thì chọn k.

Giả sử ta đã đi qua j-1 thành phố x[1], …, x[j-1] với chi phí là S. Nếu đi tiếp đến thành phố x[j] = k thì chi phí từ x[1] đến x[j] là T = S + c[x[j-1],k]. Đoạn còn lại của hành trình gồm (n-j+1) đoạn nữa, với chi phí trên mỗi đoạn không ít hơn Cmin (là chi phí trực tiếp thấp nhất giữa hai thành phố khác nhau trong ma trận chi phí). Tổng chi phí thấp nhất để hoàn thành hành trình là:

T + (n-j+1) * Cmin = S + c[x[j-1],k] + (n-j+1)*Cmin

Để giải bài toán này, ta sẽ gọi thủ tục chính *TryRờiRạc*(0, 2).

```
TryRờiRạc(S, j)
{ for  k =1 to n do
    if (ChưaĐến[k])
       { T = S + c[x[j-1],k];
         if (T + (n-j+1)*Cmin < Min)  // Min = MaxInt trước thủ tục Try
         { x[j] = k; ChưaĐến[k] = False;
           if   ( j == n)
                 { if (T + c[k, xp] < Min)        // xp là thành phố xuất phát
                     {  y = x;         // mảng y lưu trữ lời giải tốt nhất tạm thời
                        Min = T + c[k, xp];
                     }
                 }
           else Try(T, j+1);
           ChưaĐến[k] = True;
         }
       }
}
```

II.2.2. Phương pháp ngẫu nhiên

Phương pháp này ***<u>được sử dụng khi chưa biết thuật toán nào hiệu quả</u>*** hay *<u>chưa có nhiều thông tin để giải bài toán.</u>*

a. *<u>Phương pháp Monte – Carlo</u>*

* *<u>Ý tưởng</u>*: Cho hai hình $S \subset \Omega = [a, b]^m \subset R^m$, giả sử đã biết công thức tính độ đo S() của S phụ thuộc vào tham số nào đó cần tính. Tung ngẫu nhiên n lần (độc lập) vectơ $x = \{x_1, x_2, \ldots, x_m\} \in \Omega$ (có phân phối đều trên Ω), gọi n_S là số lần $x \in S$. Khi đó, với n đủ lớn, ta có: $S() \approx (b-a)^m.n_S/n$.
Từ đó rút ra tham số cần tính theo số liệu thực nghiệm n_S, n:

$$\approx S^{-1}((b-a)^m.n_S/n)$$

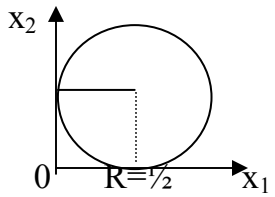
* *<u>Thuật toán ngẫu nhiên</u>*:

```
{     nS = 0;
      for  (j=1; j ≤ n; j++)
      { for  (k=1; k ≤ m; k++)
            x[k] = random(a,b); // tạo  các số ngẫu nhiên ∈ [a; b)
        if (x ∈ S) nS = nS + 1;
      }
      ≈ S^-1((b-a)^m.nS/n);
}
```

* *<u>Ví dụ 8</u>*: Dựa vào phương pháp trên, với m = 2, a = 0, b = 1 và S là hình tròn bán kính R=½, ta có thể tính số π như sau:

$$\pi = S/R^2 \approx 4*nS/n$$

$$x \in S <\text{---}> (x_1 - ½)^2 + (x_2 - ½)^2 \le ¼$$



- Trong một số trường hợp, ta có thể *dùng PP thử ngẫu nhiên để kiểm tra một tính chất hay giả thuyết nào đó.*

. *Ví dụ 9*: Kiểm tra giả thuyết Fermat bằng thực nghiệm: N ≥ 2 (N khá lớn) là nguyên tố nếu: x^{N-1} mod N = 1, ∀x nguyên dương < N ?

Để phù hợp với điều kiện thực nghiệm trên máy tính, ta xét *N* và số lần lặp *n* lớn vừa phải. Ta có thuật toán sau:

```
{       for  (j = 1; j ≤ n; j++)
        { x = 1 + random(N-1);  // 1 ≤ x < N
          y = x^(N-1) mod N;        // hãy cải tiến cách tính này hiệu quả hơn trên máy tính !
          if (y ≠ 1) {cout << N << " không là số nguyên tố"; Dừng; }
        }
        cout << N << " là số nguyên tố";
}
```

b. *Thuật giải di truyền GA (Genetic Algorithm) và lập trình tiến hoá*

* Lập trình tiến hoá:

Chương trình Tiến hóa = CTDL + GA

* *Các đặc trưng cơ bản của GA*

. Ngẫu nhiên;

. Duyệt toàn bộ giải pháp (lời giải), sau đó chọn giải pháp tốt nhất dựa trên *độ thích nghi* của chúng;

. Không quan tâm đến chi tiết vấn đề mà chỉ quan tâm đến giải pháp và phương pháp biểu diễn nó.

* *Các bước tiến hành chính của thuật giải di truyền GA*

- Bước 1: Chọn mô hình để biểu diễn vấn đề thông qua các *dãy ký hiệu* (số, chữ hoặc hỗn hợp) để biểu diễn cho mỗi giải pháp của vấn đề và *số cá thể* (số lời giải chấp nhận được) trong quần thể biểu diễn vấn đề.
- Bước 2: Tìm *hàm số thích nghi (Fitness function*) và tính số thích nghi *cho từng giải pháp*.
- Bước 3: Dựa trên các số thích nghi, thực hiện việc *sinh sản* và *tiến hoá* (gồm: *lai ghép* và *đột biến*) các giải pháp.
- Bước 4: Tính số thích nghi cho các giải pháp mới sinh sản, *loại bỏ giải pháp kém nhất*, *chỉ giữ lại một số nhất định các giải pháp* (có độ thích nghi cao).
- Bước 5: Nếu chưa tìm được *giải pháp tối ưu* hoặc chưa đến *thời hạn (hay số thế hệ) ấn định* thì trở lại bước 3 để tìm giải pháp mới.
- Bước 6: Nếu tìm được *giải pháp tối ưu* hay *hết thời hạn ấn định thì dừng và xuất kết quả.*

* *Các phương pháp tiến hoá của GA*: *sinh sản (Reproduction), lai ghép (*hay *lai chéo, Crossover), đột biến (Mutation)*. So với lai ghép, tần suất đột biến xảy ra ít hơn nhiều vì quá trình này tạo ra thông tin hoàn toàn mới.

* *Thuật giải GA*

```
Thuật_Giải_Di_ Truyền
{ t =0;
  Khởi tạo lớp P(t);
  Đánh giá lớp P(t);
  while (not(Điều kiện kết thúc)) do
  {     t = t + 1;
        Chọn lọc P(t) từ P(t-1);
        Kết hợp các cá thể của P(t);
        Đánh giá lớp P(t);
  }
}
```

* *Ví dụ 10*: Giải phương trình sau trên tập số tự nhiên: $x^2 = 64$.

. Bước 1: - Xác định số lượng cá thể (giải pháp, biến x): 4
- Dùng dãy ký hiệu nhị phân {0, 1} để biểu diễn mỗi biến

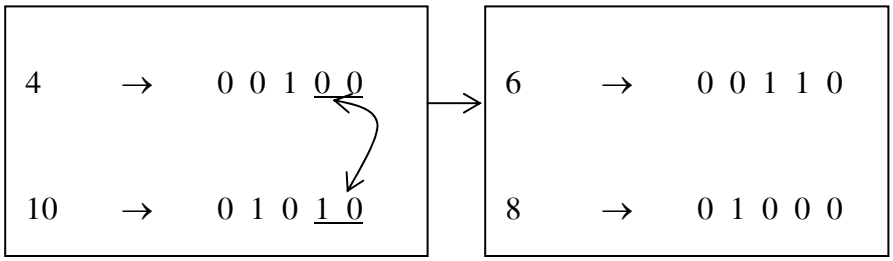
. Bước 2: - Chỉ định các cá thể: 4, 21, 10, 24
- Biểu diễn đáp số: dùng 5 bits

STT	Thập phân	Nhị phân
1	4	00100
2	21	10101
3	10	01010
4	24	11000

. Bước 3: - Chọn hàm số thích nghi: $F(x) = 1000 - |x^2 - 64| \geq 0, \forall x \in [-32..31]$
- Tính độ thích nghi phù hợp cho đáp số ≈ 1000

STT	Thập phân	Nhị phân	HS thích nghi	Chọn
1	4	00100	1048	X
2	21	10101	623	
3	10	01010	964	X
4	24	11000	488	

. Bước 4: - Chọn các đáp số (có độ thích nghi gần phù hợp): 4, 10
- Tiến hoá: lai ghép ở vị trí thứ 3:



. Bước 5: Tính độ thích nghi cho giải pháp mới:

STT	Thập phân	Nhị phân	HS thích nghi	Chọn
1	6	00110	1028	
2	21	10101	623	
3	8	01000	1000	X
4	24	11000	488	

. Bước 6: x = 8 có độ thích nghi 1000: phù hợp → Dừng.
. Bước 7: Đáp số: x = 8.

II.2.3. Nguyên lý mê cung

* *Vét cạn bằng cách quay lui:* Để tránh tràn bộ nhớ vì không gian D quá lớn, ta có thể *xác định dần các thành phần của lời giải* bài toán mặc dù các lời giải có *cấu trúc không tuyến tính* phức tạp. *Lời giải* $x_{LG} = (x_0, x_1, ...)$ *được xây dựng dần (xuất phát từ* $x_0 \in X_0$*) trong quá trình giải cho đến khi gặp điều kiện kết thúc* $P(x_{LG})$.

Các thành phần của lời giải được lưu dần vào tập DONG (có cấu trúc stack). Khi đó, tại mỗi thời điểm đang xét, phần tử ở *đỉnh của DONG chính là phần tử cuối của đường đi* có khả năng dẫn đến lời giải. Vì vậy, ta có thể *kiểm tra điều kiện kết thúc P(DONG) bởi Top(DONG)* $\in$ *DICH.*

Dưới dạng đệ qui, ta có các thuật toán sau để giải BT1 và BT2.

* *Thuật toán* ***vét cạn V2.1*** *(dưới dạng đệ qui giải BT1)*

```
{   for each x ∈ X0  do
    {DONG = {x}; Try_1(DONG);
    }
}

Try_1(DONG)
{ // điều kiện nhận biết trạng thái kết thúc của một lời giải          (*)
  if (P(DONG)) // hay (Top(DONG) ∈ DICH)
        XuấtLờiGiải(DONG);
  else  for each x ∈ B(Top(DONG))
          if (Q(x)) // nếu có yêu cầu thêm về tính chất của lời giải, chẳng hạn
                // (x∉DONG) khi đòi hỏi các thành phần của lời giải không trùng lặp
          {  Push (x, DONG);
             Try_1(DONG);
             Pop(y, DONG); // Trả lại trạng thái trước để quay lui
          }
}
```

trong đó: *Push(x,DONG), Pop(x,DONG)* và *x = Top (DONG)* lần lượt là các thao tác đưa vào, lấy ra và xem một phần tử *x* ở đỉnh ngăn xếp *DONG; B(x)* ($\subset D$) là tập các trạng thái kế tiếp có thể lấy từ *x* để xét tiếp.
Nếu tập xuất phát X_0 trong bài toán chỉ gồm một điểm x_0 thì để thi hành thuật toán, ta chỉ cần gọi *Try_1(*$\{x_0\}$*).*

Để đưa ra *thuật toán V2.2 giải BT2,* ta chỉ cần *thay điều kiện nhận biết trạng thái kết thúc một lời giải (*) bởi điều kiện dừng chương trình* sau đây*:*

```
if (P(DONG)) // hay (Top(DONG) ∈ DICH)
{ XuấtLờiGiải(DONG);
  Dừng;                                  // điều kiện dừng
}
```

Thuật toán *vét cạn kiểu đệ qui* trên đây *vét hết mọi khả năng nhưng tại mỗi bước chỉ lưu một khả năng*, những khả năng còn lại được lưu dần trong ngăn xếp thông qua cơ chế đệ qui. Nếu biểu diễn *không gian tìm kiếm của bài toán dưới dạng đồ thị hay cây* thì *thuật toán vét cạn* trên đây thực chất là thủ tục *tìm kiếm theo chiều sâu*.

Trong trường hợp lời giải là vectơ hữu hạn chiều (chẳng hạn, kích thước được biết trước là cố định hữu hạn), ta có phiên bản đơn giản và hiệu quả sau đây thường gặp trong các tài liệu tin học trước đây.

Thủ tục vét cạn Try trong trường hợp cấu trúc rời rạc tuyến tính đơn giản

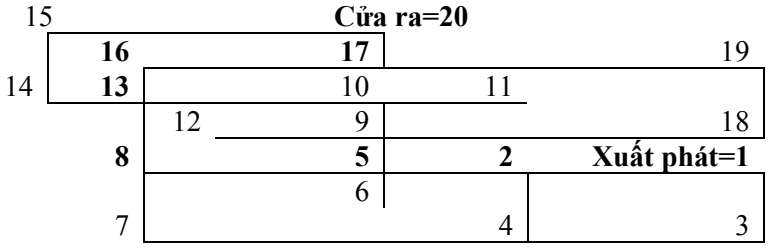
```
TryRờiRạc(j: integer)
{    for (k thuộc tập các khả năng) do
      if  (chấp nhận khả năng thứ k)
         { Xác định x_j theo khả năng thứ k;
           Đánh dấu (đã xét) trạng thái mới;
           if (x_j là trạng thái kết thúc)     // hay thoả điều kiện kết thúc
                    XuấtLờiGiải(x);
           else TryRờiRạc(j+1);
           Trả lại trạng thái cũ;              // Bỏ việc đánh dấu trạng thái cũ
         }
}
```

* *Ví dụ 11*: Tìm các đường đi từ điểm xuất phát đến cửa ra của mê cung trên *hình II.1*. Ta biểu diễn mê cung dưới dạng ma trận kề a[j,k] =1 hay 0 nếu có hay không có đường đi từ j đến k tương ứng, với số đỉnh n=20.

Gọi thủ tục *TryRờiRạcMêCung*(1) sau đây để giải bài toán mê cung.

```
TryRờiRạcMêCung (SoDinh)   // trước đó gán x[1] = XuấtPhát = 1
{    int k;
   for (k=1; k <= n; k++)
    // nếu có đường đi từ đỉnh x[SoDinh] đến đỉnh k và chưa đi qua k
     if  (a[x[SoDinh], k]==1 && DaDiQua[k]==0)
        {  x[SoDinh+1]=k;
           DaDiQua[k] = 1;                 // Đánh dấu đã đi qua k
           if (x[SoDinh+1] == Cửa_ra)
                    XuấtLờiGiải(x, SoDinh+1);
           else TryRờiRạcMêCung (SoDinh+1);
           DaDiQua[k] = 0;                 // Bỏ việc đánh dấu trạng thái cũ
```

```
        }
}
```



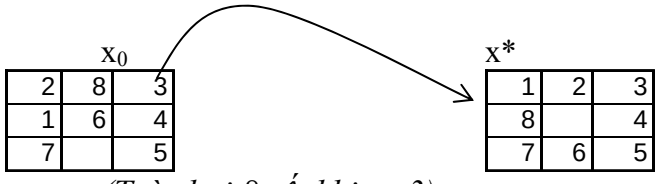
(Hình II.1)

II.2.4. Các phương pháp biểu diễn và giải quyết vấn đề trong không gian trạng thái bằng cây và đồ thị

a. Biểu diễn vấn đề trong không gian trạng thái

Trước tiên, ta xét hai ví dụ.

- *Ví dụ 12* (*bài toán Toci* hay *trò chơi n^2 - 1 số,* n là số tự nhiên, n > 2): Trong bảng ô vuông n hàng, n cột, mỗi ô chứa một số nguyên từ 1 đến n^2 -1 sao cho không có hai ô có cùng giá trị. Chỉ được có một ô trong bảng bị trống (ta có thể gán trị 0). Xuất phát từ một cách sắp xếp x_0 nào đó các số trong bảng, hãy dịch chuyển ô trống sang phải, trái, lên, xuống (nếu có thể được) để đưa về bảng mục tiêu x^* như sau:



(Trò chơi 8 số, khi n=3)

Để giải bài toán này, ta *biểu diễn nó trong không gian trạng thái S*. Mỗi trạng thái là một ma trận cấp *n x n* nhận các giá trị nguyên từ 0 đến n^2 -1 (trị 0 thay cho vị trí trống trên bảng) sao cho không có hai phần tử khác nhau có cùng trị, mỗi toán tử *o* là một phép dịch chuyển hợp lệ từ bảng này sang bảng khác. Số trạng thái chấp nhận được khá lớn: khoảng (1/2)*16! ≈ $10,5*10^{12}$ (khi n=4). Một cách *biểu diễn trực quan đối với không gian trạng thái và các toán tử là* ***đồ thị****. Trong đồ thị định hướng này các đỉnh tương ứng với các trạng thái, còn các cung tương ứng với các toán tử*. Ta cần *xây dựng dần các toán tử*, bắt đầu từ các toán tử có thể áp dụng cho trạng thái đầu, sau đó ở mỗi bước thêm vào một toán tử hợp lệ nào đó cho đến khi đạt được trạng thái đích.

Điểm mấu chốt khi giải quyết bài toán trong không gian trạng thái là lựa chọn một dạng mô tả nào đó của các trạng thái phù hợp với bản chất vật lý của bài toán. Ta *cần biểu diễn các trạng thái sao cho việc áp dụng các toán tử biến đổi trạng thái trở nên đơn giản hơn.*

* Có *hai cách biểu diễn: tập* ***O*** *các toán tử biến đổi trạng thái hoặc tập* ***P*** *những luật sinh để chuyển trạng thái:*

- Gọi ***O*** là tập các hàm *o* xác định và nhận trị trên không gian trạng thái S:

$o : S \to S$

Với ví dụ trên, tập các toán tử chuyển đổi trạng thái O gồm 4 toán tử $O=\{o_{len}, o_{xuong}, o_{trai}, o_{phai}\}$. Chẳng hạn, toán tử dịch chuyển vị trí ô trống lên trên o_{len} được xác định như sau:

$o_{len}(A) = B$

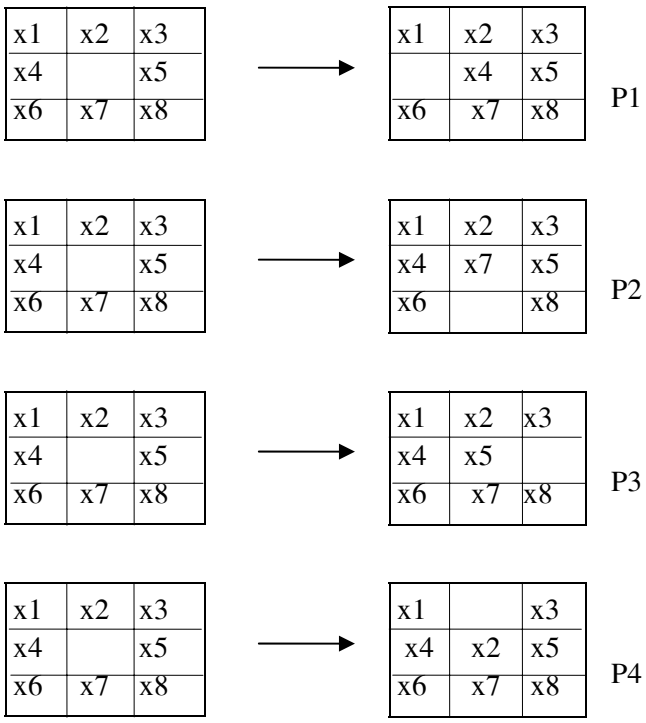
trong đó:

. B = $[b_{ij}]$, A = $[a_{ij}]$, giả sử ô trống trong ma trận A ở vị trí (i_0, j_0)

$$. b_{i,j} = \begin{cases} a_{io-1, jo} & \text{nếu } (i, j) = (i_o, j_o) \quad , i_o > 1 \\ a_{io, jo} & \text{nếu } (i, j) = (i_o - 1, j_o) \quad , i_o > 1 \\ a_{i,j} & \text{nếu ngược lại} \end{cases}$$

- Gọi ***P*** là tập các luật sinh (production rules) p_{ik}: $S_i \Rightarrow S_k$ để chuyển từ trạng thái S_i đến trạng thái S_k

Với ví dụ trên (với n=3), ta có các luật sinh sau:



x1	x2	x3
	x4	x5
x6	x7	x8

⟶

	x2	x3
x1	x4	x5
x6	x7	x8

P5

x1	x2	x3
	x4	x5
x6	x7	x8

⟶

x1	x2	x3
x6	x4	x5
	x7	x8

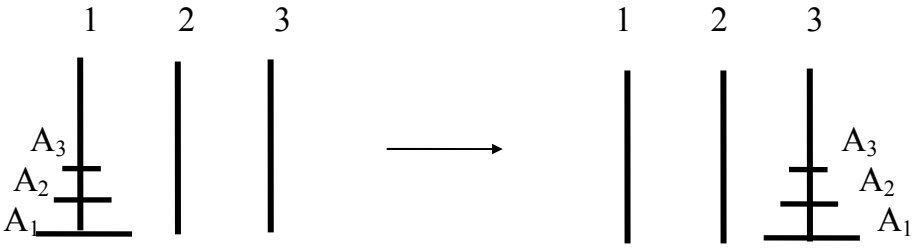
P6

- *Nhận xét*:

. Với cách biểu diễn dùng các *toán tử có biểu diễn tổng quát*: *số các toán tử ít (4, với bài toán $n^2 - 1$ số) , rất gọn và dễ cài đặt*.

. Với cách *liệt kê dưới dạng các luật sinh* tuy *trực quan* nhưng *số lượng quá lớn (4($n^2 - n$), với bài toán $n^2 - 1$ số).* Ta thường dùng cách liệt kê này với các bài toán mà phép chuyển đổi trạng thái rất khó khái quát.

- *Ví dụ 13* (bài toán *tháp Hà Nội*): Cho 3 cọc 1, 2, 3. Ở cọc 1 ban đầu có n đĩa sắp xếp theo thứ tự đĩa lớn ở dưới và đĩa nhỏ ở trên. Hãy dịch chuyển n đĩa đó sang cọc 3 sao cho: mỗi lần chỉ chuyển 1 đĩa, trong mỗi cọc không chấp nhận đĩa lớn nằm trên đĩa nhỏ. Với n=3, ta có:



Với bài toán này, ta có thể biểu diễn mỗi trạng thái là bộ ba (i, j, k) để mô tả đĩa A_1 ở cọc i, đĩa A_2 ở cọc j, đĩa A_3 ở cọc k. Khi đó, các toán tử dịch chuyển trạng thái sau là hợp lệ:

(i, j, k) $\rightarrow$ (i, j, j) , $k \neq j$

(i, j, k) $\rightarrow$ (i, j, i) , $k \neq i$

. . .

* Để *biểu diễn bài toán trong không gian trạng thái, cần xác định rõ:*

- *Không gian S biểu diễn các trạng thái.*
- *Tập O các toán tử biến đổi trạng thái hoặc tập P các luật sinh.*
- *Trạng thái đầu $x_o \in S$ và tập các trạng thái đích DICH $\subset S$*

* Một cách hình thức, ta có thể *phát biểu bài toán tìm kiếm trong không gian trạng thái dưới 3 dạng tương đương* như sau: Cho trạng thái đầu $x_o \in S$ *và tập các trạng thái đích DICH* $\subset S$.

Dạng 1 (toán tử): *Problem(S, O, x_o, DICH)*

Hãy tìm:

. dãy trạng thái $x_o,..., x_n$ sao cho $x_n \in$ DICH và có thể áp dụng dãy các toán tử biến đổi trạng thái $o_i \in O$ nào đó để chuyển từ x_{i-1} đến x_i

$$o_i : x_{i-1} \rightarrow x_i \quad \forall\, i = 1, 2, ..., n$$

. hoặc dãy toán tử $o_1 ,..., o_n \in O$: $o_n (o_{n-1} (...o_1 (x_o)...)) \in$ DICH

Dạng 2 (luật sinh): *Problem(S, P, x_o, DICH)*

Hãy tìm:

. dãy trạng thái $x_o,..., x_n$ sao cho $x_n \in$ DICH và có thể áp dụng dãy các luật sinh $p_i \in P$ nào đó để chuyển từ x_{i-1} đến x_i

$$p_i : x_{i-1} \Rightarrow x_i \ \forall\, i = 1, 2, ..., n$$

. hoặc dãy luật sinh $p_1 ,..., p_n \in P$ sao cho:

$$x_o \overset{p_1}{\Rightarrow} x_1 \Rightarrow \ ... \ \overset{p_n}{\Rightarrow} x_n \in \text{DICH}$$

Nếu không gây ra nhầm lẫn ta có thể dùng ký hiệu *Problem(x_o, DICH)*.

Dạng 3 (đồ thị):

Đồ thị định hướng G là cặp $G = (S, A)$, trong đó S là tập các đỉnh, A là tập các cung: $A = \{(a, b)/\ a, b \in S\} \subset S \times S$.

Với mỗi $n \in S$, ta ký hiệu tập các đỉnh con của n là: $B(n) = \{con \in S: (n,con) \in A\}$. Một cách tổng quát, ta định nghĩa:

$$B^1(n) \equiv B(n)$$

$$B^k(n) \equiv B(B^{k-1}(n)) = \bigcup_{m \in B^{k-1}(n)} B(m), k > 1$$

Khi đó:

$$\hat{B}(n) \equiv \bigcup_{i=1}^{\infty} B^i(n)$$

được gọi là *tập các đỉnh hậu duệ* của n và nếu $m \in B(n)$ thì n được gọi là *tổ tiên* của m. Dãy các đỉnh $p = \{n_1,..., n_k\}$ sao cho: $\forall\, i=1,..., k-1, a_i = (n_i , n_{i+1}) \in A$ được gọi là *đường đi* từ n_1 đến n_k hay còn có thể biểu diễn bởi: $p = \{a_1 ,..., a_{k-1}\}$, với $a_i = (n_i , n_{i+1})$. Nếu tồn tại một đường đi từ đỉnh *n* đến đỉnh *m* thì $m \in B(n)$ và ngược lại. Khi đó ta còn nói rằng m có thể đạt được từ n.

Khi đồ thị G có đỉnh gốc $n_o \in S$ và $\forall n \in S\backslash\{n_o\}$ tồn tại duy nhất đường đi từ n_o đến n ($\forall n \in S\backslash\{n_o\}$, $n \in B(n_o)$ và $\exists! m \in S$, $n \in B(m)$) thì G được gọi là *cây với gốc n_o*.

Thông thường, người ta thêm vào mỗi cung một đại lượng thể hiện *ý nghĩa được lượng hóa* của nó thông qua hàm giá *c:* $A \to R_+$, $c(n_i, n_k) \in R_+$, $\forall(n_i, n_k) \in A$. Khi đó, ta định nghĩa giá của đường đi $p = \{n_1, ..., n_k\}$:

$$c(p) = \sum_{i=1}^{k-1} c(n_i, n_{i+1})$$

Trường hợp $c \equiv 1$ thì c*(p)* được gọi là *độ dài đường đi của p.*

Theo ngôn ngữ đồ thị, *không gian trạng thái tương ứng với đồ thị định hướng trong đó: các trạng thái tương ứng với các đỉnh trong đồ thị và có một cung nối từ trạng thái s đến trạng thái t nếu tồn tại toán tử o sao cho o(s) = t* (hoặc tồn tại luật sinh *p* sao cho *p*: $s \Rightarrow t$).

* Bài toán tìm kiếm trong không gian trạng thái có thể phát biểu dưới *dạng đồ thị* như sau*:*

- <u>Bài toán 3</u>: Problem_3(G = (S, A), x_0, DICH) ***(BT3)***

*Cho đồ thị G = (S, A), với đỉnh xuất phát $x_0 \in S$, tập đích DICH $\subset S$. Hãy tìm **một** đường đi từ x_0 đến một đỉnh nào đó thuộc tập DICH.*

***- <u>Bài toán 3*</u>**: (bài toán tìm kiếm tối ưu)*

Problem_3*(G = (S, A), x_0, DICH, c) ***(BT3*)***

*Cho đồ thị G = (S, A), với hàm giá c: $A \to R_+$, đỉnh xuất phát $x_0 \in S$ và tập đích DICH $\subset S$. Hãy tìm **một** đường đi từ x_0 đến một đỉnh nào đó thuộc tập DICH* và làm *tối ưu hàm giá.*

Từ tập các cung *A*, ta có thể *xây dựng các thuật toán tìm kiếm lời giải dựa trên toán tử B(x),* $\forall x \in S$. Trong nhiều bài toán thực tế, *B(x) chính là tập các trạng thái kế tiếp hợp lệ từ x. Khi không gian trạng thái S quá lớn,* lời giải $x_{LG} = \{x_0, x_1, ...\}$ thường *được xây dựng dần trong quá trình giải cho đến khi gặp điều kiện kết thúc $P(x_{LG})$ (hay P'(x*): khi tìm được trạng thái mục tiêu $x^* \equiv TrạngTháiCuối(x_{LG}) \in DICH$ ứng với BT3 hoặc thêm một điều kiện nào đó, chẳng hạn tối ưu một hàm mục tiêu hay hàm giá ứng với BT3*).*

** Hai phương pháp xây dựng đồ thị G = (S, A)*

- *PP tường minh*: *tập các nút S và tập các cung A đã biết và được xây dựng trước*. Thông thường, đối với các *đồ thị hữu hạn ứng với các bài toán đơn giản và có kích thước nhỏ* mới có thể biểu diễn tường minh chúng dưới dạng bảng.
- *PP không tường minh*: xuất phát từ đỉnh ban đầu x_0, *trong quá trình tìm kiếm lời giải, xây dựng dần các đỉnh con dựa trên toán tử B(n)* để tạo ra các đỉnh con của *n*. Nghĩa là chỉ khi nào xét đến đỉnh n, tập các đỉnh con *n* mới được xây dựng. Phương pháp này rất có ý nghĩa, đặc biệt là đối với các *đồ thị biểu diễn những bài toán phức tạp và có kích thước lớn*.

Nếu *xét BT3 dưới dạng BT1 thì D là một tập con của tập* $U_{n\geq 1}S^n$.Trong những bài toán *lớn và phức tạp,* do *khó xác định tập D ngay từ đầu* hoặc *có thể xác định D nhưng kích thước của nó quá lớn,* việc dùng các *thuật toán vét cạn V1.a* hay *V1.b là không hiệu quả.* Đối với lớp các bài toán mà *các thành phần của lời giải có thể xác định dần trong quá trình giải,* ta sẽ *cải biên các thuật toán đơn sơ V1* để *thu được dần các thuật toán hiệu quả hơn* trong phần tiếp theo như*: TìmKiếm,* A^T, A^{KT}, *A*, ...*

Trong các thuật toán hay thuật giải tìm kiếm lời giải sau đây, ta dùng tập DONG để lưu các trạng thái đã xét, tập MO dùng để lưu các trạng thái dự định sẽ xét trong các bước kế tiếp.

*b. Phương pháp tìm kiếm lời giải cho BT3 (dưới dạng **lặp**)*

Trong phần này, không có gì khó khăn, ta đưa ra thuật toán tìm kiếm lời giải cho bài toán mở rộng của *BT3* như sau: Cho X_0 ($\subset$ S) là tập những trạng thái có thể xuất phát. Thay vì tìm đường đi từ x_0 đến DICH, ta tìm đường đi mà có thể xuất phát từ một trong các trạng thái $x \in X_0$ đến DICH. Khi đó, để giải *BT3* ta chỉ cần gọi: *TìmKiếm({x_0}, DICH).*

*** *Thuật toán tìm kiếm***: (Giải *BT3'*: *Problem_3'(G = (S, A), X_0, DICH)*)

TìmKiếm(X_0, DICH)

```
{   MO = X0; DONG = ∅; // tập DONG có cấu trúc stack
    if (∃y ∈ X0: P'(y))              // hay if (∃y ∈ MO ∩ DICH)
              {XuấtLờiGiải(y, DONG); Dừng;}
    while (MO ≠ ∅)
    { x = get(MO);                   // lấy một phần tử  x ra khỏi tập MO
      DONG  = DONG U {x};            // đưa x vào tập DONG
      if (∃y ∈ B(x) : P'(y) đúng)    // hay if (∃y ∈ B(x)∩DICH)
              {XuấtLờiGiải(y, DONG); Dừng;}
      else MO = MO U B(x);           // đưa tập đỉnh con B(x) của x vào tập MO
    }
    write ("Không có lời giải");
```

}

Từ đây về sau, để gần hơn với dạng cài đặt thành chương trình máy tính, ta *qui ước*: Remove(x, MO) là thao tác rút phần tử x khỏi đầu hàng đợi MO, Add(x, MO) là thao tác thêm phần tử x vào đuôi hàng đợi MO, AddSet(B(x),MO) là thao tác thêm tập B(x) vào đuôi hàng đợi MO; Push(x, MO) là thao tác thêm phần tử x vào đầu (hay đỉnh) ngăn xếp MO, PushSet(B(x), MO) là thao tác thêm tập B(x) vào đỉnh ngăn xếp MO, Pop(x, MO) là thao tác rút x khỏi đỉnh ngăn xếp MO, x = Top(MO) là hàm trả lại (nhưng không lấy ra khỏi) phần tử ở đầu danh sách MO.

- Thuật toán *XuấtLờiGiải*(y, DONG) cho *cây G (tổng quát hơn cho đồ thị* mà mỗi nút có *không quá một nút cha)*, với tập DONG có cấu trúc stack, dưới dạng đường đi từ x_0 đến y được lưu trong ngăn xếp *LờiGiải* như sau:

```
XuấtLờiGiải(y, DONG)
{ LờiGiải = {y};
  while (not(EmptyStack(DONG)))
  {     Pop(x, DONG);
        // nếu có cung nối từ x đến phần tử ở đỉnh ngăn xếp LờiGiải thì đưa // x vào thành
        phần LờiGiải
         if ((x, Top(LờiGiải)) ∈ A) // hay Top(LờiGiải) ∈ B(x)
                Push(x, LờiGiải);
  }
   ShowStack(LờiGiải); // xuất các phần tử của stack LờiGiải
}
```

Dựa trên thuật toán *TìmKiếm* nhằm tìm lời giải đầu tiên cho bài toán *BT3*, ta có thể cải biên để thu được thuật toán vét cạn dưới dạng lặp *VétCạn để tìm <u>mọi</u> lời giải của bài toán sau (bài tập):*

- <u>Bài toán 4</u>: Problem_4(G = (S, A), x_0, DICH) ***(BT4)***

*Cho đồ thị G = (S, A), với đỉnh xuất phát $x_0 \in S$, tập đích DICH $\subset S$. Hãy tìm **mọi** đường đi từ x_0 đến một đỉnh nào đó thuộc tập DICH.*

***c. <u>Các dạng đặc biệt thường gặp</u>**: Các thuật toán tìm kiếm lời giải trong không gian trạng thái theo chiều rộng, chiều sâu, chiều sâu dần, tìm kiếm cực tiểu giá thành A^T.*

Để *giải BT3 trên cây G*, trong thuật toán *TìmKiếm*, với các *cách chọn cấu trúc dữ liệu khác nhau của tập MO, ta sẽ có các phương pháp tìm kiếm khác nhau*. Sau đây, ta luôn xem *x = Get(MO)* là thao tác *lấy phần tử x ra khỏi đầu* danh sách *MO*.

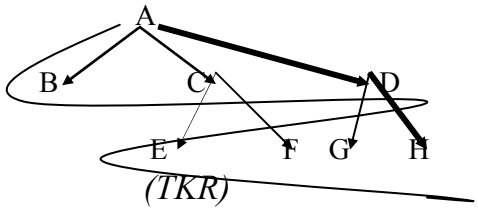
**** Phương pháp tìm kiếm theo chiều rộng (TKR)**:* chính là *thuật toán TìmKiếm,* với tập *MO có cấu trúc hàng đợi (queue)* hay thay MO = MO U B(x) bởi phép toán *AddSet(B(x),MO).*

- *Nhận xét:* Khi G là cây với gốc x_0,

. nếu tồn tại ít nhất một đường đi từ x_o tới tập DICH thì thuật toán TKR dừng và cho ta đường đi p có độ dài ngắn nhất (thậm chí khi G là cây vô hạn);

. nếu không tồn tại đường đi như vậy thì thuật toán dừng nếu và chỉ nếu cây G là hữu hạn.

- *Ví dụ 14*: Xét cây như hình dưới đây với tập DICH = {H}. Thủ tục TKR cho kết qủa là đường đi ADH theo hành trình duyệt các đỉnh ABCDEFGH.



(TKR)

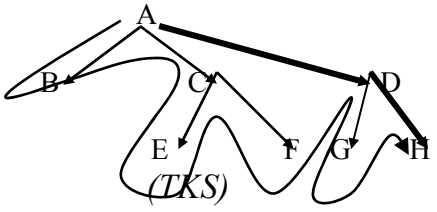
**** Phương pháp tìm kiếm theo chiều sâu (TKS)**:* chính là *thuật toán TìmKiếm,* với tập *MO có cấu trúc ngăn xếp (stack),* hay thay MO = MO U B(x) bởi phép toán *PushSet(B(x),MO)*.

- *Nhận xét:* Khi G là cây với gốc x_0,

. nếu cây *G hữu hạn* thì thủ tục *TKS sẽ dừng* và nếu tồn tại đường đi từ x_0 đến DICH thì sẽ cho kết qủa là một đường đi từ x_o tới tập DICH, khi đó *đường đi* nhận được trong thủ tục TKS *không nhất thiết là đường đi ngắn nhất*. Hơn nữa, nếu cây *G vô hạn*, thủ tục *TKS có thể lặp đến vô hạn, thậm chí trong trường hợp tồn tại đường đi từ x_o đến tập DICH*.

. nếu *không tồn tại đường đi* từ x_0 đến DICH thì thủ tục *TKS chỉ dừng khi cây G hữu hạn*.

- *Ví dụ 15*: Xét cây như hình dưới đây với tập DICH = {D, H}. Thủ tục TKS cho kết qủa là đường đi AD theo hành trình duyệt các đỉnh ABCEFD.



(TKS)

Để *khắc phục tình trạng không dừng của thuật toán TKS* ngay cả khi tồn tại đường đi từ gốc của cây đến DICH, ta đưa vào đại lượng DS đặc trưng cho giới hạn sâu và dùng khái niệm *độ sâu d(x)* của đỉnh x trong cây G = (S, A) được định nghĩa đệ qui như sau:

$d(x_o) = 0$

$d(con) = d(cha) + 1$, nếu $con \in B(cha)$ hay $(cha, con) \in A$.

**** Phương pháp tìm kiếm <u>sâu dần</u> (TKSD)*** *giải BT3 với độ sâu DS = k≥ 1: điều chỉnh lại thuật toán TìmKiếm, với tập MO được trang bị các phép toán cơ bản của cả hàng đợi lẫn ngăn xếp.*

```
TKSD(x0, DICH, k)
{   MO = {x0};
    DONG = ∅;                          // chọn DONG có cấu trúc stack
    DS = k;
    if (P'(x0))                        // hay if (x0 ∈ DICH)
            {XuấtLờiGiải(x0, DONG); Dừng;}
    while (MO ≠ ∅)
    { x = get(MO);                     // lấy phần tử x từ đầu danh sách MO
      Push(x, DONG);                   // đưa x vào tập DONG
      if (∃y ∈ B(x)∩DICH) {XuấtLờiGiải(y, DONG); Dừng;}

      if (d(x) > DS) DS = DS + k;      // khi đó d(x) <= DS
      if (d(x) < DS) PushSet(B(x), MO);// tìm theo chiều sâu
      else AddSet(B(x), MO);           // d(x) = DS: việc tìm được lan theo chiều rộng
    }
    write ("Không có lời giải");
}
```

- <u>Nhận xét</u>: Khi G là cây với gốc x_0,

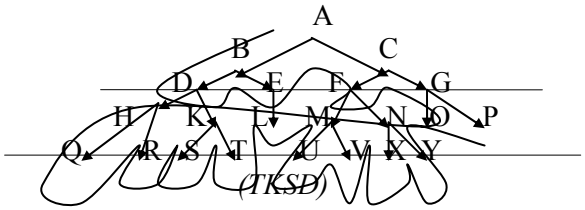
. với *k = 1, thủ tục TKSD trở thành thủ tục TKR*. Khi *k rất lớn* (chẳng hạn k ≥ chiều cao của cây G hay $k \to \infty$ đối với cây vô hạn) thủ tục *TKSD sẽ giống* hay *gần giống tương ứng như thủ tục TKS*;

. với k ≥ 2 thủ tục TKSD *tìm kiếm theo chiều sâu* đối với các đỉnh *có độ sâu nằm trong khoảng từ tk đến (t+1)k* với t = 0, 1, 2, ...;

. nếu *tồn tại ít nhất một đường đi* từ gốc tới DICH thì thủ tục TKSD sẽ *dừng* và cho kết qủa là đường đi *có độ dài khác đường đi ngắn nhất không quá k-1;*

. nếu trong cây *không tồn tại đường đi* như vậy thì thủ tục TKSD *dừng khi và chỉ khi cây G là hữu hạn*.

- *Ví dụ 16*: Xét thủ tục TKSD trên cây như hình dưới đây, với độ sâu k = 2, thứ tự duyệt các đỉnh là : ABDECFG__HQRKSTLMUVNXYOP



**** Phương pháp tìm kiếm cực tiểu A^T:*** Giải *bài toán tối ưu BT3** trên cây G - *Problem_3*(G = (S, A), x_0, DICH, **c**)* - với tiêu chuẩn tối ưu: *tìm một đường đi* từ x_0 đến một trong các trạng thái thuộc tập DICH *có hàm giá đường đi g^0 cực tiểu*. Trong đó, hàm giá đường đi g^0 được xác định theo kiểu đệ qui như sau:

$g^0(x_0) = 0$

$g^0(con) = g^0(cha) + c(cha, con), \ \forall \ (cha, con) \in A$.

Thuật toán sau đây tương tự như thuật toán *TìmKiếm*, trong đó: tập DONG có cấu trúc stack và tập MO chứa các phần tử $(x, g^0(x))$, thao tác *lấy ra một phần tử* được thực hiện *ở đầu* danh sách MO, việc đưa một phần tử x vào danh sách MO *ChènTăng*($(x, g^0(x))$, MO) được thực hiện theo kiểu *chèn $(x, g^0(x))$ vào MO tăng dần theo $g^0()$ từ đầu đến cuối* danh sách MO.

```
A^T(x0, DICH)
{   DONG = ∅;                          // DONG có cấu trúc stack
    g^0(x0)=0; MO = {(x0, g^0(x0))};
    while (MO ≠ ∅)
    { Pop((x, g^0(x)), MO);            // trong MO, x ở đỉnh và có g^0(x) nhỏ nhất !
      if (x ∈ DICH) {XuấtLờiGiải(x, DONG); Dừng; }
      Push(x, DONG);                   // đưa x vào đỉnh stack DONG
    //ChènTăng(y,MO) ∀y∈B(x)={(con,g^0(con)):(x,con)∈A & g^0(con)=g^0(x)+c(x, con)}
      ChènTăngTập(B(x), MO);
    }
    write ("Không có lời giải");
}
```

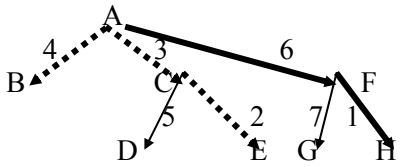
- *Nhận xét*:

. Thủ tục *TKR là trường hợp riêng của thủ tục A^T* khi hàm giá c≡1.

. Thủ tục *TKS cũng là trường hợp riêng của thủ tục A^T* khi: hoặc thay thao tác *ChènTăngTập*(B(x), MO) bởi *ChènGiảmTập*(B(x), MO); hoặc xem hàm giá c: $A \to R$ và $c \equiv -1$.

. Nếu trong cây G *tồn tại đường đi p: $x_o \rightarrow DICH$* thì *thủ tục A^T sẽ dừng* và cho kết qủa là đường đi p_o sao cho $c(p_o) \rightarrow$ min, nếu: hoặc cây *G hữu hạn*, hoặc cây *G vô hạn nhưng tổng giá theo mọi nhánh vô hạn phải phân kỳ hoặc hội tụ về một giá trị lớn hơn giá tối ưu* !

- *Ví dụ 17*: Xét cây và giá các cung được cho như hình dưới đây và tập DICH = {D, H}. Thủ tục tìm kiếm cực tiểu A^T cho kết qủa là đường đi AFH.



(Thuật toán A^T)

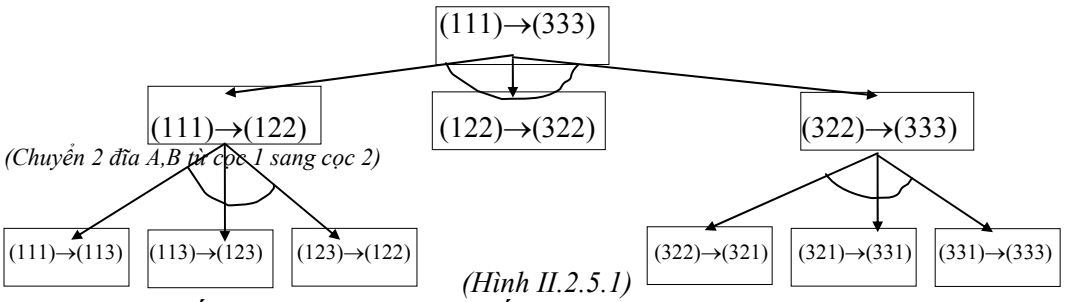
- *Chú ý:* Trong thuật toán $\mathbf{A^T}$, ta *chọn phương án tối ưu nhất* trong *tất cả những phương án đã đi qua (được lưu lại để quay lui khi lạc hướng)*. Chính vì thế, với các bài toán lớn, có thể *xảy ra tình trạng nhanh tràn bộ nhớ* nếu các bước kế tiếp cần lưu quá nhiều; khi đó ta có thể *dùng heuristic* sau*: thay vì lưu hết 100% các phần tử B(x) vào tập MO, ta chỉ cần lưu p%, $1 \leq p \leq 100$,* mà thôi ! *Giá trị của p phụ thuộc vào từng bài toán cụ thể* và *tài nguyên máy tính hiện có.*

II.2.5. Quy bài toán về bài toán con và các chiến lược tìm kiếm trên đồ thị VÀ/HOẶC

a. Qui bài toán về bài toán con

* *Ý tưởng*: dựa trên ý tưởng của phương pháp chia để trị, tách bài toán lớn và phức tạp ban đầu thành những bài toán con nhỏ, đơn giản tới mức sơ cấp (lời giải của chúng đã biết).

- *Ví dụ 18*: Xét bài toán tháp Hà nội với n = 3 và dùng cách biểu diễn như trong ví dụ 2 phần 2.2.4.a. Ta *qui bài toán về các bài toán con dạng VÀ* như sau:



(Hình II.2.5.1)

Liên kết những lời giải sơ cấp trên từ trái sang phải, ta có lời giải của bài toán ban đầu:

(111)→(113)→(123)→ (122)→(322) →(321)→(331)→(333).

Cây bình thường là cây HOẶC, dạng đặc biệt của cây VÀ/HOẶC. Cây VÀ/HOẶC thường được dùng để *biểu diễn quá trình chia bài toán lớn thành nhiều nhóm các bài toán con được nối kết lôgic với nhau bởi dãy các phép toán lôgic: and, or*.

b. <u>Biểu diễn bài toán dưới dạng đồ thị VÀ/HOẶC</u>

Đồ thị định hướng VÀ/HOẶC là đồ thị định hướng thông thường G = (S, A) và thêm vào tính chất: với mỗi đỉnh n ∈ S tất cả các đỉnh con của nó B(n) cùng thuộc vào một trong 2 kiểu: đỉnh VÀ hay đỉnh HOẶC. Khi các đỉnh con m của n là đỉnh VÀ thì các cung nối các đỉnh con của nó m ∈ B(n) được nối với nhau bởi ngoặc tròn (như dấu góc tròn trong hình học phẳng). Sự tương ứng giữa quá trình qui bài toán về bài toán con với đồ thị VÀ/HOẶC được cho trong bảng sau:

<u>Qui bài toán về bài toán con</u>	*<u>Đồ thị VÀ/HOẶC</u>*
Bài toán	Đỉnh
Toán tử qui bài toán về bài toán con	Cung
Bài toán ban đầu	Đỉnh gốc (đỉnh xuất phát)
Bài toán sơ cấp	Đỉnh lá
Các bài toán con phụ thuộc	Đỉnh con dạng VÀ
Các bài toán con độc lập	Đỉnh con dạng HOẶC
Lời giải bài toán	Đồ thị con lời giải
Giải bài toán	Tìm đồ thị con lời giải

<u>Định nghĩa đỉnh giải được</u> (đỉnh gđ):

- Các đỉnh lá tương ứng với bài toán con *sơ cấp giải được* (hay *đỉnh kết thúc*) là đỉnh giải được.
- Nếu đỉnh n có các đỉnh con là đỉnh HOẶC thì nó giải được khi và chỉ khi *tồn tại* một đỉnh con của nó giải được.
- Nếu đỉnh n có các đỉnh con là đỉnh VÀ thì nó giải được khi và chỉ khi *mọi* đỉnh con của nó giải được.

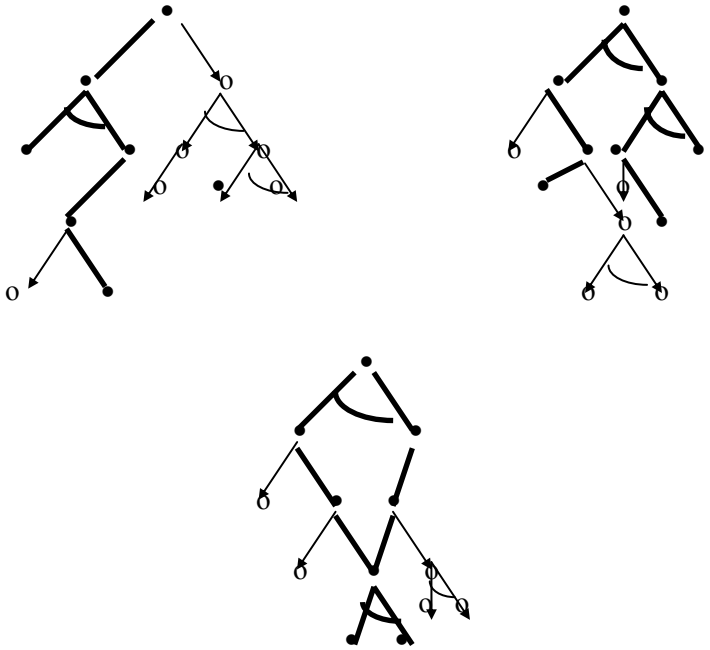
Đồ thị lời giải là đồ thị con của đồ thị VÀ/HOẶC chỉ bao gồm đỉnh xuất phát và các đỉnh giải được liên quan đến nó.

<u>Định nghĩa đỉnh không giải được</u> (đỉnh kgđ):

- Các đỉnh lá tương ứng với bài toán con *sơ cấp không giải được* là đỉnh không giải được.
- Nếu đỉnh có các đỉnh con là đỉnh VÀ thì nó là không giải được khi và chỉ khi *tồn tại* một đỉnh con của nó không giải được.
- Nếu đỉnh có các đỉnh con là đỉnh HOẶC thì nó là không giải được khi và chỉ khi *mọi* đỉnh con của nó không giải được.

- *<u>Ví dụ 19</u>*: Trên hình II.2.5.2 là các đồ thị VÀ/HOẶC, trong đó các đỉnh giải được tương ứng với các hình tròn tô đậm và các cung trong đồ thị con lời giải cũng được tô đậm, các đỉnh không giải được được đánh dấu bởi vòng tròn.

- <u>*Nhận xét*</u>: Khi đồ thị VÀ/HOẶC *không có đỉnh VÀ* thì nó trở thành *đồ thị thông thường*. Lúc đó *đồ thị lời giải suy biến thành một đường đi* từ đỉnh xuất phát đến một đỉnh kết thúc nào đó. *Mục đích của quá trình tìm kiếm trên đồ thị VÀ/HOẶC là chứng tỏ đỉnh xuất phát có thể giải được hay không giải được theo hướng ngược từ các đỉnh lá đến đỉnh xuất phát* và trong trường hợp khẳng định thì *chỉ ra đồ thị con lời giải (*có thể thỏa mãn thêm một tính chất nào đó).



(Hình II.2.5.2)

c. <u>Các phương pháp tìm kiếm trên cây VÀ/HOẶC</u> *(theo chiều rộng, chiều sâu, cực tiểu giá thành)*

Để đơn giản, ta *chỉ xét các phương pháp tìm kiếm trên cây VÀ/HOẶC*. Tùy theo *phương pháp (PP) lựa chọn thứ tự các đỉnh sẽ xét* mà ta có các phương pháp tìm kiếm khác nhau theo: chiều sâu, chiều rộng, tìm kiếm cây lời giải có giá thành nhỏ nhất, ...

Sự *khác biệt chủ yếu của các PP tìm kiếm trên đồ thị hay cây VÀ/HOẶC với các PP tìm kiếm trong không gian trạng thái* ở mục trước là việc *kiểm tra tính giải được của đỉnh xuất phát* (xem nó giải được hay không giải được) và các PP sắp xếp, lựa chọn đỉnh để mở phức tạp hơn nhiều. Với mỗi đỉnh n, ta sẽ cần dùng đến 2 thủ tục gán nhãn giải được *gđ(n* $\in$ *S)* hay gán nhãn không giải được *Kgđ(n* $\in$ *S)*.

Với mỗi đỉnh n ∈ S, ta dùng các ký hiệu:

- Nếu n là đỉnh kết thúc thì kt(n) = true, ngược lại kt(n) = false.

-Nhãn(n) = "*gđ*" , khi n là đỉnh giải được
"*kgđ*" , khi n là đỉnh không giải được
"*kxđ*" , (không xác định) nếu n chưa đủ thông tin để quyết định hoặc n chưa được xét tới

Chú ý rằng, một *đỉnh không phải là giải được* không nhất thiết là không giải được mà còn có thể là không xác định !

- KiểuVÀ(n)=true nếu các đỉnh con của n là đỉnh VÀ và nhận trị false nếu ngược lại.

- Để đơn giản hoá trong các thuật toán sẽ trình bày sau đây, ngay từ đầu, ta sẽ gán nhãn:

. "*gđ*" : đối với các đỉnh lá giải được
. "*kgđ*" : đối với các đỉnh lá không giải được
. "*kxđ*" : đối với mọi đỉnh còn lại

* ***Thủ tục gán nhãn giải được***

gđ(n ∈ S)

```
{1 if (nhãn(n) = "kxđ") then                // do đó B(n) ≠ ∅
      if (n ∈ MO U DONG) then               // nếu n ∉ MO U DONG: chưa xét n
            if (KiểuVÀ(n))
            then {2 bien = true;
                  while (B(n) ≠∅ and bien) do
                  {     con ← get(B(n));
                        gđ(con); bien = (nhãn(con) == "gđ");
                  }
                  if (bien) then      // mọi đỉnh con dạng VÀ của n đều gđ
                        nhãn(n) = "gđ"
                 }2
            else {3 bien = false;
                  repeat { con ← get(B(n));
                           gđ(con); bien = (nhãn(con) == "gđ")
                         }
                  until (bien or B(n) == ∅);
                  if (bien) then      // có một đỉnh con dạng HOẶC của n gđ
                        nhãn(n) = "gđ"
                 }3
}1
```

Tương tự, có thể viết thủ tục *không giải được* ***Kgđ***(x ∈ *S) (bài tập).*

*** *Dạng chung của các thủ tục tìm kiếm trên cây VÀ/HOẶC***

Input : Cây VÀ/HOẶC G = (S,A) với gốc x_o.

Output: Thông báo "Không thành công" nếu x_o *không giải được* và "thành công" nếu x_o *giải được* và khi đó xuất cây lời giải.

Tính *gđ* hay *kgđ* của *gốc được lan truyền ngược từ lá.* Trong các thuật toán của phần này, ta sử dụng tập *DONG* có cấu trúc stack.

```
TìmKiếm_VÀ_HOẶC(x_o, DONG)
{1 MO = {x_o}; DONG = ∅;
  while (MO ≠ ∅) do
  {2 n ← get(MO);                          // lấy n ra khỏi đầu danh sách MO
   Push(n, DONG);
   if (B(n) ≠∅)                            // đỉnh n có con
   then {3 MO ← MO U B(n);
           bool = false;
           while (B(n) ≠ ∅ and not bool) do
           { con ← get(B(n)); bool = (nhãn(con) == "gđ");
           }
         if (bool) then   // nếu có ít nhất 1 con gđ thì xem lại gốc có gđ không ?
                { gđ(x_o);
                  if (nhãn(x_o) == "gđ")
                  then {  write("Thành công");
                          XuấtCâyLờiGiải(x_0, DONG); exit;
                       }
                  else Loại khỏi MO các đỉnh có tổ tiên là đỉnh giải được
                }
        }3
   else                                              // n là đỉnh lá
         if (kt(n))              // n là đỉnh lá gđ, xem lại gốc có gđ không ?
         then { gđ(x_o);
                  if (nhãn(x_o) == "gđ")
                  then {  write("Thành công");
                          XuấtCâyLờiGiải(x_0, DONG); exit;
                       }
                  else Loại khỏi MO các đỉnh có tổ tiên là đỉnh giải được
                }
         else                    // n là đỉnh lá kgđ, xem lại gốc có kgđ không ?
               { Kgđ(x_o);
                  if (nhãn(x_o) == "Kgđ")
                  then exit("Không thành công")
```

else *Loại khỏi MO các đỉnh có tổ tiên là đỉnh không giải được*
}
}2
}1

* ***Thuật toán*** *tìm kiếm theo chiều rộng trên cây VÀ/HOẶC*

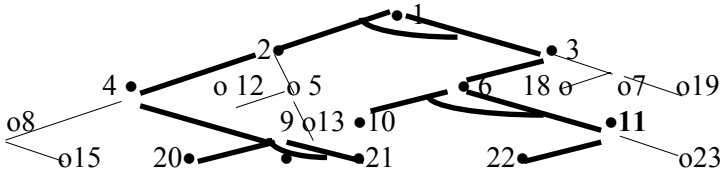
Chính là thuật toán *TìmKiếm_VÀ_HOẶC(x_o, DONG),* với cấu trúc tập *MO* là *hàng đợi.*

Nếu cây lời giải tồn tại thì thủ tục tìm kiếm theo chiều rộng sẽ dừng và cho kết qủa là cây lời giải có độ cao nhỏ nhất.

- *Ví dụ 20*: Xét cây VÀ/HOẶC như trên hình *II.2.5.3*. Thứ tự đưa các nút vào danh sách DONG được chỉ ra bởi các số ở bên cạnh các đỉnh. Các đỉnh tô đậm là các đỉnh giải được. Cây lời giải được phân biệt bởi các cung tô đậm.

Quá trình đưa các đỉnh vào tập MO và DONG như sau:

MO	DONG
1	
2,3	1
3;4,5	1,2
4,5;6,7	1,2,3
5;6,7;8,9	1,2,3,4
6,7;8,9;12,13	1,2,3,4,5
7;8,9;12,13;10,11	1,2,3,4,5,6 (đỉnh 2,3: "gđ")
	(Đỉnh gốc 1 : "gđ" → Kết thúc)



(Hình II.2.5.3)

* ***Thuật toán*** *tìm kiếm theo chiều sâu trên cây VÀ/HOẶC*

Chính là thuật toán *TìmKiếm_VÀ_HOẶC(x_o, DONG),* với cấu trúc tập *MO* là *ngăn xếp.*

* ***Thuật toán*** *tìm kiếm cực tiểu giá thành trên cây VÀ/HOẶC*

Cho cây VÀ/HOẶC $G = (S, A)$ với gốc x_0 và hàm giá c: $A \rightarrow R_+$. Để tìm cây lời giải có giá cực tiểu, ta cần đến khái niệm hàm giá tối ưu *h(n)* của cây lời giải có gốc là đỉnh *n* bất kỳ và hàm giá ước lượng $h^0(n)$ của nó.

Định nghĩa: hàm giá tối ưu h(n) của cây lời giải có gốc n được xây dựng như sau:

- Nếu n là đỉnh lá và

. giải được (đỉnh kết thúc) thì $h(n) = 0$.

. không giải được thì hàm h(n) không xác định

- Nếu n có các đỉnh con $n_1, ..., n_k$ là:

. đỉnh HOẶC thì: $h(n) = \min (c(n, n_i) + h(n_i))$.

. đỉnh VÀ thì:

$$h(n) = \sum_{i=1}^{k} (c(n, n_i) + h(n_i)), \text{ với kiểu giá tổng cộng}$$

$$h(n) = \max_{1 \le i \le n} (c(n, n_i) + h(n_i)), \text{ với kiểu giá cực đại}$$

(Lưu ý rằng h(n) không xác định đối với các đỉnh không giải được)

Mục đích của việc tìm kiếm là xây dựng cây lời giải có giá cực tiểu $h^0(n_o)$.

- Trên thực tế, trong thuật toán ta *cần xác định và sử dụng hàm ước lượng heuristics h^o đối với các đỉnh không phải là đỉnh không giải được*. *Cây lời giải được xây dựng dần* trong quá trình mở rộng cây lựa chọn, tại mỗi thời điểm các nút lá của nó thuộc một trong 3 dạng sau:

a. Các đỉnh lá giải được (đỉnh kết thúc).

b. Các đỉnh lá không giải được (đỉnh không kết thúc và không có con).

c. Các đỉnh chưa được tháo (chưa được xử lý).

Mỗi đỉnh này của cây lựa chọn gọi là <u>*đỉnh mút*</u>.

- Ta <u>*xây dựng ước lượng h^o*</u> của h như sau: (1)

1. n là đỉnh mút

- Nếu n là đỉnh lá giải được thì $h^o(n) = 0$.
- Nếu n là đỉnh lá không giải được thì $h^o(n)$ không xác định.
- Nếu n chưa được xử lý thì để làm giá trị $h^o(n)$ có thể lấy một ước lượng heuristic nào đó của h(n). Ước lượng này dựa trên các thông tin heuristics về bài toán.

2. n không là đỉnh mút: nếu n có các đỉnh con $n_1,..., n_k$ là:

- đỉnh HOẶC thì: $h^o(n) = \min (c(n, n_i) + h^o(n_i))$.
- đỉnh VÀ thì:

$$h^o(n) = \sum_{i=1}^{k} (c(n, n_i) + h^o(n_i)), \text{ với kiểu giá tổng cộng}$$

$$h^o(n) = \max_{1 \le i \le n} (c(n, n_i) + h^o(n_i)), \text{ với kiểu giá cực đại}$$

- Trong cây tìm kiếm, ở mỗi bước có thể chứa một tập các cây con có gốc x_o sao cho chúng trở thành phần trên của cây lời giải đầy đủ. Ta gọi các cây này là cây lời giải tiềm tàng gốc x_o. Dựa vào h^o có thể xây dựng một <u>*cây lời giải tiềm tàng T_o gốc x_o*</u> có nhiều triển vọng trở thành phần trên của cây lời giải thật sự gốc x_o như sau: (2)

1. Đỉnh xuất phát $x_o \in T_o$.

2. Nếu $n \in T_o$ có các đỉnh con $n_1,..., n_k$ là đỉnh :

a. HOẶC thì chọn một con n_{io} đưa vào T_o sao cho: $c(n,n_{io})+h^o(n_{io}) \rightarrow \min$ và *nhãn*$(n_{io}) \neq$ "*kgđ*".

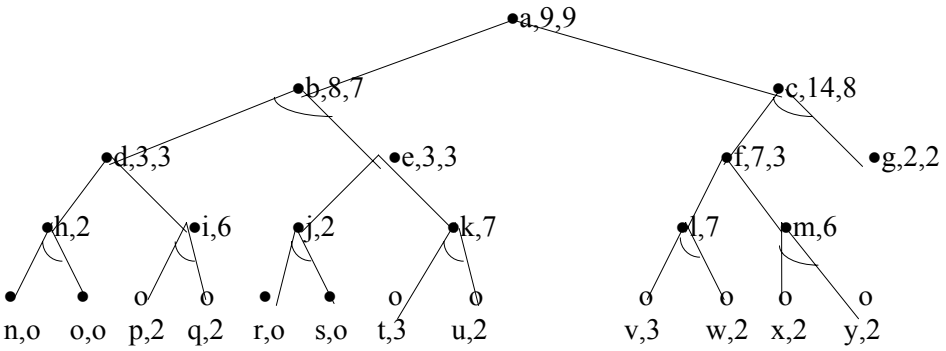
b. VÀ thì đưa tất cả $n_1,..., n_k$ vào T_o nếu *nhãn*$(n_i) \neq$ "*kgđ*" $\forall$ i = 1, .., k

TìmKiếm_VÀ_HOẶC_CựcTiểu

```
{1 MO = {xo}; To ← xo ;
  while (MO ≠∅) do
  {2 n ← Get(MO);                    // lấy n ra khỏi đầu danh sách MO
    DONG ← {n} U DONG;
   if (kt(n))                                          // n là đỉnh lá gđ
     then {      gđ(xo);
```

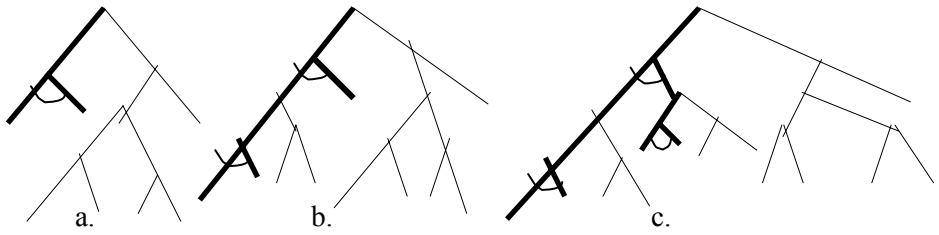
```
                if (nhãn(x_o) == "gđ")
                then {   Xuất cây lời giải tối ưu T_o; exit;
                     }
                else Loại khỏi MO các đỉnh có tổ tiên là đỉnh giải được
            }
    else        if (B(n) ≠∅)                                  // n không là đỉnh lá
                then { for each m ∈ B(n) do Tính h^o(m);       // theo (1)
                         for each m ∈ MO U DONG do Tính h^o(m);
                         MO←MO U B(n); // ChènTăngTập(B(n), MO) theo h^0(n)
                         xd(T_o);  // Xây dựng cây T_o dựa trên n và B(n) theo (2)
                     }
                else                                          // n là đỉnh lá Kgđ
                     { Kgđ(x_o);
                       if (nhãn(x_o) == "kgđ")
                       then exit("Không thành công")
                       else Loại khỏi MO các đỉnh có tổ tiên là đỉnh không giải được
                     }
    }2
}1
```

- *Ví dụ 21*: Xét cây VÀ/HOẶC như hình *II.2.5.4*.



(Hình II.2.5.4)

Các số bên cạnh đỉnh là giá trị h và h^o. Đối với các nút mút trong quá trình tìm kiếm các giá trị h^o là các đánh giá heuristics giá tổng cộng của bài toán tương ứng. Lấy $h^o = h$ đối với các nút ở độ sâu 3, 4 (các nút từ h đến y), $h^o(d) = 3$, $h^o(e) = 3$, $h^o(f) = 3$, $h^o(g) = 2$, $h^o(b) = 7$, $h^o(c) = 8$, $h^o(a) = 9$. Quá trình tìm kiếm được chỉ ra trên hình *II.2.5.5 và II.2.5.6*. Cây lời giải tiềm tàng chỉ ra bởi các cung tô đậm.



(Hình II.2.5.5)

Vấn đề cần đặt ra là chọn đỉnh nào trong các đỉnh VÀ của T_o để xử lý. Có thể chọn các đỉnh mút trong T_o có xác suất lớn nhất sao cho có thể dẫn đến bác bỏ giả thuyết rằng T_o sẽ là phần trên của cây lời giải có giá tối ưu. Với giá tổng cộng thì đỉnh có xác suất lớn nhất chứng tỏ sự không phù hợp của T_o là đỉnh có giá trị h^o lớn nhất. Với giá cực đại ta chọn đỉnh MO trong T_o như sau: Xuất phát từ x_o đi trong cây T_o, nếu gặp một đỉnh n nào đó có đỉnh con là đỉnh VÀ thì chọn cung từ n tới đỉnh con n_i sao cho $c(n,n_i) + h^o(n_i) \rightarrow \max$.

Quá trình đưa các đỉnh vào tập MO và DONG như sau:

T_o	MO		DONG
(a,9) /* (a,h^o) */ (a,9)		(a,9)	
(b,7)	(b,7),(c,8);	min	(b,7)
(d,3) ,(e,3)	(d,3) ,(e,3);(c,8);	∀	(d,3)
(h,2)	(h,2),(i,6);(e,3);(c,8);	min	(h,2)
(n,0),(o,0)	(n,0),(o,0);(i,6);(e,3);(c,8)	∀	(n,0),(o,0)
(e,3)	(e,3),(c,8)		(e,3)
(j,2)	(j,2),(k,7);(c,8)	min	(j,2)
(r,0),(s,0)	(r,0),(s,0);(k,7);(c,8)	∀	(r,0),(s,0)
	(→ Kết thúc)		

(Hình II.2.5.6)

II.2.6. Phương pháp GPS (General Problem Solving)

Phương pháp này còn gọi là phương pháp *mục đích - phương tiện* (Ends-Means). PP GPS là sự *tổng quát hóa* của những PP giải quyết vấn đề đối với bài toán biểu diễn vấn đề trong không gian trạng thái đã trình bày.

- *Ý tưởng:* Giả sử bài toán được biểu diễn trong không gian trạng thái S và cần phải đưa ra cách biến đổi từ trạng thái ban đầu $x_o \in S$ về trạng thái cuối $x^* \in S$. Trong các PP dựa trên lôgic hình thức quá trình mở rộng cây tìm kiếm dựa trên các thông tin định lượng: số phép biến đổi đã sử dụng và ước lượng heuristics h^o phản ánh mối liên hệ giữa x_o và x*. Trong khi đó , GPS quan tâm đến *sự khác biệt δ_i giữa x_o và x** (theo nghĩa *định tính lẫn định lượng*) và mối liên hệ giữa chúng với các phép biến đổi trạng thái o_j. Như vậy, giả sử xuất phát từ x_o ta đã đi tới trạng thái x. Xem xét một *khác biệt "quan trọng nhất" δ_i giữa x và x*,* dựa vào đó xác định phép biến đổi o_j nào đó *"hiệu qủa nhất" để biến đổi x sao cho giảm sự khác biệt δ_i nhiều nhất*. Kết qủa nhận được trạng thái mới $o_j(x)$ và cứ như vậy tiếp tục cho đến khi $x \equiv x^*$.

- *Ba yếu tố cơ bản trong PP GPS*:

a. Xác định *không gian trạng thái S, trạng thái đầu $x_o \in S$, trạng thái cuối $x^* \in S$ và tập các phép biến đổi* O từ trạng thái này sang trạng thái khác:

$$O = \{ o_j \mid o_j: S \rightarrow S, j = 1..n \}$$

b. Xác định *tập Δ các kiểu khác biệt* giữa các trạng thái trong không gian

$$\Delta = \{ \delta_i \mid \delta_i : S^2 \rightarrow R, i = 1..m \}$$

c. Xây dựng *ma trận M* với các cột ứng với các toán tử, các hàng ứng với *các sự khác biệt có thể có*

$M = (m_{ij})_{m \times n}$, trong đó:

$$m_{ij} = \begin{cases} 1, \text{ nếu phép biến đổi } o_j \text{ làm giảm sự khác biệt } \delta_i \\ 0, \text{ nếu ngược lại} \end{cases}$$

*(*Có thể cho m_{ij} *những trị khác để phản ánh tốt hơn tác dụng của toán tử* o_j *đến sự khác biệt* δ_i*).*

<u>*Bài toán*</u>

Input: - Tập S các trạng thái, trạng thái đầu x_o, trạng thái cuối x*
- Tập $\Delta = \{ \delta_1, ..., \delta_m \}$ các sự khác biệt
- Tập $O = \{ o_1, ..., o_n \}$ các toán tử
- Ma trận $M = (m_{ij})_{m \times n}$

Output: Ra thông báo "Thành công" nếu đưa được x_o về x*

Thuật giải GPS

```
{ x = x_o; OP = ∅;
  while (x ≠ x*) do
  { D = match(x, x*) ; // D = { δ_i1, ..., δ_ik} là tập sự khác biệt giữa x và x*
    δ ← get(D);          // chọn ra sự khác biệt quan trọng nhất δ = δ_i ∈ D
    for j=1 to n do
        if (m_ij = 1) then OP = OP U {o_j}; // OP ={o_j | m_ij=1}
    o ← get(OP); // chọn toán tử o = o_j ∈ OP làm giảm sự khác biệt δ_i nhiều nhất
    x = o(x);
  }
  if (x = x*) Xuất lời giải;
}
```

Ở đây thủ tục *match(x, x*) cho phép xác định những điểm khác biệt giữa x và x**. Chẳng hạn, sự khác biệt giữa đỉnh x và x* trong đồ thị biểu diễn không gian trạng thái S có thể là độ dài đường đi từ x tới x*.

- <u>*Ví dụ 22*</u>: Cần phải chứng minh sự tương đương logic giữa hai biểu thức mệnh đề:

$$R \wedge (\neg P \Rightarrow Q) \equiv (Q \vee P) \wedge R$$

Áp dụng phương pháp GPS để giải bài toán trên.

. Trong không gian biểu diễn các mệnh đề lôgic xét *4 kiểu khác biệt* sau:

ΔT: Có sự khác biệt về dấu $\neg$ (phủ định) trong biểu thức.

ΔC: Khác nhau về số các phép toán ($\wedge, \vee, \Rightarrow$).

ΔG: Khác nhau về phương pháp nhóm các mệnh đề.

ΔP: Khác nhau về vị trí các thành phần trong hai biểu thức.

. Trong bài toán này, ta lấy c*ác toán tử biến đổi trạng thái là 3 phép biến đổi tương đương* sau:

R1: $A \vee B \Leftrightarrow B \vee A$; $A \wedge B \Leftrightarrow B \wedge A$

R2: $A \rightarrow B \Leftrightarrow \neg A \vee B$

R3: $\neg\neg A \Leftrightarrow A$

. Xét ma trận M sao cho các cột tương ứng với các phép biến đổi : R1-R3.

	R1	R2	R3
ΔT		1	1
ΔC		1	
ΔG			
ΔP	1		

Đặt : $L1 \equiv R \wedge (\neg P \Rightarrow Q)$, $Lo \equiv (Q \vee P) \wedge R$

Đích 1: Biến đổi L1 về Lo

Sự khác biệt: D = {ΔT, ΔP, ΔC}. Với ΔP (ΔP = get(D)), OP = {R1}: chọn R1 áp dụng vào L1 ta được: $L2 = (\neg P \Rightarrow Q) \wedge R$.

Đích 2: Biến đổi L2 về Lo

Sự khác biệt: D = {ΔT, ΔC, ΔP}. Với ΔC (ΔC = get(D)), OP = {R2}: chọn R2 áp dụng vào L2 ta được: $L3 = (\neg\neg P \vee Q) \wedge R$.

Đích 3: Biến đổi L3 về Lo

Sự khác biệt: D = {ΔT, ΔP}. Với ΔP (ΔP = get(D)), OP = {R1}: chọn R1 áp dụng vào L3 ta được: $L4 = (Q \vee \neg\neg P) \wedge R$.

Đích 4: Biến đổi L4 về Lo

Sự khác biệt: D = {ΔT}. Với ΔT (ΔT = get(D)), OP = {R2, R3}: ta chọn R3 mới làm giảm sự khác biệt ΔT của L4, ta được: $L5 = (Q \vee P) \wedge R \equiv Lo$.

Bài toán được chứng minh.

- Ngoài ra, nếu qui bài toán về dạng:

$[R \wedge (\neg P \rightarrow Q)] \Rightarrow [(Q \vee P) \wedge R]$

và $[(Q \vee P) \wedge R] \Rightarrow [R \wedge (\neg P \Rightarrow Q)]$

thì ta có thể giải bài toán bằng: thủ tục tách Wong hay thủ tục hợp giải Robinson sẽ được trình bày trong chương 3.

II.3. Kỹ thuật Heuristic

Những bài toán, đặc biệt là khi chúng có kích thước lớn và có cấu trúc phức tạp, đã có thuật toán khả thi với độ phức tạp không quá đa thức, thường rất hẹp.

- <u>*Hạn chế của thuật toán*</u>: Trên thực tế, ta thường gặp nhiều bài toán:

. hoặc cho đến nay *chưa có thuật toán nào để giải*;

. hoặc đã *có thuật toán để giải nhưng không khả thi* về không gian nhớ và thời gian do độ phức tạp của nó có cấp vượt quá đa thức;

. hoặc *có các phương pháp giải mặc dù được thực tế chấp nhận nhưng lại vi phạm một số tính chất của thuật toán như tính xác định hay tính đúng*. Chẳng

hạn, để chuyển từ bước sơ cấp này đến bước sơ cấp sau đôi khi lại cần bổ sung thêm thông tin kinh nghiệm (rút ra trong quá trình tìm kiếm lời giải thực tế) đặc thù của bài toán mới quyết định được. Những bài toán liên quan đến số thực được giải trên máy tính, nếu chấp nhận lời giải gần đúng hoặc gần tối ưu thì có thể tồn tại nhiều cách giải đơn giản và hiệu quả hơn.

Để giải quyết khó khăn này người ta đã *mở rộng* tính *xác định, tính đúng* của thuật toán và bổ sung *thêm các tri thức kinh nghiệm* đặc thù hay *mẹo giải* để có được các *thuật giải heuristics*.

- *Các đặc trưng của thuật giải heuristic*: *thường* tìm được lời giải *tốt* (không chắc luôn luôn là tốt nhất) hay *gần tốt* một cách *đơn giản, nhanh chóng* và đôi khi độc đáo (so với những thuật toán tương ứng giải cùng một bài toán). Có *2 cách đưa các thông tin heuristics đặc tả vào các thủ tục tìm kiếm:*

. các tri thức được nạp ngay trong biểu diễn của bài toán;

. các hàm đánh giá heuristics nhằm lượng hoá cấp độ của khả năng lựa chọn việc thực hiện.

- *Các kỹ thuật heuristics liên hệ chặt chẽ với chiến lược điều khiển các hướng tìm kiếm lời giải và xử lý cạnh tranh*; chúng được chia thành 2 lớp chính:

. *Định tính*: dưới dạng các luật *nếu ... thì*

. *Định lượng*: dưới dạng các hàm ước lượng heuristics h^o. Các *hàm đánh giá được xem là công cụ có rất hiệu quả* khi sử dụng kỹ thuật heuristic. Hai vấn đề quan trọng trong lập trình heuristic có sử dụng các hàm đánh giá là: *tách những dấu hiệu có ích và tổ hợp các dấu hiệu này để thể hiện thành những hàm đánh giá tốt*. Một kỹ thuật heuristics được coi là hợp lý khi nó *cho phép tiến hành đánh giá các khả năng để làm rõ khả năng nào tốt hơn các khả năng còn lại*.

- *Hàm số heuristics*

Dạng tổng quát của các hàm đánh giá: $f(p) = f(t_1, ..., t_n)$, trong đó ứng với mỗi khả năng p, dạng giải tích của hàm đánh giá f phụ thuộc vào các tham số $t_1, ..., t_n$ và trả về một giá trị $f(t_1, ..., t_n)$ đặc trưng cho khả năng p.

. Trường hợp đơn giản nhất (*dạng afin*): $\boldsymbol{f(t_1, ..., t_n) = \sum a_i.t_i}$, với a_i là các hệ số đặc trưng cho trọng số của các tham số t_i.

. Phức tạp hơn là dạng toàn phương:

$$\mathbf{f(t_1, ..., t_n) = \sum_{i=1}^{n} a_i.t_i + \sum_{j=1}^{n} b_i.t_i^2 + \sum_{i \neq j=1}^{n} c_{ij}.t_i.t_j}$$

- *Quá trình xây dựng hàm heuristic gồm 3 vấn đề chính* sau:

. Xác định các dấu hiệu đặc tả s_i của bài toán;

. Với mỗi khả năng p, các dấu hiệu đặc tả s_i được lượng hoá bởi các giá trị t_i. Xác định dạng của hàm đánh giá $f(p) = f(t_1, ..., t_n)$ dựa trên các giá trị đặc trưng cho các thuộc tính t_i và các tham số;

. Xác định giá trị của các tham số cho từng bài toán cụ thể.

II.3.1. Các thuật giải tìm kiếm tối ưu trên cây và đồ thị với tri thức heuristic

Problem_3**(G = (S, A), x_0, DICH, c, h^0) ***(BT3**)***

a. <u>Thuật giải A^{KT}</u> (Algorithm for Knowledge Tree search) giải *bài toán* ***tối ưu*** *BT3** với tri thức bổ sung* ***heuristic*** *trên* ***cây***

Tương tự thuật toán A^T tìm kiếm cực tiểu trên cây G = (S, A) có gốc x_0 (phần II.2.4.c), nhưng thay hàm $g^0(x)$ bởi $f^0(x) = g^0(x) + h^0(x)$.

- <u>Ý nghĩa</u>:

. $g^0(x)$: *giá chi phí thật sự từ x_0 đến x,* cách tính $g^0(x)$ theo công thức đệ qui: $g^0(x_0) = 0$ & $g^0(con) = g^0(cha) + c(cha, con)$ $\forall$ (cha, con) $\in$ A.

. $h^0(x)$: *ước lượng chi phí từ x đến tập DICH*, hay là *hàm ước lượng khả năng dẫn đến lời giải.*

. Việc dùng hàm $f^0(x) = g^0(x) + h^0(x)$ *khi chọn lựa phương án kế tiếp, không chỉ quan tâm đến chi phí đã trả từ điểm xuất phát x_0 đến trạng thái hiện tại x, mà còn tham khảo thêm cả thông tin heuristic đặc trưng cho khả năng nhanh dẫn đến đích cuối cùng từ trạng thái hiện tại*. Có thể xem các phương pháp này là các *thuật giải vét cạn thông minh*. Đây là cách giải thông minh và độc đáo mà con người thường sử dụng khi gặp các bài toán khó trong thực tế như chứng minh định lý, giải các bài toán suy luận lôgic, chơi cờ, …

- <u>Nhận xét</u>: Có thể xem A^{KT} là tổng quát hoá của A^T khi xét $h^0 \equiv 0$. Để tìm được đường đi tối ưu, ta cần chọn hàm ước lượng $h^0(x) \le h^*(x)$ $\forall x \in S$, với $h^*(x)$ là hàm giá tối ưu thật sự từ *x* đến DICH.

b. <u>Thuật giải A*</u> giải *bài toán* ***tối ưu*** *BT3** với tri thức bổ sung* ***heuristic*** *trên* ***đồ thị*** *tổng quát*

Tập MO, DONG chứa các phần tử E có dạng (x, cha(x), $g^0(x)$, $f^0(x)$), với qui ước: E_x =x, E_{cha} = cha(x), $E_{g0} = g^0(x)$, $E_{f0} = f^0(x)$. ChènTăng(E, MO) là thao tác chèn E vào MO theo thứ tự tăng của $f^0(x) = g^0(x) + h^0(x)$ từ đầu đến cuối danh sách MO.

A*(x_o, DICH)

```
{1 g°(x_o) = 0; Tính f°(x_o) = h°(x_o); MO ← {(x_o, rỗng, 0, f°(x_o))};
  DONG ← ∅;
  while (MO ≠ ∅) do
  {2    E ← get(MO);                    // trong MO, E ở đỉnh và có f°(E_x) nhỏ nhất !
        Push(DONG, E);
        if (E_x ∈ DICH) then { XuấtLờiGiải(E, DONG); Dừng;}
        for each con ∈ B(E_x) do
        {3    g°(con) = g°(E_x) + c(E_x, con); f°(con) = g°(con) + h°(con);
              if (con ∉ (MO U DONG)_x)
              then ChènTăng((con, E_x, g°(con), f°(con)), MO);
              else {4 if (∃ y ∈ MO and con == y_x and g°(con) < g°(y_x))
                        { Loại(y, MO);              // loại y khỏi tập MO
                          ChènTăng((con, E_x, g°(con), f°(con)), MO);
          }
```

```
                        if (∃ y ∈ DONG  and  con == y_x and g°(con) < g°(y_x))
                          { Loại(y, DONG);      // loại y khỏi tập DONG
                            Push((con, E_x, g°(con), f°(con)), DONG);
                            Cập nhật lại g° và f° cho các hậu duệ của con đã lưu
                          }
                      }4
                }3
        }2
        write("Không thành công");
}1
```

c. <u>Các ví dụ</u>

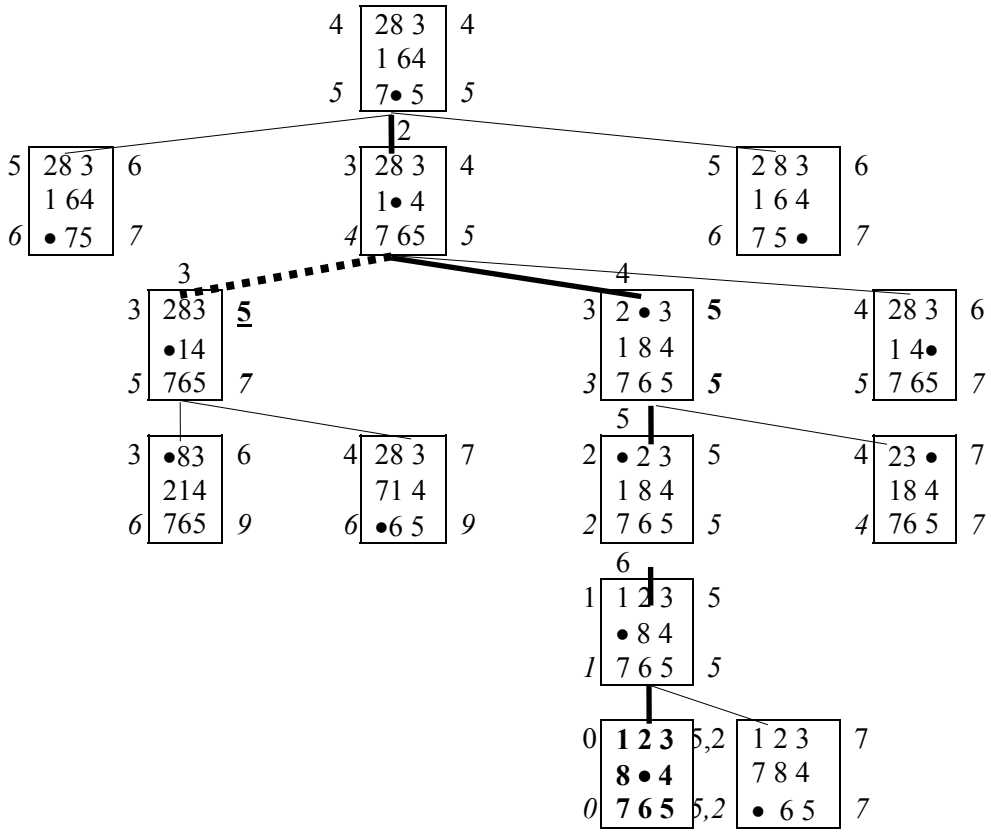
- *<u>Ví dụ 23</u>*: Xét bài toán trò chơi 8 số như trong ví dụ 1 (*hình II.3.1*).

Ta xác định hàm $g^o(x)$ là độ dài đường đi hiện tại từ x_o đến x (số phép dịch chuyển vị trí trống để đưa trạng thái x_o về trạng thái x), còn đối với $h^o(x)$, ta có thể tính ít nhất theo hai cách sau đây.

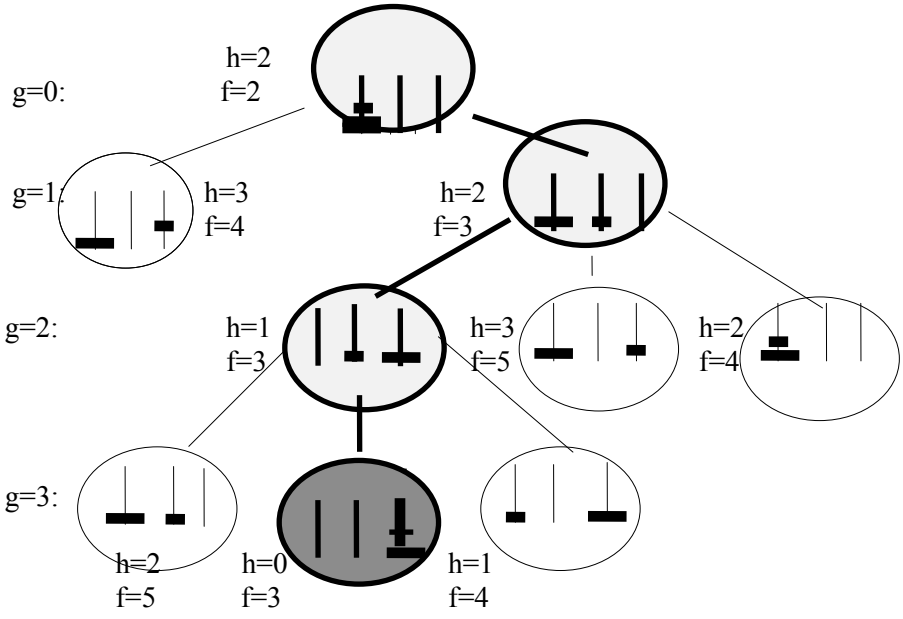
. *Cách 1*: $h^o_1(x)$ là số lượng chữ số (khác trống) trong trạng thái x không nằm ở đúng vị trí của chúng trong trạng thái đích. Chẳng hạn, với đỉnh gốc x_o, $g^o(x_o) = 0$, $h^o_1(x_o) = 4$, do đó $f^o_1(x_o) = g^o(x_o) + h^o_1(x_o) = 4$. Trên *hình II.3.1,* ta có kết qủa áp dụng thủ tục *A** với tri thức bổ sung $h^o_1(n)$ được cho ở góc trên bên trái mỗi trạng thái, giá trị của f^o_1 đối với mỗi nút được cho ở góc trên bên phải của nó. Các số nằm ở đỉnh trên mỗi trạng thái là thứ tự xét chúng trong thủ tục. Với cách chọn hàm ước lượng heuristic không tốt như h^o_1, ta có thể đi nhầm một bước (bước 3).

. *Cách 2*: $h^o_2(x)$ là tổng số các phép di chuyển ít nhất tương ứng với mỗi chữ số (khác trống) trong trạng thái x cần di chuyển đến đúng vị trí của nó trong trạng thái đích. Chẳng hạn, với đỉnh gốc x_o, $g^o(x_o) = 0$, $h^o_2(x_o) = 5$, do đó $f^o_2(x_o) = g^o(x_o) + h^o_2(x_o) = 5$. Trên *hình II.3.1,* ta có kết qủa áp dụng thủ tục *A** với tri thức bổ sung $h^o_2(n)$ được cho ở góc dưới bên trái mỗi trạng thái, giá trị của f^o_2 đối với mỗi nút được cho ở góc dưới bên phải của nó. Với cách chọn hàm ước lượng heuristic h^o_2 tốt hơn h^o_1, ta đi không nhầm bước nào, do h^o_2 *phản ánh sự khác nhau giữa hai trạng thái tốt hơn.* Ta nói *heuristic* h^o_2 *chứa nhiều thông tin hơn heuristic* h^o_1 *nếu:* $h_2^o(n) \geq h_1^o(n)$.

Qua đó, ta thấy *việc sử dụng tri thức bổ sung thông qua hàm* h^o *đóng vai trò rất quan trọng. Nó cho phép gạt bỏ những thông tin không có triển vọng, nhưng nếu* h^o *vượt quá giới hạn h thật sự* thì *có thể thuật toán nhanh dừng song không cho kết quả mong muốn.* Hàm h^o và g^o có ảnh hưởng khá lớn đến sức mạnh heuristics của thuật giải *A**.



(Hình II.3.1)



(Hình II.3.2)

- *Ví dụ 24*: Xét bài toán tháp Hà nội với n = 2 (*hình II.3.2*). Hàm $f^o(n) = g^o(n) + h^o(n)$, với $g^o(n)$ là số đĩa đã di chuyển, $h^o(n)$ là thông tin về mối liên hệ giữa n và trạng thái đích, chẳng hạn:

- Nếu ở cọc C có 2 đĩa, đĩa nhỏ trên đĩa to thì $h^o = 0$.
- Nếu ở cọc C có 1 đĩa to thì $h^o = 1$.
- Nếu ở cọc C chưa có đĩa nào thì $h^o = 2$.
- Nếu ở cọc C có 1 đĩa nhỏ thì $h^o = 3$.

Lời giải tối ưu cho bài toán tháp Hà Nội được trình bày qua cây lời giải in đậm trong *hình II.3.2.*

Một cách *tổng quát*, ta có thể lấy *$f^o(n) = g^o(n) + h^o(n)$, với $h^o(n)$ là tri thức bổ sung chỉ ra triển vọng của đỉnh nằm trên đường tối ưu. Do đó có thể xem $f^o(n)$ là ước lượng của hàm $f(n) = g(n) + h(n)$ với $g(n)$ là giá đường đi tối ưu từ n_o tới n, $h(n)$ là giá đường đi tối ưu từ n tới tập DICH. Nói cách khác $f^o(n)$ là xấp xỉ giá đường đi tối ưu từ n_o tới tập DICH và đi qua đỉnh n. Giá trị $g^o(n)$ đối với n có thể tính được dựa trên ý tưởng thuật giải A* (giá đường đi "gần cực tiểu"từ n_o tới n), còn ước lượng $h^o(n)$ dựa trên thông tin heuristics gắn liền với bài toán*.

Những kết qủa sau bảo đảm tính đúng đắn và tính tối ưu của giải thuật A*.

- *(Tính đúng đắn)*: *Nếu đối với mỗi đỉnh $n \in N$ ta có $0 \le h^o(n) \le h(n)$ và tồn tại số $\Delta > 0$ sao cho đối với mọi cung $a \in A$, $c(a) \ge \Delta$ thì thủ tục A* dừng và cho kết qủa là đường đi p từ n_o tới $n^* \in DICH$ sao cho $g(n^*) \to min$.*

- Giả sử các thủ tục A_i* sử dụng hàm đánh giá:

$$f_i^o(n) = g_i^o(n) + h_i^o(n), \quad i = 1, 2$$

Thêm vào đó, h_2^o thỏa mãn điều kiện $h_2^o(m) - h_2^o(n) \le h(m, n)$, trong đó h(m,n) là độ dài đường đi ngắn nhất từ m tới n và với mọi $n \in N : 0 \le h_1^o(n) \le h_2^o(n) \le h(n)$. Khi đó: số nút đưa vào tập DONG của thủ tục A_2* bao giờ cũng nhỏ hơn số nút đối với thủ tục A_1*.

- Ba nhân tố ảnh hưởng đến sức mạnh heuristics là: giá của đường đi nhận được, số đỉnh phải tháo trong quá trình xử lý, chi phí tính toán giá trị hàm h^o.

- <u>*Nhận xét*</u>:

. Trong số các thủ tục tìm kiếm sử dụng tri thức bổ sung heuristic *h^o, thủ tục sử dụng triệt để các thông tin về các đỉnh ($h^o = h$ và $f^o = g + h$) là tốt nhất và thủ tục A^T ($f^o = g^o$, $h^o = 0$) là tồi nhất*.

. Nếu *chỉ cần tìm một đường đi nào đó tới đỉnh đích* thì có thể không cần tính đến g^o, tức là $f^o = h^o$. Trong một số trường hợp, nếu không sử dụng g^o thì số nút phải tháo có thể tăng đáng kể so với khi sử dụng g^o.

- <u>*Một số heuristics*</u>

. *Lựa chọn theo giai đoạn*: Quá trình tìm kiếm có thể tiến hành theo từng giai đoạn, ở mỗi giai đoạn chỉ giữ lại một tập con nào đó các đỉnh trong tập MO, chẳng hạn các đỉnh với giá trị f^o gần nhỏ nhất.

. *Hạn chế số đỉnh con*: Mỗi lần tháo một đỉnh n, chỉ giữ lại những đỉnh $m \in B(n)$ sao cho $f^o(m)$ thuộc lân cận của f^o nhỏ nhất.

. *Xây dựng dần các đỉnh con*: Mỗi lần tháo đỉnh con chỉ xét tới một vài đỉnh có triển vọng nhất, đầu tiên chỉ chú ý đến đỉnh con quan trọng nhất, còn các đỉnh khác khi nào cần mới đưa ra.

II.3.2. Nguyên lý tham lam

Với thuật giải sử dụng *nguyên lý tham lam* giải *bài toán tối ưu BT3**: ta sử dụng các thuật toán *TìmKiếm* với một ít điều chỉnh: *chỉ chọn lời giải tốt nhất trong một bước kế tiếp* (chứ chưa chắc tốt nhất trong cả quá trình), do đó *không cài đặt cơ chế quay lui. Ưu* điểm của nguyên lý này là trong một số trường hợp, có thể *nhanh thấy lời giải*. Nhưng *nhược* điểm của nó là trong các trường hợp khác, *có thể tồn tại lời giải tối ưu nhưng thuật toán này không tìm ra*. Khi đó, do không dự phòng cơ chế quay lui nên có thể không tìm ra lời giải, mặc dù nó vốn tồn tại ! (*Tham cái lợi trước mắt* mà không để ý đến cái lợi toàn cục và *không dự phòng lưu lại đường đi đã qua để quay lui* khi lạc hướng!)

```
ThamLam(x0, DICH)
{  DONG = ∅;                          // có thể chọn DONG có cấu trúc stack
   MO = {x0};
   if (x0 ∈ DICH) {XuấtLờiGiải(x0, ∅); Dừng;}
   while (MO ≠ ∅)
   {  x = get(MO);                    // lấy một phần tử  x ra khỏi tập MO
      Push(x, DONG);                  // đưa x vào tập DONG
      if (∃phần tử y "tốt nhất" trong B(x))     // luôn tồn tại nếu B(x) ≠ ∅ !
      {  if (y ∈ DICH)
                {XuấtLờiGiải(y, DONG); Dừng;}
         MO = MO U{y};                // đưa y vào tập MO
      }
   }
   write ("Không tìm thấy lời giải");
}
```

Thật ra MO chỉ gồm không quá 1 phần tử ! Vì vậy, có thể thay thuật toán trên bởi:

```
ThamLam2(x0, DICH)
{  DONG = ∅;      // có thể chọn DONG có cấu trúc stack
   if (P(x0))                                  // hay if (x0 ∈ DICH)
             {XuấtLờiGiải(x0, ∅); Dừng;}
   x = x0;
   while (∃phần tử y "tốt nhất" trong B(x))  // luôn tồn tại nếu B(x) ≠ ∅ !
   {  Push (x, DONG);                          // đưa x vào tập DONG
      if (P(y))                                // hay if (y ∈ DICH)
```

```
            {XuấtLờiGiải(y, DONG); Dừng;}
      x = y;
    }
    write ("Không tìm thấy lời giải");
}
```

- <u>*Ví dụ 25*</u>: (*Traveling Saleman)*

Xây dựng một hành trình TOUR có chi phí nhỏ nhất COST cho bài toán đi qua *n* thành phố, với ma trận chi phí C, sao cho bắt đầu từ thành phố *u* và đi qua mỗi thành phố đúng một lần.

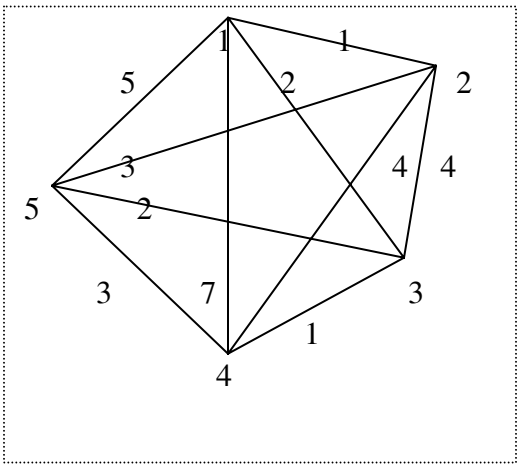
Biến thể của thuật giải *ThamLam2* là *thuật giải GTS* giải bài toán người bán hàng du lịch trên đây.

Thuật giải GTS

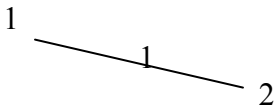
```
{  // Khởi tạo
     TOUR  = ∅; COST = 0;
     v = u;
  // thăm tất cả các thành phố
     for  k = 2 to n do
     {  // Chọn cạnh kế tiếp <v, w> là cạnh có chi phí nhỏ nhất từ v đến
        // các đỉnh chưa  được sử dụng w:
          TOUR = TOUR + <v, w>;
          COST  = COST + C(v, w);
          v = w;                              // khi đó w đã được sử dụng
     }
  // Chuyến đi hoàn thành
          TOUR = TOUR + <v, u>;
          COST = COST + C(v, u);
}
```

Minh họa việc thực hiện:

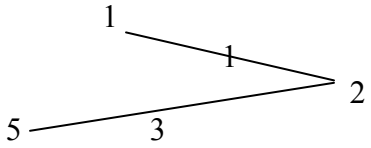
$$C = \begin{bmatrix} \infty & 1 & 2 & 7 & 5 \\ 1 & \infty & 4 & 4 & 3 \\ 2 & 4 & \infty & 1 & 2 \\ 7 & 4 & 1 & \infty & 3 \\ 5 & 3 & 2 & 3 & \infty \end{bmatrix}$$



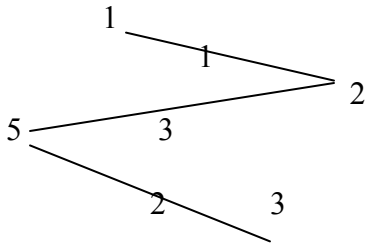
1. TOUR = ∅; COST = 0; u = 1;
   ⇒ w = 2;



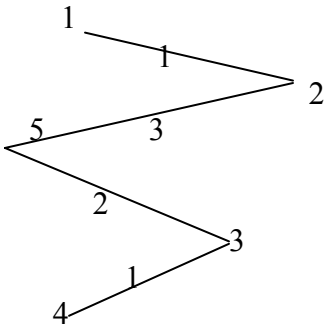
2. TOUR = <1,2>; COST = 1; u = 2;
   ⇒ w = 5;



3. TOUR = {<1,2>, <2,5>}; COST = 4; u = 5;
   ⇒ w = 3;



4. TOUR = {<1,2>, <2,5>, <5,3>}
   COST = 6; u = 3;
   ⇒ w = 4;



5. TOUR = {<1,2>, <2,5>, <5,3>, <3,4>}; COST = 7;
   u = 1;
   TOUR = {<1,2>, <2,5>, <5,3>, <3,4>, <4,1>}; COST = 14;

Tìm min trong các phần tử còn lại của hàng. Xuất phát ở cột nào thì bỏ cột đó

1	*2*	*4*	*5*	*3*
∞	1	2	7	5
1	∞	4	4	3
2	4	∞	1	2
7	4	1	∞	3
5	3	2	3	∞

Do không quay lui nên lời giải vừa tìm chưa chắc là đường đi ngắn nhất. Thật vậy, lời giải vừa tìm có chiều dài lớn hơn đường đi:

TOUR2 = {<1, 3>, <3, 4>, <4, 5>, <5, 2>, <2, 1>} với COST2 = 10.

II.3.3. Nguyên lý hướng đích, phương pháp leo núi (Hill-Climbing)

- Với thuật giải sử dụng *nguyên lý hướng đích* hay *leo đồi (Hill climbing)* giải *bài toán tối ưu BT3*,* ta sử dụng các thuật toán *TìmKiếm* với một ít điều chỉnh, trong đó *không cài đặt cơ chế quay lui ! Ưu* điểm của nguyên lý này là trong một số trường hợp, có thể *tìm thấy rất nhanh lời giải*. Nhưng *nhược* điểm của nó là trong nhiều trường hợp khác, *có thể tồn tại lời giải tối ưu nhưng thuật giải này không tìm ra,* hoặc nó *chỉ tìm thấy lời giải tối ưu địa phương! (Chỉ chú ý đến tối ưu địa phương trước mắt, không dự phòng lưu lại đường đã đi qua* để quay lui khi lạc hướng*, không biết lùi một bước để tiến nhiều bước hơn!)*

```
LeoĐồi(x0, DICH)
{   DONG = ∅;                          // tập DONG có cấu trúc stack
    MO = {x0};
    if (P(x0))                                  // hay if (x0 ∈ DICH)
              {XuấtLờiGiải(x0, ∅); Dừng;}
    while (MO ≠ ∅)
    { x = get(MO);                     // lấy một phần tử  x ra khỏi tập MO
      Push(x, DONG);                   // đưa x vào tập DONG
      if (∃phần tử  tốt nhất y ∈ B(x) mà "tốt hơn" x)
                                       // chưa chắc tồn tại dù B(x) ≠∅ !
      { if (P(y) đúng)                 // hay if (y ∈ DICH)
              {XuấtLờiGiải(y, DONG); Dừng;}
        MO = MO U{y};                           // đưa y vào tập MO
      }
    }
    write ("Không tìm thấy lời giải");
}
```

Thật ra MO chỉ gồm không quá 1 phần tử ! Vì vậy, có thể thay thuật toán trên bởi:

```
LeoĐồi2(x0, DICH)
{   DONG = ∅;                                   // tập DONG có cấu trúc stack
    x = x0; Push(x, DONG);
    if (P(x0))  {XuấtLờiGiải(DONG); Dừng;}
    while (B(x) ≠∅  and ∃phần tử  tốt nhất y ∈ B(x) mà "tốt hơn" x)
                              //chưa chắc tồn tại dù B(x) ≠∅ !
```

```
    { Push(y, DONG);                    // đưa x vào tập DONG
      if (P(y))                         // hay if (y ∈ DICH)
              { XuấtLờiGiải(DONG); Dừng; }
      x = y;
    }
    write ("Không tìm thấy lời giải");
}
```

- Một *biến thể của thuật giải LeoĐồi_2* là thủ tục sau:

Hill_Climbing

{
- Bước 1: n = Đỉnh xuất phát;
- Bước 2: if (n ∈ DICH) then Dừng;
- Bước 3: else *Chọn* $h^0(n_i)$ *bé nhất* trong số mọi đỉnh con $n_i \in B(n)$
  *(*gọi đỉnh này là *Next(n));*
- Bước 4: if $(h^0(n) < h^0(Next(n))$ then Dừng;
  else n = Next(n);
- Bước 5: Goto 2;

}

- <u>Nhận xét</u>: Để *khắc phục nhược điểm* của thuật toán: *nhanh rơi vào lân cận của tối ưu địa phương* mà không thoát ra được, ta có thể kết hợp thêm *thuật giải di truyền GA.*

II.3.4. Nguyên lý sắp thứ tự, nguyên lý trùng khớp nhất

Ta minh họa các nguyên lý trên qua bài toán phân công công việc trên các máy và bài toán xếp hàng vào container.

- <u>*Ví dụ 26*</u>: (Bài toán *phân công công việc*)

Giả sử có n máy: $P_1, \ldots, P_n$ và m công việc: $J_1, \ldots, J_m$. Các công việc được tiến hành đồng thời và có thể được thực hiện trên bất kỳ máy nào. Mỗi lần một công việc được đưa vào máy, máy chỉ dừng khi công việc hoàn tất. Thời gian hoàn tất công việc J_i là t_i. Thời gian nạp công việc vào máy là 0.

Vấn đề đặt ra là: Bố trí công việc vào các máy sao cho tổng thời gian xử lý hết mọi công việc là bé nhất.

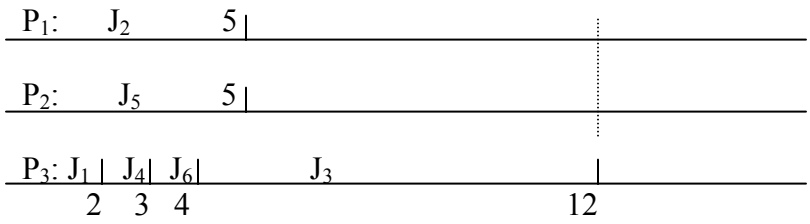
<u>*Mô hình 1*</u>:

. 3 máy: P_1, P_2, P_3.

. 6 công việc

J_i	1	2	3	4	5	6
t_i	2	5	8	1	5	1

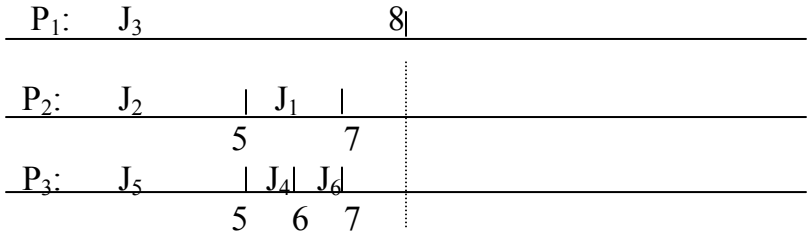
Chọn phương án: $(J_2, J_5, J_1, J_4, J_6, J_3)$; T = 12.



**** Nguyên lý sắp thứ tự:*** *Sắp xếp công việc theo thứ tự thời gian giảm dần.*

Bố trí công việc theo thứ tự trên: $L^* = (J_3, J_2, J_5, J_1, J_4, J_6)$; T = 8.

Đây là lời giải tối ưu.



Mô hình 2(*Phản ví dụ về nguyên lý sắp thứ tự không cho lời giải tối ưu*)

. 2 máy: P_1, P_2.

. 5 công việc

J_i	1	2	3	4	5
t_i	3	3	2	2	2

Phương án:$(J_1, J_2, J_3, J_4, J_5)$ không tối ưu (sắp giảm dần theo thời gian): T=7.

Phương án tối ưu là: $L^* = (J_1, J_3, J_4, J_2, J_5)$; T = 6.

- *Ví dụ 27.1*: Bài toán đóng gói (*xếp hàng vào container)*

Cho dãy các gói hàng:1, 2, 3, 4, 6, 7, 7, 9; kích thước container: 13.

Vấn đề đặt ra: cần xác định số container ít nhất để chứa hết hàng.

- *Các cách (heuristics) xếp* thùng *khác nhau* vào container:

H1: - Thứ tự: tùy ý
- Cách xếp tùy ý.

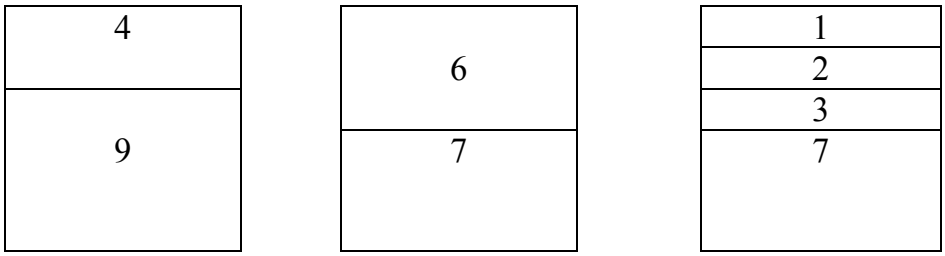
H2: - Thứ tự: Giảm dần theo kích thước
- Cách xếp: Lớn trước, nhỏ sau.

Nguyên lý trùng khớp nhất

H3: - Thứ tự: tùy ý
- Ưu tiên xếp trùng khít.

H4: - *Ưu tiên xếp trùng khít.*
- Sau đó xếp theo *Thứ tự: Giảm dần của kích thước*

Áp dụng nguyên lý này vào ví dụ trên, ta được phương án tối ưu: (9, 7, 7; 4, 6, 3; , , 2; , , 1;) chứa trong 3 container.



- *Ví dụ 27.2*: (*Phương án thu được từ nguyên lý trùng khít nhất không là phương án tối ưu !*)

Cho dãy các gói hàng:2, 2, 5, 6, 8, 9; kích thước container: 16.

. Phương án thu được từ nguyên lý trùng khít nhất: cần 3 thùng

. Phương án tối ưu chỉ cần 2 thùng ! (*Bài tập*)

- *Nhận xét*: Đối với các bài toán phức tạp và đa dạng trong thực tế, nếu *chỉ áp dụng riêng một* phương pháp, ý tưởng hay một nguyên lý heuristic riêng rẽ thì *không thể giải quyết nổi*. Khi đó ta cần áp dụng *chiến lược lai nhiều phương pháp, ý tưởng và nguyên lý giải một cách phù hợp*.

Bài tập

1. *Tìm biểu thức giải tích tường minh từ các hệ thức truy hồi* sau:
   a. $\begin{cases} {}^{+}a_n = a_{n-1} + 2^n, & \forall n \geq 1 \\ a_0 = 1 \end{cases}$
   b. $\begin{cases} {}^{+}a_n = 2a_{n-1} + 1, & \forall n \geq 1 \\ a_0 = 1 \end{cases}$
   c. $*F_n = b_1 F_{n-1} + b_2 F_{n-2}, \quad \forall n \geq 2$ với $\{F_0, F_1\}$, b_1, b_2 cho trước (*trường hợp tổng quát*)
   d. $^{+}$Dãy số *Fibonacci*:
   $\begin{cases} F_0 = 0, \ F_1 = 1 \\ F_n = F_{n-1} + F_{n-2}, & \forall n \geq 2 \end{cases}$
   e. $\begin{cases} *u_n - 4u_{n-1} + 4u_{n-2} = 2n, & \forall n \geq 2 \\ u_0 = 0, \ u_1 = 1 \end{cases}$
   f. $\begin{cases} *T(n) = 2T(n/2) + n - 1, & \forall n \geq 2 \\ T(1) = 1 \end{cases}$
   g. $\begin{cases} u_n = u_{n-1} - u_n u_{n-1}, & \forall n \geq 1 \\ u_0 = 1 \end{cases}$
   h. $\begin{cases} **c_0 = 0, \ c_1 = 0 \\ c_n = n + 1 + \frac{2}{n}\sum_{k=1}^{n-1} c_k, & \forall n \geq 2 \end{cases}$

2. Thể hiện phương pháp *"chia để trị"bằng các thuật toán đệ qui* để giải các bài toán sau:
   a. $^{+}$Tìm min và max của một dãy số.
   b. *Tráo đổi hai phần của mảng (không nhất thiết có độ dài bằng nhau) mà không dùng mảng phụ.

Viết *thuật toán* (bằng *nhiều phương pháp khác nhau*, nếu có thể) *và cài đặt chương trình* để giải các bài toán sau (trong đó có các ví dụ đã được giới thiệu trong lý thuyết):

(Nguyên lý qui hoạch động)

3. $^{+}$Ví dụ 2: bài toán người giao hàng.
4. Ví dụ 3 (bài toán *sắp ba lô có giá trị lớn nhất*): Có *n* loại đồ vật có khối lượng $\{a_i\}$ và giá trị $\{c_i\}$ vào một ba lô có khối lượng w (nguyên dương) . Giả sử mỗi loại đồ vật có đủ nhiều để xếp. Hãy tìm cách xếp sao cho đạt giá trị cao nhất.
5. Tìm dãy các hệ số của nhị thức.
6. Bài toán đổi tiền: Có n loại tiền A_1, A_2, …, A_n (nguyên dương). Hãy tìm cách dùng các loại tiền này để có được số tiền L cho trước (nguyên dương và không bé hơn loại tiền nhỏ nhất) sao cho tổng số tờ là ít nhất.

(Phương pháp vét cạn đơn giản hoặc vét cạn có quay lui)

7. [+]Ví dụ 4.
8. Bài toán 8 quân hậu.
9. [+]Bài toán mã đi tuần (chú ý có thể sử dụng *phần tử cầm canh*).
10. *Tìm các tập con của dãy A các số nguyên dương sao cho có tổng bằng một số nguyên dương M cho trước (*sắp vừa khít* các đồ vật vào *ba lô*).

(Nguyên lý mắt lưới)

11. [+]Ví dụ 5: tìm nghiệm gần đúng của phương trình f(x) = 0 trên [a, b], với f(x) liên tục trên [a,b].

(Phương pháp sinh và thử)

12. [+]Ví dụ 6: bài toán tìm các bộ chỉnh hợp lặp.
13. Tìm các bộ tổ hợp, chỉnh hợp (không lặp).

(Phương pháp nhánh cận)

14. [+]Ví dụ 7: bài toán người du lịch.

(Phương pháp Monte-Carlo)

15. [+]Ví dụ 8: *tìm số* π từ công thức tính diện tích hình tròn hoặc thể tích hình cầu.
16. [+]Ví dụ 9: Kiểm tra *giả thuyết Fermat* bằng thực nghiệm.
17. [+]*Tính gần đúng diện tích hình phẳng* giới hạn giữa trục hoành và đồ thị hàm số liên tục không âm trên đoạn [a, b].

(Thuật giải di truyền - GA)

18. *Ví dụ 10: giải phương trình $x^2 = 25$.

(Nguyên lý mê cung)

19. [+]Ví dụ 11: bài toán mê cung.

20. [+]Dựa trên thuật toán *TìmKiếm* nhằm tìm lời giải đầu tiên cho bài toán *BT3*, ta có thể cải biên để thu được thuật toán vét cạn dưới dạng lặp *VétCạn để tìm mọi lời giải của bài toán sau (bài tập):*

> - <u>Bài toán 4</u>: *Problem_4(G = (S, A), x_0, DICH)* *(BT4)*
> *Cho đồ thị G = (S, A), với đỉnh xuất phát $x_0 \in S$, tập đích DICH $\subset S$.*
> *Hãy tìm <u>mọi</u> đường đi từ x_0 đến một đỉnh nào đó thuộc tập DICH.*

(Các phương pháp tìm kiếm trong không gian trạng thái theo chiều rộng, chiều sâu, sâu dần, cực tiểu)

21. [+]Ví dụ 12: bài toán *Toci* hay *$n^2 - 1$ số.*

(Các phương pháp tìm kiếm trên đồ thị VÀ/HOẶC theo: chiều rộng, chiều sâu, sâu dần, cực tiểu)

22. $^{+}$Ví dụ 13: bài toán tháp Hà Nội với n = 2, 3.

(Phương pháp GPS)

23. Bài toán chứng minh tính hằng đúng của các biểu thức lôgic sau:
    a. $^{+}$ $[a \rightarrow (b \rightarrow c)] \Leftrightarrow [c \vee \neg b \vee \neg a]$
    b. *$[a \wedge (a \vee c \rightarrow b) \wedge (a \wedge b \rightarrow c \wedge d)] \Rightarrow d$

(Phương pháp heuristic)

24. *Tìm thuật giải heuristics* để giải bài toán *tháp Hà Nội* với:
    a. $^{+}n = 3$
    b. *$n \geq 2$ bất kỳ.

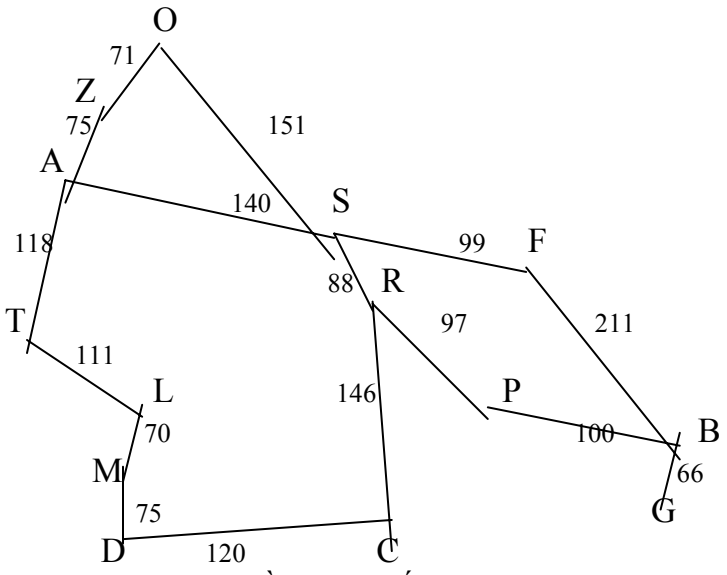
25. $^{+}$Bài toán *Toci* hay *$n^2 - 1$ số*

.

26. $^{+}$*Tìm đường đi ngắn nhất từ* thành phố *A đến* thành phố *B* dựa trên bản đồ các thành phố sau đây, trong đó ngoài các khoảng cách thật sự giữa hai thành phố được chỉ ra bởi các số trên các cung, còn có dãy h^0 chứa chiều dài ước lượng từ một thành phố bất kỳ đến B là khoảng cách đường chim bay từ thành phố đó đến thành phố B.

A	B	C	D	F	G	L	M	O	P	R	S	T	Z
366	0	160	242	178	77	244	241	380	98	193	253	329	374

Dãy h^0 chứa khoảng cách ước lượng từ các đỉnh đến B



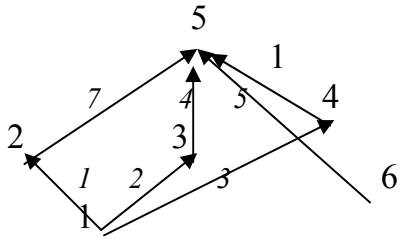
(Bản đồ thành phố)

(Nguyên lý tham lam)

27. +Ví dụ 25: bài toán người du lịch.
28. *Bài toán tô màu (trên đồ thị).

(Nguyên lý leo đồi)

29. Tìm đường đi từ đỉnh 1 đến một trong các đỉnh thuộc tập DICH = {5} trên đồ thị sau với giá thành lớn nhất:



(Nguyên lý sắp thứ tự và nguyên lý trùng khớp nhất)

30. +Ví dụ 26: bài toán phân công công việc.
31. +Ví dụ 27: bài toán sắp xếp vào container.

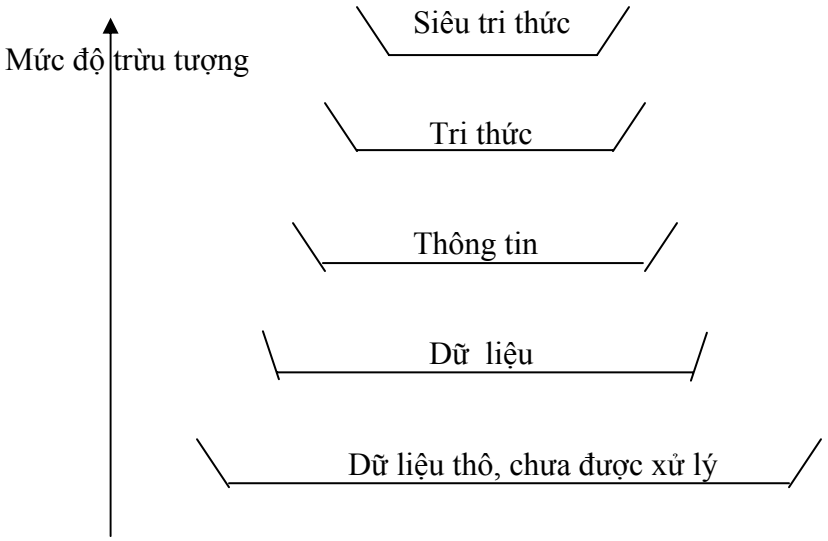
Chương III

BIỂU DIỄN VÀ XỬ LÝ TRI THỨC

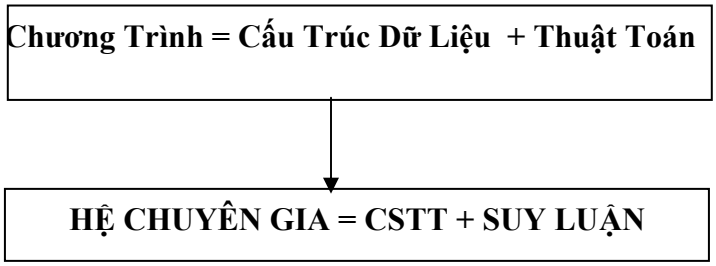
III.1. Khái niệm về biểu diễn và xử lý tri thức

III.1.1. *Từ dữ liệu đến tri thức*

Dữ liệu → Thông tin → Tri thức → Siêu tri thức và Cơ sở tri thức



- *Dữ liệu (Data)*: được biểu diễn dưới dạng chuỗi số, ký tự hay hỗn hợp cả hai loại mà tự bản thân nó không có ý nghĩa độc lập.
- *Thông tin (Information)*: là dữ liệu được tổ chức có đầy đủ ý nghĩa đối với người nhận nó.
- *Tri thức (Knowledge)*: là những thông tin liên quan đến một vấn đề nào đó được tổ chức để có thể giải quyết được vấn đề. *Siêu tri thức (Meta Knowledge)* là loại tri thức ở mức cao, nó có tính khái quát và trừu tượng hơn tri thức.
- *Cơ sở tri thức (CSTT, Knowledge Base)*: là tập hợp các tri thức liên quan đến một vấn đề được sử dụng trong một hệ thống trí tuệ nhân tạo. Nó không chỉ bao gồm các *sự kiện* mà còn chứa các *luật suy diễn* và nó có thể *được sắp xếp lại* khi có thêm một tri thức mới làm thay đổi mối liên hệ giữa chúng. *Hệ chuyên gia (Expert system)* là một hệ cơ sở tri thức (*Knowledge-based system*) được xây dựng từ tri thức của các chuyên gia trong một lĩnh vực nào đó.



III.1.2. ***Một số đặc trưng của tri thức****:* tự giải thích nội dung, có cấu trúc (sự phân cấp giữa các khái niệm và mối quan hệ giữa chúng), có tính liên hệ giữa các tri thức, tính chủ động.

III.1.3. ***Phân loại tri thức***

Có thể phân loại tri thức theo các quan niệm sau:

* *Tri thức tất định và bất định*:

- Tri thức *tiền định (*hay *tất định*) : độc lập với cảm giác, có tính phổ quát và phi mâu thuẫn
- Tri thức *tất yếu (*hay *bất định)*: thu được từ kinh nghiệm mà tính đúng – sai của nó chỉ có thể kiểm tra bằng cảm giác hay kinh nghiệm với độ tin cậy nào đó và có thể có một "mức độ mâu thuẫn" nào đó.

* *Tri thức tồn tại dưới hai dạng cơ bản*:

- *Tri thức định lượng*: thường gắn với các loại heuristics khác nhau, nó phụ thuộc vào chất lượng các hàm đánh giá heuristic và có thể được dùng làm cơ sở để chọn các chiến lược điều khiển.
- *Tri thức định tính gồm 3 dạng chính*:

. *Tri thức mô tả (*hay *khai báo):* cho những thông tin về một sự kiện, hiện tượng hay quá trình mà *không đưa ra thông tin về cấu trúc bên trong cũng như phương pháp sử dụng bên trong tri thức* đó. Nó cho phép *mô tả các mối liên hệ, ràng buộc* giữa các đối tượng.

. *Tri thức thủ tục:* cung cấp các phương pháp *cấu trúc tri thức, ghép nối, suy diễn và tổng hợp các tri thức mới từ các tri thức đã có*. Nó tạo cơ sở cho công nghệ xử lý tri thức: suy diễn, qui diễn, qui nạp, học tri thức.

. *Tri thức điều khiển:* dùng để *điều khiển, phối hợp các nguồn tri thức thủ tục và tri thức mô tả* khác nhau.

III.1.4. ***Các phương pháp biểu diễn tri thức****:* thông qua *lôgic* (lôgic cổ điển hay tất định: lôgic mệnh đề hay vị từ; lôgic bất định: lôgic tình huống, lôgic xác suất, lôgic khả xuất, lôgic mờ), *luật sinh, mạng ngữ nghĩa, khung (frame)*, ...

III.1.5. ***Các phương pháp xử lý tri thức***: *chuyển đổi, suy luận (*chứng minh tự động, các phương pháp giải quyết vấn đề, …), *tổng hợp* tri thức (liên kết, sắp xếp, khái quát hoá, …).

Trong bất kỳ một *hệ thống biểu diễn tri thức đều có chứa 3 yếu tố: ngôn ngữ biểu diễn, cơ chế suy dẫn và công cụ tạo lập cơ sở tri thức cho từng lĩnh vực cụ thể.*

III.2. Một số phương pháp biểu diễn tri thức

III.2.1. Biểu diễn tri thức nhờ lôgic

a. <u>Lôgic cổ điển (hay tất định)</u> gồm : lôgic mệnh đề, lôgic vị từ

* <u>*Lôgic mệnh đề*</u>:

- Xét các mệnh đề p chỉ có thể *nhận một trong hai giá trị lôgic*: đúng (True, 1) hoặc sai (False, 0).
- *Các phép toán lôgic* nối kết các mệnh đề: phép hội (and, $\wedge$, và), phép tuyển (or, $\vee$, hoặc), phủ định (not, $\neg$, không), kéo theo ($\Rightarrow$, $\rightarrow$), tương đương ($\Leftrightarrow$, $\leftrightarrow$, $\equiv$), …

- Vài *phép biến đổi tương đương*:

. $(a \leftrightarrow b) \Leftrightarrow ((a \rightarrow b) \wedge (b \rightarrow a))$

. $(a \rightarrow b) \Leftrightarrow (\neg a \vee b) \Leftrightarrow (\neg b \Rightarrow \neg a)$

$\neg(\underset{1\le i\le n}{\Lambda} a_i) = \underset{1\le i\le n}{V}(\neg a_i); \neg(\underset{1\le i\le n}{V} a_i) = \underset{1\le i\le n}{\Lambda}(\neg a_i) : (DeMorgan)$

. $a \wedge (b \vee c) \Leftrightarrow (a \wedge b) \vee (a \wedge c)$; $a \vee (b \wedge c) \Leftrightarrow (a \vee b) \wedge (a \vee c)$: *phân phối*

. Các phép toán $\wedge$, $\vee$ có tính *giao hoán , kết hợp, lũy đẳng.*

- Các phép toán lôgic được thực hiện theo *thứ tự ưu tiên giảm dần* sau: phủ định, hội, tuyển, kéo theo, tương đương.

- Mọi biểu thức logic mệnh đề đều có thể *đưa về dạng biểu thức tương đương chỉ chứa các phép $\wedge$, $\vee$* và *$\neg$ dưới dạng:*

. *chuẩn hội:*

$$\underset{1\le i\le n}{\Lambda}\left(\underset{1\le ki\le m_i}{V} a_{ki}\right)$$

. *chuẩn tuyển:*

$$\underset{1\le i\le n}{V}\left(\underset{1\le ki\le m_i}{\Lambda} a_{ki}\right)$$

trong đó: $a_{ki} = p_{ki}$ hoặc $a_{ki} = \neg p_{ki}$, với p_{ki} là mệnh đề đơn.

- *Giá trị chân lý* của một biểu thức có thể được tính dựa trên bảng chân lý. Nếu biểu thức có *n* biến mệnh đề khác nhau thì để xác định giá trị chân lý của nó ta phải xét đến 2^n bộ giá trị của các biến mệnh đề (hạn chế là sẽ bị bùng nổ tổ hợp).

- Để *chứng minh một biểu thức lôgic mệnh đề là đồng nhất đúng*, ta có thể dùng: bảng chân lý, các phép biến đổi tương đương, số lôgic, phương pháp GPS, các thuật toán: Robinson, Wong Havard (Vương Hạo), suy diễn tiến, suy diễn lùi mà ta sẽ xét trong phần sau.

- Một *luật suy diễn ở* ***dạng chuẩn Horn*** được biểu diễn như sau:

$$p_1 \wedge .... \wedge p_n \quad \rightarrow \quad q_1 \vee .... \vee q_m \qquad (1)$$

Ta thường gặp các trường hợp đặc biệt sau:

. n = 0, m = 1: biểu diễn các *sự kiện (facts*)

$\square \rightarrow q = F(t_1, ..., t_k)$;

. n ≥ 1, m = 1: ta có ***dạng chuẩn*** *sơ cấp dùng để biểu diễn các luật (rules):*

$$p_1 \wedge .... \wedge p_n \quad \rightarrow \quad q \qquad (2)$$

Mọi luật (1) đều có thể biểu diễn dưới dạng (2):

$(1) \quad \Leftrightarrow \quad (p_1 \wedge .... \wedge p_n \wedge \neg q_2 \wedge .... \wedge \neg q_m \rightarrow q_1\})$

- Quá trình suy diễn có thể dựa trên 2 nguyên lý:

$$\text{Modus Ponens}: \frac{A, A \rightarrow B}{B}$$

$$\text{Modus Tollens}: \frac{\neg B, A \rightarrow B}{\neg A}$$

- *Cơ sở tri thức biểu diễn bằng logic mệnh đề*: dựa trên tập các sự kiện *Facts* và tập các luật suy diễn *Rules* (tương tự cách xây dựng một chương trình trong ngôn ngữ lập trình PROLOG).

* <u>*Lôgic vị từ*</u>:

- *Vị từ p(*$x_1, x_2, ... , x_n$*)* là một phát biểu chứa các biến mệnh đề $x_1, x_2, ... , x_n$ sao cho khi chúng nhận các giá trị lôgic cụ thể thì nó cũng nhận một trong hai giá trị lôgic đúng hoặc sai.

- Trong logic vị từ, *ngoài các phép toán trong logic mệnh đề*, người ta còn sử dụng *lượng từ tồn tại (*ký hiệu là *∃)* và *lượng từ với mọi (*ký hiệu là *∀)*.

- <u>*Ví dụ 1*</u>: Vị từ p(x, y, z) biểu diễn đẳng thức: x.y = z, cho ta một vị từ của 3 biến thực x, y, z. Tính chất giao hoán của phép nhân được diễn tả:

$$\forall x, y \;\; p(x,y,z) \Rightarrow p(y,x,z)$$

- Logic vị từ cho phép *diễn đạt hầu hết các khái niệm và nguyên lý khoa học* cơ bản, nhất là toán học và vật lý. Có 2 phạm vi ứng dụng cơ bản của lôgic hình thức là: giải quyết vấn đề và chứng minh định lý tự động.

- Ngoài việc dùng lôgic vị từ thông thường (lôgic vị từ bậc 1), người ta còn nghiên cứu thêm lôgic vị từ bậc cao. Lôgic vị từ bậc n là lôgic vị từ mà các biến vị từ của nó thuộc lôgic vị từ bậc n-1.

* *Ưu điểm* của phương pháp lôgic:

- Là *ngôn ngữ biểu diễn kiểu mô tả*;
- Có *khả năng suy diễn với các cơ chế* quen thuộc: *modus ponens, modus tollens*;
- Khá *trực quan* đối với người sử dụng;
- Khá *gần gũi về mặt cú pháp* với các câu lệnh của *ngôn ngữ lập trình logic* như PROLOG;
- Có thể *dùng để mô tả cấu trúc mô hình và xử lý động mô hình*;
- Có thể *kiểm tra tính mâu thuẫn* trong cơ sở tri thức;
- Tính *môđun hoá cao*, do đó có thể thêm, bớt, sửa các tri thức khá độc lập với nhau và có cả cơ chế suy diễn.

* *Nhược điểm* của phương pháp lôgic:

- *Mức độ hình thức hoá quá cao*, dẫn tới *khó hiểu ngữ nghĩa* của các vị từ khi xem xét chương trình.
- *Năng suất xử lý thấp* (khó khăn cơ bản trong quá trình suy diễn là cơ chế hợp và suy diễn vét cạn).
- Do tri thức được biểu diễn bằng logic vị từ nên các ưu thế sử dụng cấu trúc dữ liệu không được khai thác triệt để.

b. Lôgic bất định gồm: lôgic *tình huống* (các kết xuất của nó có thể rút ra từ các tình huống không gian và thời gian xác định có liên quan với nhau), lôgic *xác suất*, lôgic *khả xuất* và lôgic *mờ*.

III.2.2. Biểu diễn tri thức nhờ luật sinh: thông qua các luật nếu ... thì:

Nếu P_1, P_2, … và P_n thì Q , *hay:*

$$P_1 \wedge P_2 \wedge \ldots \wedge P_n \rightarrow Q$$

Cách biểu diễn này được áp dụng trong:

- Lôgic mệnh đề và vị từ: với $P_1, P_2, \ldots, P_n$ là các biểu thức lôgic và $\rightarrow$ là phép kéo theo
- Ngôn ngữ lập trình: luật sinh là các câu lệnh

  if (P_1 and P_2 and … and P_n) then Q ;
- Ngôn ngữ tự nhiên: luật sinh là phép dịch.
- Hệ chuyên gia: là một hệ cơ sở tri thức, gồm:

  . CSDL các sự kiện F (Facts) : $\{f_1, ..., f_n\}$

  . Tập luật R (Rules) gồm các luật có dạng:

$$f_{i_1} \wedge f_{i_2} \ldots f_{i_k} \rightarrow q$$

. Phương pháp suy diễn tiến và lùi, chẳng hạn:

$$f_{i_1}, f_{i_2}, \ldots, f_{i_k} \in F \Rightarrow q \in F$$

III.2.3. Biểu diễn tri thức nhờ mạng ngữ nghĩa (Semantic Network)

- *Ví dụ 2*: Ta có thể biểu diễn CSTT:

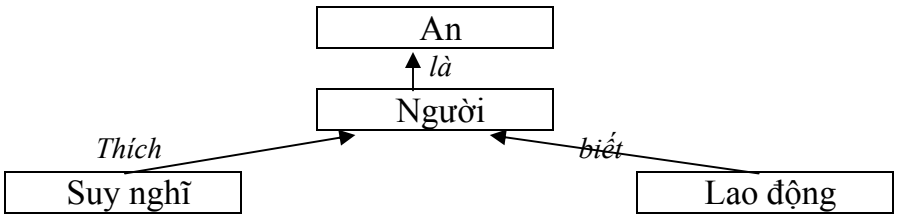
. An là người

. Người thích suy nghĩ

. Người biết lao động

thông qua sơ đồ *III.2.3.1*.

Theo cơ chế lan truyền thông tin theo các cung nối được thể hiện bằng các mũi tên, ta thu được: An thích suy nghĩ và biết lao động.



(*Hình III.2.3.1*)

- *Biểu diễn mạng ngữ nghĩa bằng đồ thị*:

Mạng ngữ nghĩa là đồ thị có hướng mở rộng: G = (V, R), trong đó: $V = \{x_1, x_2, \ldots, x_n\}$ là tập các đỉnh, $R = \{r_1, r_2, \ldots, r_m\}$ là tập các quan hệ nhiều ngôi trên V.

- *Đặc điểm của mạng ngữ nghĩa*: trực quan, có tính kế thừa và lan truyền thông tin; khó kiểm tra tính phi mâu thuẫn, …

- *Ứng dụng của mạng ngữ nghĩa*: trong xử lý ngôn ngữ tự nhiên (dễ dàng hơn trong việc hiểu ngữ nghĩa của câu, do đó có thể tiến hành việc đọc và dịch các bài text), hệ chuyên gia, giải các bài toán thông minh (chẳng hạn, giải bài toán hệ thức lượng trong tam giác), …

III.2.4. Biểu diễn tri thức bằng Frame

- Tương tự như bản ghi (*record*), nhưng *ngoài các trường dữ liệu*, *khung (frame)* còn có thể có các trường là các *thao tác,* thủ tục hay luật suy diễn nào đó, các trường này có thể *kế thừa các tính chất từ các khung khác* (tương tự như khái niệm lớp trong lập trình hướng đối tượng *OOP*). Frame có *tính cấu trúc* rất cao.

- *Ví dụ*:
Frame MáyTínhCáNhân;
{ Là một dạng riêng của MáyTínhĐiệnTử;
  KiếnTrúc {8 bits, 16 bits, 32 bits};
  SốỔĐĩa {1, .., 5};
  MànHình Thủ tục xác định tính năng của màn hình;
}

Ngoài ra, người ta còn có thể biểu diễn tri thức bằng *bộ ba liên hợp OAV* (Object - Attribute – Value), *mạng nơron (ANN* - Artificial Neural Network), …

III.3. Xử lý tri thức tất định bằng phương pháp suy diễn logic

III.3.1. Các cơ chế lập luận với tri thức tất định

- *Suy diễn* (Deduction, diễn dịch): modus ponens, modus tollens, các kiểu suy diễn: tiến và lùi.
- *Qui diễn* (Abduction, tương tự): Cho trước: $A \to B$.
  . $A \sim A' \Rightarrow A' \to B$
  . $B \sim B' \Rightarrow A \to B'$
  . $A \sim A'$ & $B \sim B' \Rightarrow A' \to B'$
- *Qui nạp* (Induction): hoàn toàn và không hoàn toàn

Sau đây ta sẽ xét các thuật toán và thuật giải nhằm chứng minh tự động các biểu thức lôgic mệnh đề.

Bài toán A (BTA):Xét bài toán lôgic mệnh đề sau có hằng đúng hay không:

$$\bigwedge_{1 \le i \le m} GT_i \Rightarrow \bigvee_{1 \le j \le m} KL_j$$

với $\{GT_i , KL_j\}$ là các biểu thức lôgic mệnh đề bất kỳ.

Tại sao ta chỉ xét bài toán dạng trên đây (*bài tập)* ?

III.3.2. Thuật toán Vương Hạo (Wong Havard) giải BTA

* *Ý tưởng*: Áp dụng chiến lược "*Chia để trị*" nhằm tách bài toán xuất phát thành các bài toán con dạng "VÀ" đơn giản hơn. Bài toán ban đầu sẽ được giải khi và chỉ khi mọi bài toán con sơ cấp giải được.

* Trước tiên, ta đưa vế *trái VT* (hay vế *phải VP*) về dạng *chuẩn hội (*hay *chuẩn tuyển* tương ứng) bằng cách:

. Thay các phép toán tương đương $\leftrightarrow$ (nếu có) bởi các phép toán kéo theo $\rightarrow$

. Thay các phép toán $\rightarrow$ bởi các phép toán $\neg, \vee$

. Dùng các luật *De Morgan* để bỏ các dấu $\neg$ của nguyên một nhóm mệnh đề

. Dùng các *luật phân phối* (nếu chưa gặp dạng chuẩn cần tìm) của *phép tuyển đối với phép hội* (hay của *phép hội đối với phép tuyển* tương ứng).

Trong thuật toán Wong, sau khi đưa VT về dạng chuẩn hội và VP về dạng chuẩn tuyển, ta sẽ sử dụng các qui tắc sau (hãy chứng minh tại sao ta có quyền sử dụng chúng ? *Bài tập*).

- Thủ tục *Chuyển*(VT,VP): thay các dấu $\wedge$ các nhóm chính bên VT và các dấu $\vee$ các nhóm chính bên VP bởi dấu phẩy; chuyển vế các mệnh đề chính ở dạng phủ định $\neg$p từ vế này sang vế kia và bỏ đi dấu $\neg$ (chỉ còn p). Nói cách khác, ta có thể xem dấu phẩy (,) bên VT là phép hội và dấu phẩy (,) bên VP là phép tuyển.

. <u>*Ví dụ 3*</u>: Kiểm tra tính hằng đúng của biểu thức lôgic mệnh đề sau:

$VT \equiv (\neg a) \wedge (\neg b \vee c) \quad \Rightarrow \quad (a \wedge b) \vee (\neg b) \vee c \equiv VP$

Các *nhóm chính* trong: vế trái VT là $(\neg a)$ và $(\neg b \vee c)$, vế phải VP là $(a \wedge b)$, $(\neg b)$ và (c).

$VT \equiv (\neg a) \wedge (\neg b \vee c) \quad \Rightarrow \quad (a \wedge b) \vee (\neg b) \vee c \equiv VP$

Chuyển ↓

$\neg a, \neg b \vee c \quad \Rightarrow \quad a \wedge b, \neg b, c$

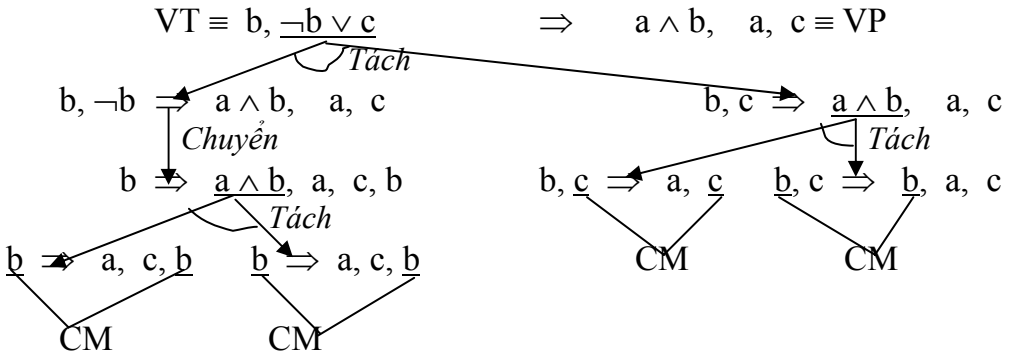
Chuyển ↓

$VT \equiv b, \neg b \vee c \quad \Rightarrow \quad a \wedge b, \quad a, \ c \equiv VP$

- *Hàm lôgic Tách*(VT,VP): tách mỗi *nhóm tuyển* $\vee$ *chính trong VT* (hay *mỗi nhóm hội* $\wedge$ *chính trong VP)* thành *nhiều bài toán con dạng VÀ*. Mỗi bài toán con gồm một mệnh đề trong nhóm chính và giữ nguyên các nhóm chính khác. Nếu tách được thì thủ tục Tách(VT,VP) nhận giá trị *true*; nếu ngược lại, nó nhận trị *false*. Với ví dụ 3 trên đây, ta có sơ đồ biến đổi như *hình III.3.2.1* sau đây.

- Với mỗi bài toán con, nếu mỗi vế của nó *có một mệnh đề đơn đứng độc lập giống nhau thì nó được chứng minh (hằng đúng)*. *Bài toán xuất phát chỉ <u>được chứng minh</u> (hằng đúng) khi <u>mọi</u> bài toán con của nó được chứng minh*. Nói cách khác, *bài toán xuất phát <u>không hằng đúng</u> nếu có <u>tồn tại</u> một bài toán con của nó*

không hằng đúng (không thể áp dụng qui tắc tách cũng như chuyển được nữa và hai vế không có chung một mệnh đề đơn đứng độc lập nào cả).



(Hình III.3.2.1)

Vậy bài toán nêu ra trong ví dụ 3 là hằng đúng.

* ***Thuật toán Vương Hạo***

```
{       VT = VP = ∅ ;
        for  i = 1 to m  do { ChuẩnHội(GTi);      VT ← VT U {GTi }; }
        for  i = 1 to n  do { ChuẩnTuyển(KLi);  VP ← VP U {KLi }; }
        P ← {(VT,VP)};
        while  (P≠∅)  do
        {   (VT,VP) ← get(P);
            if  (VT ∩ VP = ∅) then
            {   Chuyển(VT,VP);
                if  (VT ∩ VP = ∅)  then
                    if  (not(Tách(VT,VP)))
                        then  exit("Không thành công"); // BT không hằng đúng
            }
        }
        write("Thành công");                          // BT hằng đúng
}
```

* *Nhận xét*:

. Thuật toán Wong *dừng sau một số hữu hạn bước* và cho ra thông báo "Thành công" nếu và chỉ nếu từ các giả thiết GT_1, ..., GT_m có thể suy ra một trong các kết luận KL_1, ..., KL_n.

. Nếu số các phép toán liên kết $\vee$ trong GT_i và $\wedge$ trong KL_j là M thì thuật toán sẽ sinh ra từ M đến 2^M dòng (VT,VP) $\in$ P (bùng nổ tổ hợp nếu M khá lớn !).

* *Ví dụ 4 (biểu thức hằng đúng)*: Có thể chứng minh rằng:

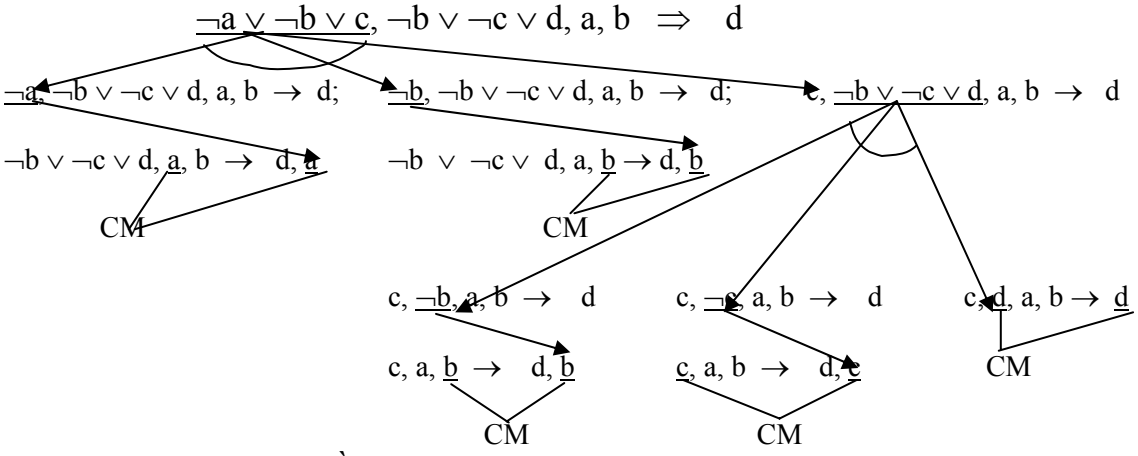
$$[(a \wedge b \rightarrow c) \wedge (b \wedge c \rightarrow d) \wedge a \wedge b] \Rightarrow d \ ?$$

Đưa VT về dạng chuẩn hội và VP về dạng chuẩn tuyển:

VT ≡ {¬a ∨ ¬b ∨ c, ¬b ∨ ¬c ∨ d, a, b}: dạng chuẩn hội

VP ≡ {d}: dạng chuẩn tuyển

Ta có thể *biểu diễn quá trình giải* của thuật toán Wong thông qua *đồ thị suy diễn* hay *đồ thị lời giải* như sau*:*



(*Đồ thị lời giải trong thuật toán Wong)*

* *Ví dụ 5 (biểu thức không hằng đúng)*: Kiểm tra tính hằng đúng của biểu thức:

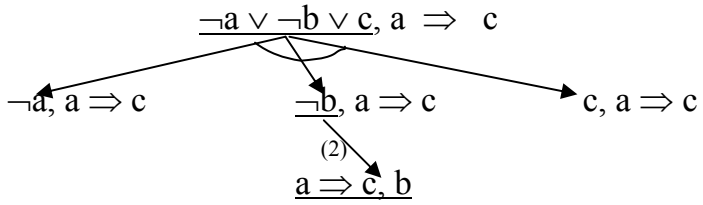
$$[(a \wedge b \rightarrow c) \wedge a] \Rightarrow c \ ? \qquad (1)$$

Đưa VT về dạng chuẩn hội và VP về dạng chuẩn tuyển:

VT ≡ {¬a ∨ ¬b ∨ c, a}: dạng chuẩn hội

VP ≡ {c}: dạng chuẩn tuyển

Đồ thị lời giải:



BT con *(2)* sai vì không thể tách, chuyển và không có mệnh đề chung. Do đó bài toán *(1)* xuất phát cũng sai.

III.3.3. Thuật toán Robinson giải BTA

* *Ý tưởng*: Sử dụng:

. phương pháp *chứng minh phản chứng*:

$[a \rightarrow b$ *đúng*$] \equiv [a \wedge \neg b$ *sai* hay *mâu thuẫn*$]$

. *nguyên lý hợp giải*:

$(\neg a \vee b) \wedge (a \vee c) \Rightarrow b \vee c.$

(V*ì sao? Hãy chứng minh*).

- Ta thường gặp các trường hợp riêng của nguyên lý này:

. nguyên lý *modus ponens*:

$a \wedge (\neg a \vee b) \Rightarrow b$

. nguyên lý *modus tollens:*

$(\neg b \wedge (a \rightarrow b)) \Rightarrow \neg a$

- Để chứng minh từ các giả thiết GT_1, ..., GT_m suy ra một trong các kết luận KL_1, ..., KL_n, ta chỉ cần *lấy phủ định của KL_1, ..., KL_n đưa về cùng với các giả thiết*:

$$\mathbf{P \equiv (\bigwedge_{1 \le i \le m} GT_i) \wedge (\bigwedge_{1 \le j \le m} \neg KL_j)}$$

Nếu *suy ra được mâu thuẫn* từ tập các mệnh đề P thì quá trình chứng minh kết thúc và kết luận *BTA là hằng đúng*.

- Để suy ra được mâu thuẫn, Robinson đã đưa ra nguyên lý hợp giải trên đây để *bổ sung thêm càng ngày càng nhiều các biểu thức mệnh đề trung gian mới* cho đến khi nào trong P *có 2 mệnh đề đơn đứng độc lập là phủ định của nhau thì mâu thuẫn xảy ra*.

* ***Thuật toán Robinson*** *(giải BTA)*

```
{       P = ∅;
        for  i = 1 to m  do { GTi ← ChuẩnHội(GTi);     P  ← P U GTi; }
        for  i = 1 to n do {NotKLi ← ChuẩnHội(¬KLi); P ← P U NotKLi;}
        if  (MâuThuẫn(P)) then exit("Thành công");
        P1 = ∅;
        while  (P≠ P1  and not(MâuThuẫn(P)))  do {P1 =  P; HợpGiải(P);}
        if  (MâuThuẫn(P))
        then exit("Thành công");
        else  exit("Không thành công");
```

}

trong đó, *hàm lôgic MâuThuẫn*(P) và thủ tục *HợpGiải*(P) có nội dung như sau:

Function MâuThuẫn*(P): boolean*

{ **for each** p ∈ P **do**
  for each q ∈ P and q ≠ p **do**
    if (p = ¬ q or q = ¬ p) **then** return(*True*);
  return(*False*);
}

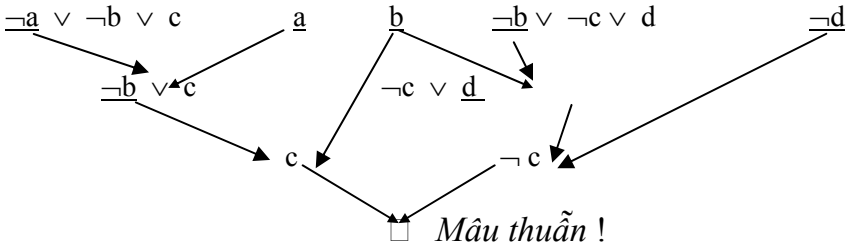
Procedure HợpGiải*(P)*

{ **for each** p ∈ P **do**
  for each q ∈ P and q ≠ p **do**
    if (p = ¬ a ∨ b and q = a ∨ c) **then** P ← P U {b ∨ c};
}

* *Ví dụ 6 (biểu thức hằng đúng)*: Xét bài toán trong *ví dụ 4*, ta có:

P ≡ {¬a ∨ ¬b ∨ c, ¬b ∨ ¬c ∨ d, a, b, ¬d}

Để tiện theo dõi, ta biểu diễn quá trình thực hiện thuật toán qua *đồ thị hợp giải* như sau:



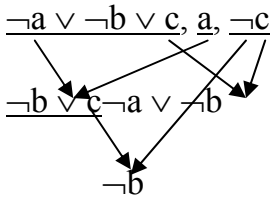
* *Ví dụ 7 (biểu thức không hằng đúng)*: Kiểm tra tính hằng đúng của biểu thức:

[(a ∧ b → c) ∧ a] ⇒ c ?

Lấy phủ định của kết luận và đưa về dạng chuẩn hội, ta có:

P ≡ {¬a ∨ ¬b ∨ c, a, ¬c}

Đồ thị hợp giải:



Không thấy mâu thuẫn cũng không còn cách nào hợp giải mà có thể sinh ra mệnh đề mới nữa. Vậy bài toán xuất phát sai.

* <u>*Nhận xét*</u>:

- Thủ tục Robinson sẽ *dừng sau hữu hạn bước* và cho ra thông báo "Thành công" nếu và chỉ nếu $GT_1 \wedge ... \wedge GT_m \Rightarrow KL_1 \vee ... \vee KL_n$.
- Để *chứng minh BTA không hằng đúng*, ta phải *hợp giải hết tất cả các khả năng cho đến khi không thể sinh ra thêm biểu thức mệnh đề nào mới* trong P.
- Cũng như thủ tục Wong H., thủ tục Robinson có *nhược điểm* là *tùy theo thứ tự lấy các cặp mệnh đề để hợp giải* có thể *xảy ra hiện tượng tràn bộ nhớ (do bùng nổ tổ hợp)* đối với các bài toán có kích thước lớn.
- Có thể *áp dụng một số heuristic* cho thuật toán trên để thu được *"thuật giải Robinson"*, chẳng hạn: khi áp dụng nguyên lý hợp giải, sau khi thêm vào tập P mệnh đề $(b \vee c)$, có thể bỏ hai mệnh đề $(\neg a \vee b)$ và $(a \vee c)$ ra khỏi P.

* <u>*Ví dụ 8*</u> *(biểu thức hằng đúng nhưng có thể kết luận sai nếu áp dụng heuristic không đúng)*: Kiểm tra tính hằng đúng của biểu thức:

$$[a \wedge (a \rightarrow b) \wedge (a \rightarrow c)] \Rightarrow c \qquad (3)$$

Lấy phủ định của kết luận và đưa về dạng chuẩn hội, ta có:

$$P \equiv \{\neg a \vee b, \neg a \vee c, a, \neg c\}$$

. Nếu hợp giải trước hai mệnh đề $\neg a \vee b$ và a để được b rồi loại chúng thì trong P còn lại:

$$P \equiv \{b, \neg a \vee c, \neg c\}$$

Hợp giải tiếp tục thì: $P \equiv \{b, \neg a\}$: không có mâu thuẫn cũng không thể hợp giải được nữa: bài toán sai chăng ? Thật ra, bài toán *(3)* vẫn đúng !

. Nếu hợp giải trước hai mệnh đề $\neg a \vee c$ và a để được c rồi loại chúng thì trong P còn lại:

$$P \equiv \{c, \neg a \vee b, \neg c\}$$

Khi đó mâu thuẫn xảy ra giữa c và $\neg c$. Vậy bài toán xuất phát đúng !

* <u>*Chú ý*</u>: Để tránh tình trạng kết luận sai như trên, khi *thay thuật toán Robinson thành "thuật giải Robinson*", ta nên chú ý rằng ngoài việc *thêm heuristic trên vào thủ tục hợp giải, ta phải bỏ đi thông báo exit("Không thành công")*. Nếu xuất hiện mâu thuẫn thì kết luận "Thành công" và dừng. Nếu *không xuất hiện mâu thuẫn* thì không được thông báo exit("Không thành công"), nghĩa là *không được quyền kết luận gì* khi tình trạng này xảy ra !

* ***<u>Bài toán B (BTB)</u>***

Input: - Tập các mệnh đề giả thiết: $GT = \{g_1, ... , g_n\} = \wedge_{1 \le i \le n} g_i$

- Tập RULE gồm *m* luật có dạng chuẩn r : $p_1 \wedge ... \wedge p_n \rightarrow q$
- Tập các mệnh đề kết luận: KL = $\{q_1, ..., q_k\} = \wedge_{1 \le i \le k} q_i$

Output: Thông báo "Thành công" nếu *mọi* $q_i \in KL$ có thể suy diễn từ GT nhờ sử dụng tập luật RULE.

- Để chứng minh các bài toán lôgic mệnh đề, người ta còn dùng phương pháp *suy diễn tiến* (hay *lùi*), bằng cách *xuất phát từ giả thiết* (hay *kết luận* tương ứng), dựa trên các nguyên lý suy diễn:

. *Modus ponens:* $\dfrac{A,\ A \rightarrow B}{B}$

. *Modus tollens:* $\dfrac{\neg B,\ A \rightarrow B}{\neg A}$

ta *thêm vào các mệnh đề mới được chứng minh đúng* (hay các mệnh đề *cần phải chứng minh thêm* tương ứng) cho đến khi thu được *kết luận* (hay *giả thiết* tương ứng) thì dừng.

III.3.4. Thuật toán suy diễn tiến giải BTB

* *Ý tưởng*: Quá trình *suy diễn tiến bắt đầu từ tập TrungGian các mệnh đề đúng xuất phát* (tập giả thiết*)* và nó *sẽ được "làm nở" dần bằng cách thêm vào các sự kiện mới đúng* nhờ các luật suy diễn trong RULE*, cho đến khi các kết luận cần chứng minh được phát hiện.*

*** *Thuật toán suy diễn tiến SDT***

```
{       TrungGian = GT;
        THỎA   = Lọc(RULE, TrungGian);
      // THỎA là tập các luật r có dạng p1∧ ... ∧pn → q mà pi∈TrungGian, ∀i=1..n
        while (KL ⊄ TrungGian and THỎA ≠ ∅) do
          {     r ← get(THỎA);      // r ∈ THỎA có dạng r: p1∧ ... ∧pn → q
                TrungGian ← TrungGian U {q};
                RULE ← RULE \ {r};
                THỎA = Lọc(RULE, TrungGian)
          }
        if (KL ⊂ TrungGian)
        then write("Thành công")
        else write("Không thành công");
}
```

* *Ví dụ 9*: Cho trước tập các sự kiện giả thiết: GT = {a, b}. Sử dụng tập RULE các luật:

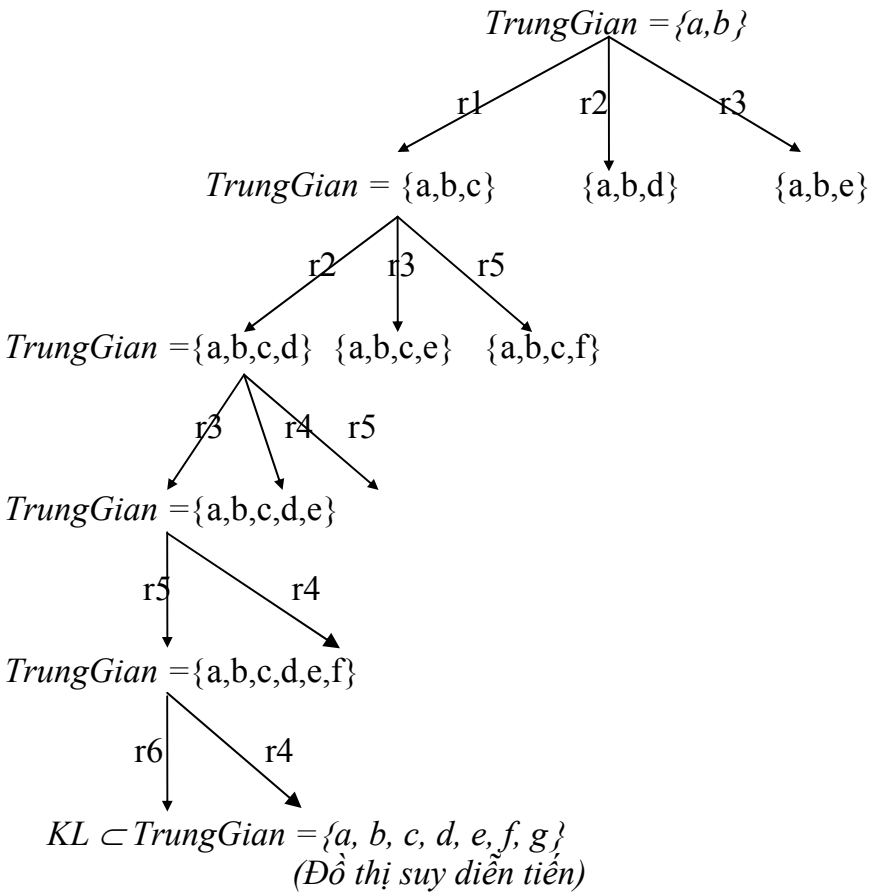
r1: $a \rightarrow c$

r2: $b \rightarrow d$

r3: $a \rightarrow e$
r4: $a \wedge d \rightarrow e$
r5: $b \wedge c \rightarrow f$
r6: $e \wedge f \rightarrow g$
Cần suy ra kết luận: KL = {g}.

. Ta có: TrungGian = {a, b}; RULE = {r1-r6}. Do đó THỎA = {r1,r2,r3}.
. Áp dụng luật r1: $a \rightarrow c$, ta được TrungGian = {a,b,c} và RULE={r2-r6}.
Do đó: THỎA = {r2, r3, r5}.
. Lúc này, KL = {g} $\not\subset$ {a, b, c} = TrungGian và THỎA $\neq \varnothing$.
Ta lại tiếp tục quá trình, bằng cách chọn r2 để thực hiện, ...

Ta minh họa quá trình trên bằng *đồ thị suy diễn tiến*, trong đó mỗi đỉnh ứng với tập TrungGian tại thời điểm đang xét và sẽ có một cung đi từ đỉnh *TrungGian* đến đỉnh *TrungGian* $\boldsymbol{U}$ *{q}* tương ứng với luật r đã được lọc và có dạng r : $p_1 \wedge ... \wedge p_n \rightarrow q$ mà $p_i \in$ TrungGian, $\forall i=1,.., n$.



(Đồ thị suy diễn tiến)

* <u>*Ví dụ 10*</u> *(biểu thức không hằng đúng)*: Kiểm tra tính hằng đúng của biểu thức:

$[(a \wedge b \rightarrow c) \wedge a] \Rightarrow c$?

TrungGian = GT = {a}, KL = {c}, RULE = {r1: $a \wedge b \rightarrow c$ }.

Khi đó: THỎA = $\varnothing$ và KL $\not\subset$ TrungGian. Vậy bài toán trên không hằng đúng.

* <u>*Nhận xét*</u>:

.Với các thứ tự khác nhau để chọn r từ tập THỎA, ta sẽ có các cơ chế duyệt theo chiều sâu, theo chiều rộng, ... trên đồ thị.

. Nếu trong *BTB* thay tập kết luận KL = $\wedge_{1\le i\le k}\, q_i$ bởi: KL = $V_{1\le i\le k}\, q_i$ thì cần hiệu chỉnh thuật toán SDT ra sao ? (*Bài tập*)

. Để tránh các kết luận sai, với *mỗi biến mệnh đề p* xuất hiện trong biểu thức cần chứng minh, bằng các qui tắc biến đổi tương đương lôgic, chuyển về *cùng dạng p* hoặc *cùng dạng* $\neg p$.

III.3.5. Thuật toán suy diễn lùi giải BTB

* Với suy diễn lùi, để đưa ra kết luận B ta thử tìm tất cả các luật có dạng:

$A_1 \wedge ... \wedge A_n \rightarrow B$

Để có B, ta cần chứng minh A_1 ,... , A_n (các kết luận mới được thêm vào tập kết luận). Quá trình xác định A_i cũng diễn ra tương tự như đối với B. Nếu đến một lúc nào đó tìm thấy một A_{io} nào đó không thể dẫn xuất được từ các giả thiết thì ta quay lui sang luật khác sinh ra B và lại tiếp tục quá trình trên. Nếu không tìm được A_{io} như vậy (nghĩa là mọi A_i đều được dẫn xuất từ giả thiết) thì quá trình dẫn xuất ra B thành công.

Để thực hiện quá trình quay lui, ta sử dụng hai tập có cấu trúc ngăn xếp GOAL và VET: GOAL là tập lưu các mệnh đề cần phải chứng minh đến thời điểm đang xét và VET là tập lưu các luật đã sử dụng để chứng minh các đích (kể cả đích trung gian).

* ***<u>Thuật toán suy diễn lùi SDL</u>***

```
{1 if (KL ⊂ GT)
  then exit("Thành công");
  else {2 GOAL = ∅;
        VET = ∅;
        CMĐược = True;
        for each q ∈ KL do
                    GOAL = GOAL U {(q,0)};
        repeat
```

```
{3  (f,i) ← get(GOAL);
    if (f ∉ GT) then
    {4 Tìm_Luật(f, i, RULE, j);          // Tìm luật r_j : left_j →f
      if (j ≤ m)
      then { VET = VET U {(f,j)};
             for each t ∈left_j\GT do GOAL=GOAL U {(t,0)};
           }
      else {5 back = true;               // dùng biến này để quay lui
             while (f ∉ KL and back) do
             {6 repeat {(g,k)←get(VET);
                  // Lấy luật r_k:left_k→g ra khỏi VET
                  // để quay lui đến luật khác mà cũng →g
                        GOAL = GOAL \ left_k;
                        }
               until (f ∈ left_k);
                  // Bỏ hậu quả của việc chọn sai luật r và dẫn đến việc
                  // không CM được mệnh đề f được giả định cần CM
               Tìm_Luật(g, k, RULE, s); // Tìm luật r_s:left_s →g
               if (s ≤ m)
               then { for each t ∈ left_s \ GT do
                                GOAL = GOAL U {(t,0)};
                      VET = VET U {(g,s)}; back = false;
                    }
               else f = g;
             }6
             if (f ∈ KL and back) then CMĐược = False;
           }5
    }4
 }3
until (GOAL = ∅ or not(CMĐược));
if (not(CMĐược)) then exit("Không Thành công")
else exit("Thành công");
}2
}1
```

Trong thủ tục trên ta sử dụng thủ tục: *Tìm_Luật*(f, i, RULE, k) để tìm xem có luật r_k nào kể từ luật thứ i+1 trở đi mà cũng suy ra được f (r_k:$left_k$ →f). Nếu không có luật nào như thế thì qui ước lấy k = m+1.

* <u>*Ví dụ 11*</u>: Cho trước tập các sự kiện giả thiết GT = {a, b}. Sử dụng tập RULE các luật:

r1. $a \wedge b \rightarrow c$
r2. $a \wedge h \rightarrow d$
r3. $b \wedge c \rightarrow e$
r4. $a \wedge d \rightarrow m$
r5. $a \wedge b \rightarrow o$
r6. $o \wedge e \rightarrow m$

Cần suy ra KL = {m}.

Ban đầu GOAL = VET = ∅;

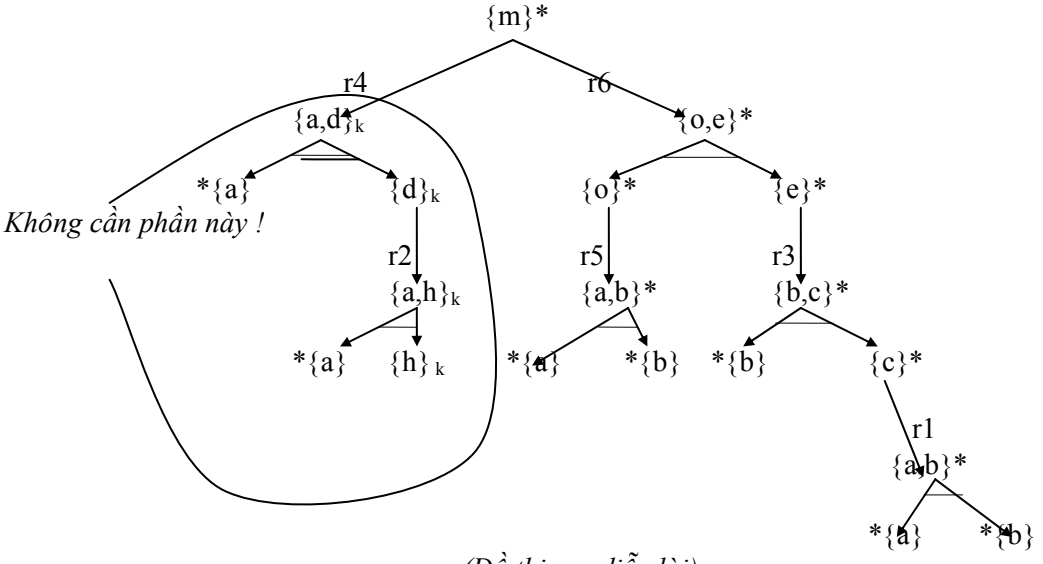
Áp dụng thủ tục *Tìm_Luật*(m, 0, RULE, j), ta được j = 4 (r4 là luật đầu tiên sinh ra m). Khi đó VET = {(m,4)}; GOAL = {(d,0)} (vì a ∈ GT nên chỉ cần xét (d,0)).

Ta tiếp tục quá trình và có bảng theo dõi sau:

GOAL (back)	(f,i)	CMĐ	j	$left_j$\GT	VET	(g,k)	l	$left_l$\GT	Quaylui
{(m,0)}	(m,0)	1	true	d	{(m,4)}				
{(d,0)}	(d,0)		2	h	{(d,2);(m,4)}				
{(h,0)}	(h,0)		7		{(m,4)}	(d,2)	7		true
∅	d				∅	(m,4)	6	o,e	
{(o,0),(e,0)}					{(m,6)}				false(thoát while 6)
{(e,0)}	o		5	∅	{(o,5),(m,6)}				
∅	e		3	c	{(e,3),(o,5),(m,6)}				
{(c,0)}	c		1	∅	{(c,1),(e,3),(o,5),(m,6)}				
∅									

(GOAL = ∅ : thoát vòng while$\{_2\}_2$; *CMĐược = True: Thành công !)*

Ta có thể biểu diễn quá trình suy diễn lùi trên đây thông qua *đồ thị* (VÀ/HOẶC) *suy diễn lùi* như sau:



(Đồ thị suy diễn lùi)

- *Ví dụ 12*: Cho trước tập các sự kiện giả thiết GT = {a}. Sử dụng tập RULE các luật:

r1. $a \wedge b \rightarrow c$
r2. $a \wedge h \rightarrow d$
r3. $b \wedge c \rightarrow e$
r4. $a \wedge d \rightarrow m$
r5. $a \wedge b \rightarrow o$
r6. $o \wedge e \rightarrow m$

Cần suy ra KL = {m}.

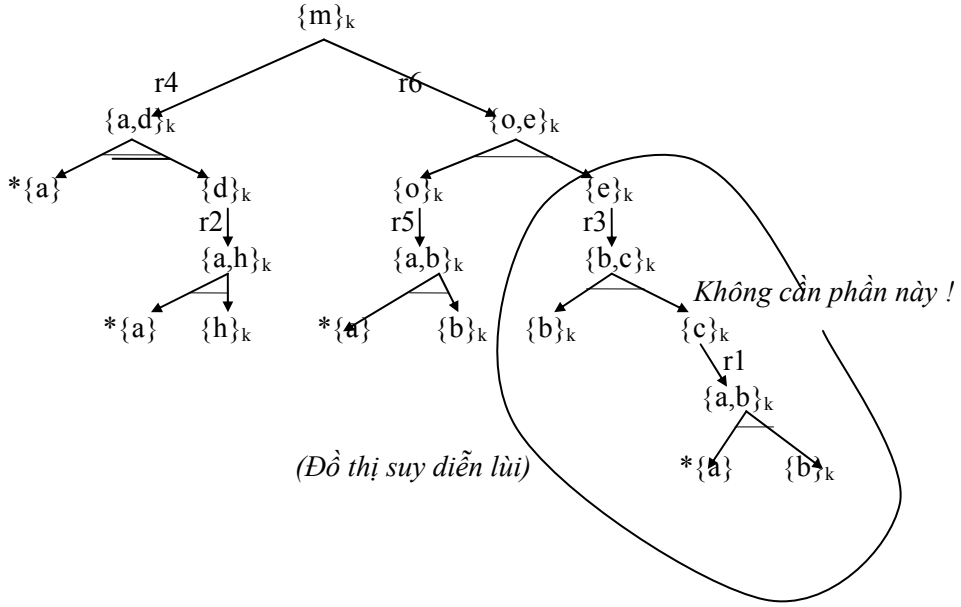
Ban đầu GOAL = VET = ∅. Áp dụng thủ tục *Tìm_Luật*(m, 0, RULE, j), ta được j = 4 (r4 là luật đầu tiên sinh ra m). Khi đó VET = {(m,4)}; GOAL = {(d,0)} (vì a ∈ GT nên chỉ cần xét (d,0)).

Ta tiếp tục quá trình, sẽ có bảng theo dõi sau:

GOAL (back)	(f,i)	CMĐ	j	$left_j$\GT	VET	(g,k)	l	$left_l$\GT	Quaylui
{(m,0)} (m,0)	1	true	d	{(m,4)}					
{(d,0)}	(d,0)		2	h	{(d,2);(m,4)}				
{(h,0)}	(h,0)		7		{(m,4)}	(d,2)	7		true
∅	d				∅	(m,4)	6	{o,e}	
{(o,0),(e,0)}					{(m,6)}			(thoát while 6)	false
{(e,0)}	o		5	b	{(o,5),(m,6)}				
{(b,0),(e,0)}	b		7	∅	{(m,6)}	(o,5)	7		true
∅	o				∅	(m,6)	7		
∅	m	false							

(f = m ∈ KL : thoát while $\{_6\}_6$; do đó: *CMĐược= false (*và GOAL = ∅): thoát vòng while$\{_2\}_2$; *CMĐược= false: Không thành công !)*

Đồ thị suy diễn lùi:



(Đồ thị suy diễn lùi)

* <u>*Nhận xét*</u>:

- Quá trình suy diễn lùi *tương tự như quá trình tìm đồ thị con lời giải trong đồ thị VÀ/HOẶC* biểu diễn tập luật.
- Để *tăng hiệu quả* của thủ tục suy diễn lùi có thể đưa vào 2 tập: tập ĐÚNG chứa các sự kiện đã được khẳng định là đúng và tập SAI chứa các sự kiện đã được khẳng định là sai. Mỗi khi lấy một sự kiện (f, i) nào đó ta cần kiểm tra trước $f \in$ ĐÚNG hay $f \in$ SAI ?

III.4. Xử lý tri thức bất định bằng phương pháp suy diễn logic

III.4.1. Các cơ chế lập luận với tri thức bất định và không chính xác

- *Khái niệm*: Trên thực tế, có nhiều mệnh đề được phát biểu không chính xác, mang tính bất định, *chúng không hẳn hoàn toàn đúng cũng không hoàn toàn sai*. Tính đúng - sai của chúng không được xác định và thể hiện rõ ràng như trong lôgic mệnh đề cổ điển. Ta có thể mở rộng tính đúng sai của mệnh đề *p* thông qua một *hàm μ đo độ đúng* của nó: *μ (p) $\in$ [0; 1]*.

- *Mô hình xác suất*: Xét P là tập các mệnh đề đóng kín đối với các phép toán $\neg$, $\wedge$ (do đó cả $\vee$) và ánh xạ (độ đo xác suất) Pr: $P \to [0, 1]$ thỏa các tính chất:
  - $\Pr(p) + \Pr(\neg p) = 1, \forall p \in P$
  - $\Pr(p \wedge q) = \Pr(p).\Pr(q), \forall p, q \in P$
  - Từ đó: $\Pr(p \vee q) = \Pr(p) + \Pr(q) - \Pr(p \wedge q)$ *(Chứng minh ? Bài tập)*

- <u>*Mô hình khái luật*</u>:

Biểu diễn mệnh đề:

- Nếu p thì q bởi ***luật*** bình thường: $p \to q$
- Nếu p thì *thông thường (nói chung, hầu như, …)* q bởi ***khái luật***: $p \to q$ **!**

III.4.2. Phân bố khả xuất của khái luật và các phép toán nối kết

* <u>*Phân bố khả xuất (PBKX) của khái luật*</u>:

- PBKX của một cặp sự kiện đối ngẫu nhau p và $\overline{p}$ (hay $\neg$p): $\pi(p)$, $\pi(\overline{p})$ thỏa: *max $(\pi(p), \pi(\overline{p})) = 1$*

- *PBKX của một sự kiện p* trong khái luật là bộ: $\begin{pmatrix} \pi(p) \\ \pi(\neg p) \end{pmatrix}$

  Nếu ta không thể khẳng định được p là tuyệt đối đúng thì:

  $\pi(p) = 1$, nhưng $\pi(\overline{p}) = \lambda$ $(\lambda \in [0, 1))$

- *PBKX của một khái luật* $p \to q$! được biểu diễn bởi ma trận:

$$\begin{pmatrix} \pi(p \to q) & \pi(\neg p \to q) \\ \pi(p \to \neg q) & \pi(\neg p \to \neg q) \end{pmatrix}$$

Bài tập

1. +Bằng các phép biến đổi tương đương trong lôgic mệnh đề, hãy chứng minh:
   a. Ta luôn có thể đưa một luật suy diễn ở dạng *chuẩn Horn*:
      $p_1 \wedge .... \wedge p_n \quad \rightarrow \quad q_1 \vee .... \vee q_m$
   về dạng *chuẩn sơ cấp*:
      $p_1 \wedge .... \wedge p_n \quad \rightarrow \quad q$
   b. Nguyên lý *chứng minh phản chứng*:
      (a $\rightarrow$ b *đúng*) $\equiv$ (a $\wedge \neg$ b *sai* hay *mâu thuẫn*)
   c. Nguyên lý *hợp giải.*
   d. Nguyên lý *modus ponens và tollens.*
   e. Trong việc chứng minh tự động tính hằng đúng của các biểu thức lôgic mệnh đề, tại sao người ta thường xét bài toán dạng BTA ?
   f. Quy tắc *Chuyển* và *Tách* trong thủ tục Vương Hạo.
   g. Phương pháp *Qui diễn (Abduction*, tương tự): Cho trước: A $\rightarrow$ B.
      . A ~ A' $\Rightarrow$ A' $\rightarrow$ B
      . B ~ B' $\Rightarrow$ A $\rightarrow$ B'
      . A ~ A' & B ~ B' $\Rightarrow$ A' $\rightarrow$ B'

2. +Hãy đưa về dạng *chuẩn hội* và *chuẩn tuyển* lần lượt cho vế trái (VT) và vế phải (VP) của các biểu thức lôgic mệnh đề sau:
   a. $[(a \wedge b \rightarrow c \vee d) \wedge (c \rightarrow e) \wedge a] \Rightarrow [b \rightarrow e]$
   b. $[(a \wedge b \rightarrow c \wedge d) \wedge (c \rightarrow e) \wedge a] \Rightarrow [b \wedge c \rightarrow e]$
   c. $[(a \vee b \rightarrow c \wedge d) \wedge (c \rightarrow e) \wedge a] \Rightarrow [d \wedge c \vee b]$
   d. $[(a \vee b \rightarrow c \vee d) \wedge (c \wedge b \rightarrow e) \wedge a] \Rightarrow [b \wedge c \rightarrow e]$
   e. $[b \wedge c \wedge \neg e] \Rightarrow [(a \wedge b \wedge (\neg c \vee \neg d)) \vee (c \wedge \neg e) \vee \neg a]$
   f. $[a \rightarrow (b \rightarrow c)] \Leftrightarrow [(a \rightarrow b) \rightarrow c]$
   g. $\{[(a \wedge b \rightarrow c \vee d) \wedge (c \rightarrow e) \wedge a] \vee [(a \wedge b \rightarrow e) \wedge a]\} \Rightarrow [b \rightarrow e]$
   h. $\{[(a \wedge b \rightarrow c \wedge d) \wedge (c \rightarrow e) \wedge a] \vee [(a \wedge b \rightarrow e \vee d) \wedge a]\} \Rightarrow [b \wedge c \rightarrow e]$

3. *Nếu trong bài toán *BTB* thay tập kết luận KL = $\wedge_{1 \le i \le k} q_i$ bởi KL = $V_{1 \le i \le k} q_i$ thì cần hiệu chỉnh thuật toán *suy diễn tiến* SDT ra sao ?

4. +*Kiểm tra tính hằng đúng* của các biểu thức lôgic trong bài tập 2 chương III (có thể *áp dụng vài heuristics để rút gọn hay chuyển đổi tương đương* cho *việc giải trở nên đơn giản hơn*) bằng các phương pháp:
   i) biến đổi *lôgic*
   ii) phương pháp *suy diễn tiến*
   iii) phương pháp *suy diễn lùi*
   iv) thuật toán *Vương Hạo (Wong Havard)*
   v) thuật toán *Robinson.* Nếu *thay thuật toán Robinson bằng "thuật giải Robinson"* thì các kết quả có luôn trùng nhau không ? Tại sao?

Chương IV

LẬP TRÌNH LÔGIC

IV.1. GIỚI THIỆU NGÔN NGỮ LẬP TRÌNH LÔGIC PROLOG

IV.1.1. Mở đầu

Một trong những ***mục tiêu*** lớn và quan trọng của trí tuệ nhân tạo (TTNT) là *mô phỏng trên máy tính những tư duy và hành vi* của con người. Trước những năm 1960, khi chưa có nhóm các ngôn ngữ lập trình (NNLT) chuyên phục vụ cho lĩnh vực TTNT như các NNLT xử lý ký hiệu (LISP- LISt Processing) và lập trình lôgic (PROLOG – PROgramming in LOGic), những thành tựu lý thuyết trong TTNT chưa được ứng dụng rộng rãi trong thực tế. Chương này nhằm giới thiệu *Prolog*, một trong những NNLT thuộc họ này.

NNLT Prolog do Alain Colmerauer và nhóm cộng sự thiết kế ra vào năm 1972 tại đại học Marseille ở Pháp. ***Prolog*** được xây dựng trên ***cở sở*** lý thuyết của *lôgic vị từ*, nhằm *biểu diễn, mô tả* các tri thức cùng các mối quan hệ cơ bản giữa chúng. Từ đó, khi đã *được trang bị sẵn* một cơ chế suy luận thông qua một ***môtơ suy diễn***, Prolog sẽ *sản sinh ra và kiểm tra nhiều mối quan hệ hệ quả khác*.

Khác với nhiều NNLT thủ tục cổ điển truyền thống, ***Prolog*** thuộc nhóm ***NNLT mô tả***. Để giải quyết một bài toán, đặc biệt là các bài toán xử lý ký hiệu trừu tượng và suy luận lôgic, ta chỉ cần *chọn cách biểu diễn phù hợp và mô tả các đối tượng cùng các tính chất và mối quan hệ cơ bản* đã biết giữa chúng. Thông qua môtơ suy diễn của Prolog, ta sẽ *nhận được các câu trả lời* liên quan đến các mối *quan hệ nhân quả và các kết luận liên quan* đến bài toán cần giải quyết. Khi tri thức đã biết về bài toán *không tăng thêm*, việc *trả lời các kết luận khác liên quan đến bài toán cần giải là hiển nhiên thông qua các câu hỏi thêm mà không cần phải lập trình lại* như các NNLT kiểu thủ tục. Chính vì vậy, NNLT Prolog thuộc kiểu ***hội thoại*** và một *chương trình nguồn* trong Prolog ***thường gọn hơn rất nhiều*** so với một chương trình nguồn của các NNLT thủ tục truyền thống khác như Pascal, C, … nhằm cùng giải quyết một bài toán chuyên biệt trong lĩnh vực TTNT.

* ***Ví dụ 1***: Để mô tả các *sự kiện (fact)*: "Socrate và Tèo là người", *qui tắc lôgic (hoặc luật, rule)*: "hễ bất cứ ai là người thì người đó sẽ phải chết", ta dùng hai vị từ chính: LàNgười(Ai), Chết(Ai) như sau:

LàNgười("Socrate").

LàNgười("Tèo").

Chết(Ai) **:-** LàNgười(Ai). //hoặc: Chết(Ai) **if** LàNgười(Ai).

Dựa trên các tri thức vừa mô tả, ta có thể yêu cầu Prolog *trả lời các câu hỏi* liên quan sau:

- *Goal:* LàNgười("Socrate") → yes
- *Goal:* LàNgười("mèo") → no

- *Goal:* Chết("Socrate") → yes
- *Goal:* Chết("mèo") → no

hoặc:

- *Goal:* LàNgười(Ai) → Ai = "Socrate", Ai = "Tèo" (2 solutions)
- *Goal:* Chết(Ai) → Ai = "Socrate", Ai = "Tèo" (2 solutions)

Các ***khái niệm chính*** trong Prolog là: vị từ (predicate), đối tượng, biến, sự kiện (fact), qui tắc (rule), câu hỏi (đích, mục tiêu: Goal), danh sách. Vài ***kỹ thuật*** thường được dùng trong Prolog là: quay lui (mặc định là tìm kiếm theo chiều sâu trên cây suy diễn), đệ qui, lát cắt.

IV.1.2. Vị từ, sự kiện, qui tắc, mục tiêu trong Prolog

a. Vị từ, đối tượng, hằng, biến

* Quan hệ giữa các đối tượng trong một bài toán, một mệnh đề được biểu diễn bằng các ***vị từ*** và các ***đối*** của vị từ biểu diễn các ***đối tượng*** thuộc *quan hệ* hay có *tính chất nào* đó.

Để ***chuyển một mệnh đề thông thường sang dạng vị từ*** (*predicate*) trong Prolog, người ta thường *chọn động từ chính trong phần vị ngữ làm tên vị từ* và các *thành phần khác làm các đối* tương ứng của vị từ.

- ***Phân loại*** các đối tượng trong Prolog: *hằng, biến* hay *đối tượng phức hợp (*là một loại *đối tượng của một vị từ mà bản thân nó lại là một vị từ khác)*.

- Các ***kiểu dữ liệu chuẩn*** (cơ sở) có sẵn trong Prolog: *int, real, char, string, symbol*. Ngoài ra, Prolog còn cho phép các *kiểu dữ liệu riêng do người dùng định nghĩa*.

- ***Kiểu (dữ liệu) của đối tượng*** có thể thuộc kiểu chuẩn hoặc do người dùng định nghĩa.

- Ta qui ước ***Name*** là một *dãy ký tự* (gồm: *chữ cái* la tinh, chữ *số* hoặc *dấu gạch dưới* "_") mà *ký tự đầu tiên phải là chữ cái*. (Do đó, trong *Name* không được có khoảng trắng " " và không được bắt đầu là chữ số !).

Khi dùng *Name* để biểu diễn tên của các đối tượng hay vị từ, người ta thường dùng một trong hai cách viết sau để *dễ đọc và tránh hiểu nhầm nghĩa*: ghế_dài, ghế_đẩu hay ghếDài, ghếĐẩu; suy_luận hay SuyLuận, so_1_Han_Thuyen hay so_1_HanThuyen, … (cho ví dụ vài dãy ký tự không phải là một *Name* ?)

- *Tên của đối tượng* là một *Name*.

- **Hằng** là đối tượng mà giá trị của chúng được gán ngay khi khai báo trong phần **constants** và *không đổi* trong suốt chương trình. ***Tên của hằng*** là một *Name* mà *ký tự đầu tiên phải là* ***chữ cái thường*** (đặc biệt là khi sử dụng giá trị của chúng).

- **Biến** là đối tượng mà giá trị của chúng có thể *thay đổi* trong chương trình. ***Tên của biến*** (*trừ biến vô danh*, được ký hiệu là *dấu gạch dưới* "_") là một *Name* mà ký tự đầu tiên là ***chữ cái hoa***. Biến được phân thành ***3 loại***:

. ***biến ràng buộc*** là biến mà tại thời điểm đang xét nó *được gán với một giá trị xác định*;

. ***biến tự do*** là biến mà tại thời điểm đang xét nó *chưa hoặc không được gán với một giá trị xác định nào cả*; do đó, tại những thời điểm khác nhau một biến có thể *chuyển từ tự do sang ràng buộc và ngược lại* trong quá trình quay lui khi tìm kiếm lời giải;

. ***biến vô danh*** (được ký hiệu là dấu gạch dưới "_") là biến mà ta *không quan tâm* (do đó, không cần lưu lại hay ghi nhớ) *đến giá trị* của nó.

Chú ý: Các *biến* trong Prolog khi được sử dụng *không cần phải khai báo trước* (vì thông thường, chúng đã được khai báo ngầm trong phần khai báo kiểu cho các vị từ)

b. Sự kiện, qui tắc:

Các vị từ trong Prolog thường được thể hiện dưới hai dạng: sự kiện hay qui tắc.

- ***Sự kiện*** (***fact***) là các *mệnh đề hằng đúng* và nó thường được thể hiện bằng *các vị từ mà các đối của chúng đều là đối tượng hằng*. Chẳng hạn, trong ví dụ 1, *LàNgười("Socrate")* là một sự kiện.

- ***Qui tắc*** (***rule***) là một *mệnh đề kéo theo đúng* và chúng thường được thể hiện bởi *các vị từ mà các đối của chúng chứa các biến*. Qui tắc thường ***gồm hai phần***: *PhầnĐầu* và *PhầnThânQuiTắc*, chúng được nối với nhau bởi từ khoá "***if***" hoặc "**:-**", theo ***cú pháp sau***:

PhầnĐầu **if** *PhầnThânQuiTắc***.**

hoặc:

PhầnĐầu **:-** *PhầnThânQuiTắc***.**

(*PhầnThânQuiTắc luôn được kết thúc* bởi một dấu chấm "**.**") Chẳng hạn, trong ví dụ 1, *Chết(Ai)* là một qui tắc.

. Để tạo nên các qui tắc phong phú, ta có thể dùng thêm trong *PhầnThânQuiTắc* các *phép toán lôgic* để nối kết các mệnh đề: **and** (hay dấu phẩy "**,**"), **or** (hay dấu chấm phẩy "**;**"), **not(vị từ)**.

. Các vị từ (sự kiện, qui tắc) phải được *khai báo kiểu* trong phần *Predicates* trước khi được *liệt kê, định nghĩa và được sử dụng* trong phần *Clauses* và *Goal*.

c. Mục tiêu, câu hỏi:

Sau khi lập trình, các yêu cầu của người dùng có thể được biết thông qua các *câu hỏi*, vị từ đưa vào trong phần mục tiêu *Goal*. Có hai loại *Goal*:

- ***Goal trong*** (đưa nội dung cần hỏi vào phần Goal trong chương trình nguồn): *chỉ tìm* ra *lời giải đầu tiên* và *không tự động xuất* chúng ra màn hình. Nếu muốn *xuất chúng*, có thể sử dụng vị từ chuẩn xuất dữ liệu: *write*; nếu muốn *tìm các lời giải kế tiếp*, có thể dùng thêm vị từ chuẩn luôn sai: *fail*.

- ***Goal ngoài***: *tìm* và *tự động xuất mọi lời giải* (ra cửa sổ hội thoại *Dialog*, với từ khóa Goal: ở đầu mỗi câu hỏi). Nếu chỉ muốn *xuất ra lời giải đầu tiên*, có thể sử dụng vị từ chuẩn *lát cắt* "**!**".

IV.1.3. Cấu trúc chính của một chương trình trong Prolog

Một chương trình của Prolog thường bao gồm *một số các phần chính* và *theo thứ tự* sau:

Constants % Đoạn khai báo các đối tượng hằng
Domains % Đoạn khai báo kiểu các đối tượng riêng của người dùng
Database % Đoạn khai báo kiểu cho các vị từ của cơ sở dữ liệu
Predicates % Đoạn khai báo kiểu cho vị từ sẽ dùng trong các đoạn sau
Clauses % Đoạn liệt kê sự kiện và định nghĩa qui tắc đã khai báo trong phần *Predicates*
Goal % Đoạn đưa vào các vị từ câu hỏi, có thể đặt trước phần *Clauses*

Chú ý: Một chương trình có thể thiếu một số (hoặc tất cả !) phần trên.

a. Đoạn khai báo các đối tượng hằng: được bắt đầu bởi từ khoá *Constants* và theo cú pháp sau:

TênĐốiTượngHằng = TrịĐốiTượngHằng
[*Name*]

TrịĐốiTượngHằng có kiểu dữ liệu chuẩn (*int, real, char, string, symbol*).

b. Đoạn khai báo kiểu dữ liệu riêng của người dùng: được bắt đầu bởi từ khoá *Domains* và theo một trong các cú pháp sau:

. Dãy TênKiểuDữLiệu = *KiểuDữLiệu*
[*Name, ...*]

trong đó: *KiểuDữLiệu* là kiểu dữ liệu chuẩn (*int, real, char, string, symbol*) hoặc kiểu dữ liệu riêng của người dùng;

. Dãy TênKiểuDanhSách = KiểuPhầnTửChung*****
[*Name* **,** ...] [*KiểuDữLiệu*]
. TênKiểuĐốiTượngPhứcHợp = Dãy TênVịTừ(KiểuĐối, …)
[*Name*] [*Name(KiểuDữLiệu, ...)***;** …]
. **file** = Dãy TênFileHìnhThức
[*Name***;** ...]

Các tên file của các thiết bị chuẩn sau không cần phải khai báo: *screen, keyboard, printer*.

c. Đoạn khai báo kiểu cho vị từ sẽ dùng trong các đoạn *Goal* và *Clauses* ở phần tiếp theo được bắt đầu bởi từ khoá *Predicates* và theo cú pháp sau:

. TênVịTừ(DãyKiểuĐối)
[*Name*]([*KiểuDữLiệu, ...*])

Đoạn khai báo kiểu cho các vị từ của cơ sở dữ liệu (CSDL) được bắt đầu bởi từ khoá *Database* và cũng theo cú pháp như phần *Predicates*. Điểm khác biệt của các vị từ khai báo trong đoạn *Database* là: các vị từ tương ứng với chúng

trong phần Clauses có các *đối là các đối tượng hằng hoặc các biến ràng buộc* để có thể *đưa vào CSDL ở bộ nhớ trong* theo các vị từ chuẩn riêng biệt.

Chú ý: Có thể khai báo *nhiều vị từ cùng tên* nhưng: *cùng số đối với kiểu khác nhau* hoặc có *số đối khác nhau* hoặc thậm chí *không có đối nào* ! Khi đó, chúng phải được *khai báo liên tiếp nhau*.

d. <u>Đoạn liệt kê các sự kiện và định nghĩa các qui tắc</u> đã khai báo trong phần *Predicates* hoặc *Database*, được bắt đầu bởi từ khoá Clauses theo cú pháp sau:

. Liệt kê các sự kiện:

TênVịTừ(Dãy TrịHằng)**.**

. Định nghĩa các qui tắc:

TênVịTừ(Dãy ĐốiTượng) **:-** *PhầnThânQuiTắc*(Dãy ĐốiTượng)**.**

hoặc:

TênVịTừ(Dãy ĐốiTượng) **if** *PhầnThânQuiTắc*(Dãy ĐốiTượng)**.**

trong đó: *PhầnThânQuiTắc* có thể gồm các vị từ, mệnh đề *được nối kết với nhau bởi các phép toán lôgic* như: and (“,”), or (“;”), not(*VịTừ(Đối*)), nhưng *Đối* của *VịTừ* trong *not* phải là *hằng* hoặc *biến ràng buộc*. Tất cả các vị từ đều phải được kết thúc bởi một dấu chấm “**.**”.

* ***<u>Chú ý</u>***:

- Các *vị từ cùng tên* phải được liệt kê hay định nghĩa *liên tiếp nhau*.
- Cơ chế tìm kiếm mặc định trong Prolog theo chiều sâu. *Thứ tự* của các *mệnh đề cùng tên* (*cùng các đối của chúng*) trong phần *Clauses* có thể *có ý nghĩa khác nhau* và *ảnh hưởng đến tốc độ tìm kiếm*.
- Cần để ý đến *thứ tự và độ ưu tiên* các phép toán lôgic trong *PhầnThânQuiTắc*.
- Qui tắc:

A:- B1; B2.

tương đương với:

A:- B1**.**

A:- B2**.**

- Để biểu diễn mệnh đề: **B and (C or D)**, dùng qui tắc sau là sai:

A:- B, C; D. % hoặc: A:- B, (C;D). cũng sai !

Khi đó, có nhiều cách đúng để biểu diễn chúng:

C1: A:- B, E.
E:- C; D.

hoặc *C2*: A:- B, C.
A:- B, D.

hoặc *C3*: A:- B, C; B, D.

e.* <u>Đoạn mục tiêu</u>** (đối với ***<u>Goal trong</u>), để đưa vào các *vị từ câu hỏi*, kết hợp với các phép toán lôgic, được bắt đầu bởi từ khóa *Goal*.

Nếu nhìn chương trình theo quan niệm Top-Down, đoạn này có thể *đặt trước* phần *Clauses.*

**** Ví dụ 2*** *(minh họa việc dùng Goal trong, Goal ngoài; cơ chế tìm kiếm nghiệm bội của Goal ngoài)*:

Domains

Ai = symbol

Predicates

Thích(Ai, Ai)

Clauses

Thích("A", b).

Thích(b, c).

Thích(X, Z):- Thích(X, Y), Thích(Y, Z).

Goal %trong: *đưa vào trong chương trình nguồn*

%Các đối trong vị từ đều ràng buộc:

Thích("A", b). %Kết quả: rỗng

Thích("A", c), write("Đúng"). %Kết quả: Đúng

%Có đối trong vị từ là biến tự do:

Thích("A", Ai). %Kết quả: rỗng

Thích("A", Ai), **write("A thích: ", Ai, "; ")**.

%Kết quả (xuất một lời giải): A thích: b

Thích(Ai, c), write(Ai, " thích c; "), **fail**.

% Kết quả (xuất nhiều lời giải): b thích c; A thích c

Goal %ngoài: *chỉ đưa vào cửa sổ hội thoại*

%Các đối trong vị từ đều ràng buộc:

> Thích("A", b) → yes

%Có đối trong vị từ là biến tự do:

> Thích("A", Ai) → Ai=b, Ai=c (2 solutions).

> Thích("A", Ai), **!**. → Ai=b (1 solutions).

> Thích(Ai_1, Ai_2) → *kết quả là gì ? (Bài tập)*

**** Ví dụ 3****(minh họa: các phép toán lôgic trong PhầnThânQuiTắc, các qui tắc cùng tên)*:

Domains

Ai = symbol

Predicates

Me(Ai, Ai)

Cha(Ai, Ai)

Cha(Ai)

ChaMe(Ai, Ai, Ai)

ChaHoacMe(Ai, Ai)

Nam(Ai)

Nu(Ai)

Nguoi(Ai)

Dung()
Sai

Clauses

Cha(ba_1, con_1).
Cha(ba_1, con_2).
Cha(Ba):- Cha(Ba, _).

Me(ma_1, con_1).
Me(ma_2, con_2).

ChaMe(Ba, Ma, Con) :- Cha(Ba, Con), Me(Ma, Con).

ChaHoacMe(BaMa, Con) :- Cha(BaMa, Con) **;** Me(BaMa, Con).

Nam(con_1).
Nam(Ai):- Cha(Ai).

Nguoi(Ai):- Cha(Ai) **;** Me(Ai) **;** ChaHoacMe(_, Ai).

Nu(Ai):- Nguoi(**Ai**), **not**(Nam(**Ai**)).
% Nu(Ai):- ***not***(Nam(Ai)). % *không trả lời được câu hỏi*: Nu(Ai) !?

Dung(). %Hoac: Dung():- true.
Sai:- not(Dung()).

Goal %ngoài: Cho kết quả là gì với từng goal sau ?
> Nu(con_2) → ?
> Nu(Ai) → ?

**** <u>Ví dụ 4</u>****(minh họa đối tượng phức hợp; cơ chế tìm kiếm nghiệm bội của Goal ngoài)*:

Domains

Ai, Ten = symbol
Gi = Mèo(Ten) ; Bồ(Ten) ; Rỗng

Predicates

Thich(Ai, Gi)

Clauses

Thich(a, Meo(miu)).
Thich(tu,rong).
Thich(b, Meo(miu)). Thich(b, Bo(miu)).
Thich(c, X) :- Thich(a, X) , Thich(b, X).

Goal %ngoài
> Thich(c, Gi) → Cho kết quả là gì ? (*Bài tập*)

IV.2. DANH SÁCH, ĐỆ QUI, LÁT CẮT TRONG PROLOG

Danh sách là một trong những cấu trúc dữ liệu rất quan trọng và được sử dụng thường xuyên trong *lập trình xử lý ký hiệu* bằng Prolog. *Đệ qui* là kỹ thuật thường dùng trong Prolog. Do *cơ chế tìm kiếm tự động mặc định trong Prolog theo chiều sâu*, nên, trong nhiều tình huống, nếu muốn *ngăn chặn* việc tìm kiếm tiếp tục không cần thiết, Prolog cho phép dùng *lát cắt* để thực hiện việc này.

IV.2.1. Danh sách

a. <u>Định nghĩa</u>: Danh sách là một *dãy có thứ tự* các phần tử. Một cách *đệ qui*, ta có thể định nghĩa danh sách là:

- *rỗng* (không có phần tử nào), hoặc:
- gồm *một phần tử đầu và danh sách đuôi*

b. <u>Khai báo, biểu diễn danh sách</u>:

TênKiểuDanhSách = KiểuChungCácPhầnTử*****

trong đó: *KiểuChungCácPhầnTử* có thể là kiểu dữ liệu chuẩn, kiểu do người dùng định nghĩa, kiểu phức hợp, hoặc thậm chí là danh sách.

- Danh sách *rỗng*: **[]**
- *Biểu diễn danh sách* theo *kiểu đệ qui*:

  BiếnKiểuDS = [**PhầnTử**Đầu | **DanhSách**Đuôi]
- Biểu diễn *danh sách* theo *kiểu liệt kê*:

  [**PhầnTử**ThứNhất **, PhầnTử**ThứHai | **DSách**CònLại]

Về mặt lý thuyết, có thể biểu diễn danh sách bằng *vị từ hoặc cây, biểu đồ ngang*.

**** <u>Ví dụ 1</u>***: Tìm phần tử đầu và danh sách đuôi của các danh sách sau:
[a, b, c], [a], [[1, 2], [3], [4,5]] , [] ? (*Bài tập*)

IV.2.2. Đệ qui - Cơ chế quay lui và tìm kiếm nghiệm bội trong Prolog

a. <u>Cơ chế quay lui và tìm kiếm nghiệm bội</u>: Minh họa qua ví dụ:

**** <u>Ví dụ 2</u>***:

```
%trace QHệ_1_2
QHệ_1(a, b).
QHệ_1(a, c).

QHệ_2(d, b).
QHệ_2(e, c).
QHệ_2(d, c).
```

QHệ_2(e, b).

QHệ_1_2(B, M, C) :- QHệ_1(B, C), QHệ_2(M, C).

Goal % ngoài
> QHệ_1_2(a, e, Ai) → Ai=b, Ai = c

b. <u>Định nghĩa</u>: Một ***thao tác đệ qui F(x) trên tập D*** gồm 2 phần :
. *Điều kiện dừng*: Một thao tác sơ cấp (đã biết cách thực hiện trực tiếp) trên tập con $X_o \subset D$
. Một *lời gọi đệ qui F(H(x))*, với $H(x) \in D$, sao cho sau một số hữu hạn lần gọi đệ qui sẽ phải dẫn đến điều kiện dừng ($H(H(\ldots H()\ldots)) \in X_o$, $\forall x \in D \backslash X_o$).

**** <u>Ví dụ 3</u>****(thao tác đệ qui thiếu điều kiện dừng)*:
Ba(B, C):- Con(C, B).
Con(C, B):- Ba(B, C).
Goal: > Ba(a,b) → ?

c. <u>Kỹ thuật khử đuôi đệ qui bằng cách dùng biến phụ</u>:
**** <u>Ví dụ 4</u>***: Tính *chiều dài* của một danh sách. Do *một vị từ có thể có nhiều tác dụng*, nên chú thích thêm *dòng vào-ra (I, o)* để chỉ các cách thức sử dụng vị từ cho đúng.

Domains
ptu = real
ds = ptu*
i = integer
Predicates
ChieuDai(ds,i) % (i, i) – (i, o) – *(o, i), ! – (o, o), readchar(_)*
ChieuDaiPhu(ds, i, i) % *dòng vào-ra*: *(i, i, i) – (i, i, o)*
% ChieuDaiPhu(i,0,_): luôn khởi tạo biến thứ 2 là 0
Clauses
ChieuDai([], 0).
ChieuDai([_|Duoi], Kq) :- ChieuDai(Duoi,KqPhu), Kq = KqPhu + 1.

ChieuDaiPhu([], Kq, Kq).
ChieuDaiPhu([_|Duoi], Phu, Kq) :- PhuMoi=Phu+1,
ChieuDaiPhu(Duoi, PhuMoi, Kq).
Goal %ngoài
> ChieuDai([1,2,3], 3) → ? > ChieuDai([1,2,3], Kq) → ?
> ChieuDaiPhu([1,2,3], 0, 3) → ?
> ChieuDaiPhu([1,2,3], 0, Kq) → ?
> ChieuDaiPhu(Nao, 0, 1) → ?

Bài tập: Viết 2 qui tắc đệ qui (có và không có đệ qui đuôi) tính giai thừa của một số tự nhiên.

Chú ý: *Thứ tự các vị từ cùng tên* trong phần Clauses có thể *có ý nghĩa khác nhau*, *đặc biệt là các vị từ đệ qui*. Chẳng hạn, xem tác dụng của 2 cách viết qui tắc kiểm tra danh sách có 1 phần tử sau (và sửa lại cho đúng: *bài tập*):
Cách 1:

```
DS1PTu([_|D]) :-  DS1PTu(D).
DS1PTu([]).
```

Cách 2:

```
DS1PTu([]).
DS1PTu([_|D]) :-  DS1PTu(D).
```

với cách hỏi: DS1PTu(Nào) ?

d. <u>Vài thao tác đệ qui cơ bản trên danh sách</u>:

**** <u>Ví dụ 5</u>***: Lập qui tắc kiểm tra *quan hệ thuộc giữa một phần tử và một danh sách*: Thuoc(Ptu, DSach) % (i, i) – (o, i) – (i, o) ?

```
Thuoc(PT, [PT|_]).
Thuoc(PT, [_|DSDuoi]) :- Thuộc(PT, DSDuoi).
```

**** <u>Ví dụ 6</u>***: Lập qui tắc *nối hai danh sách*: Noi(DSach, DSach, DSach) % (i, i, i) – (i, i, o) – (o, o, i) – (i, o, i) – (o, i, i) ?

```
Noi([], L, L).
Noi([PT|L1], L2, [PT|L3]) :- Noi(L1, L2, L3).

ThuocMoi(PT, DS) :- Noi(_, [PT|_], DS).
```

IV.2.3. Lát cắt (!)

Lý do dùng lát cắt: Do cơ chế tìm kiếm tự động mặc định trong Prolog theo *chiều sâu,* nhưng trong nhiều bài toán với các đặc trưng riêng, ta *không cần cơ chế quay lui* hoặc *tìm tiếp lời giải kế tiếp*. Khi đó, ta có thể dùng lát cắt để đạt được mục đích đó.

a. <u>Cú pháp, tác dụng của lát cắt</u>: Dùng dấu chấm than "**!**" để biểu diễn lát cắt. Lát cắt là *vị từ luôn đúng* và có tác dụng ***ngăn***: *cơ chế quay lui và tiếp tục tìm kiếm lời giải đối với các vị từ cùng tên với những vị từ đứng trước lát cắt* trong một mệnh đề nào đó. Điều đó dẫn đến việc *tiết kiệm thời gian lẫn bộ nhớ* khi dùng lát cắt thích hợp.

Ta minh họa lát cắt qua vài trường hợp sau:

- Đối với qui tắc không đệ qui:

VịTừ_1 :- a, !, b. **% (1)**

VịTừ_1 :- c. **% (2)**

Khi a sai thì vị từ c được thực hiện. Khi *a đúng*, sẽ *gặp "!" và thực hiện tiếp b (kể cả nghiệm bội của b, nếu b đúng)*, nhưng *sẽ không bao giờ tìm tiếp lời giải cho a và (kể cả trường hợp b sai) không quay xuống VịTừ_1 thứ hai trong (2) để kiểm tra c* , với các trị trong đối của VịTừ_1 tương ứng với lần gọi đó.

- Đối với qui tắc đệ qui:

VịTừ_2.

VịTừ_2 :- a, !, VịTừ_2. **% (3)**

VịTừ_2 :- b. **% (4)**

Khi a sai, (4) sẽ được kiểm tra. Khi *a đúng*, *VịTừ_2 trong (4) sẽ không được kiểm tra* ứng với lần gọi đó, *mặc dù VịTừ_2 trong thân qui tắc của (3) là sai* và *sẽ không bao giờ tìm tiếp lời giải cho a* .

b. <u>Các tình huống có thể sử dụng lát cắt</u>:

α- *Chỉ tìm lời giải đầu tiên ở Goal ngoài.*

β- Trong các bài toán *chỉ có duy nhất một lời giải*, *sau khi tìm được lời giải đầu tiên*, ta dùng lát cắt để *dừng ngay* quá trình tìm kiếm.

δ- Ta muốn *dừng khi một mệnh đề sai* (nhưng *không cần tìm cách khác để thỏa mãn mệnh đề*), hoặc *thay cho vị từ not*, bằng cách sử dụng *tổ hợp lát cắt* và *vị từ fail* phù hợp.

γ- Dùng lát cắt trong các chương trình "*Sinh và Thử*".

**** <u>Ví dụ 7</u>** (minh họa vài tình huống dùng lát cắt)*:

```
Domains
        i, ptu = integer
Predicates
        RangBuoc(ptu)  % minh họa tình huống δ
        GiaiThua(i, i) % minh họa tình huống β
        ThuongNguyen(i, i, i) % hai đối đầu ∈N, minh họa tình huống γ
        Sinh_1_So(i)
Clauses
    % RangBuoc(X) :- not(free(X)). % sai khi X là biến không ràng buộc
        RangBuoc(X) :- free(X), !, fail.
        RangBuoc(_).

        GiaiThua(0, 1) :- !.
        GiaiThua(N, Kq) :- N1 = N-1, GiaiThua(N1, Kq1), Kq = Kq1*N.

        ThuongNguyen(B, C, T) :- Sinh_1_So(T), T*C<=B, (T+1)*C>B, !.
```

```
Sinh_1_So(0).
Sinh_1_So(Sau) :- Sinh_1_So(Truoc), Sau = Truoc + 1.
```

.

**** Ví dụ 8*** *(Bài tập: tìm các lỗi sai, hạn chế và mở rộng các trường hợp có dòng vào-ra tốt hơn)*: Tính số cha và mẹ của mỗi người.

```
Domains
        Ai = symbol
        i = integer
Predicates
        SoChaMe(Ai, i)
Clauses
        SoChaMe("Adam", 0).
        SoChaMe("Eva", 0).
        SoChaMe(_,2).
% Gợi ý: SoChaMe("Adam", 0): !. hoặc: SoChaMe("Adam", BN): !, BN=0., …
```

**** Chú ý***:

- Phép toán **X = Y** có thể hiểu theo hai nghĩa:

. Phép ***so sánh*** nếu cả X và Y là biến (hay biểu thức) *ràng buộc*;

. Phép ***gán*** nếu chỉ *một trong hai vế là biến tự do*, còn vế kia là biểu thức hay biến ràng buộc (và sẽ vô nghĩa, bị lỗi khi cả hai vế đều không ràng buộc!)

- Có thể thay phép toán = bởi vị từ *Bằng(X,X)*.

**** Gợi ý tạo các lệnh cấu trúc như các NNLT cổ điển***:

- *if (ĐK) then A;*
  else B;

```
Neu(DK) :- !, A.
Neu( _ ) :- B.
```

- *switch (BT)*
  { case a1: A1; break;
  case a2: A2; break;

```
...
```

  case an: An; break;
  default: B; break;
  };

```
%Gọi sử dụng: Switch(BT)
Switch(a1) :- !, A1.
Switch(a2) :- !, A2.
        …
Switch(an) :- !, An.
Switch( _ ) :- B.
```

- *repeat A until (DK);*

```
Repeat :- Lap, A, DK, !.

Lap.
Lap :- Lap.
```

- *while (ĐK) do A;*

```
While :- ĐK, !, A, While.
While.
```

- *for (i=Bdau; i<=Kthuc; i = i+Step) do A(i);*

```
% Gọi sử dụng: For(Bdau, Kthuc, Step)
For(I, N, Step) :- Dấu(Step, D), D*(I-N) > 0, !.
 For(I, N, Step) :- A(i), I_Sau = I+Step, For(I_Sau, N, Step).
```

IV.3. CÁC VÍ DỤ

IV.3.1. Bài toán "Tháp Hà Nội" *(Minh họa vị từ đệ qui)*

Ví dụ 1: Có 3 cọc A, B, C. Trên cọc A có n đĩa tròn chồng lên nhau, với đường kính khác nhau, sao cho đĩa bé luôn nằm trên đĩa lớn; hai cọc B, C trống. Tìm cách chuyển n đĩa từ A đến C với số lần di chuyển đĩa ít nhất (2^n-1), sao cho: mỗi lần chỉ di chuyển 1 đĩa bé nhất trên cùng của một cọc tùy ý sang một cọc khác để đĩa bé luôn nằm trên đĩa lớn.

```
%Program Thap_HaNoi
Domains
        i  = integer
        s = symbol
Predicates
        HaNoi(i, i)
        ChuyenDia(i, i, s, s, s, i)
Clauses
        HaNoi(N, SoLanDiChuyen) :- ChuyenDia(N, N, a, c, b, SoLanDiChuyen).

        ChuyenDia(1, Nhan, Tu, Den, _, 1) :- !, nl,
                write("Chuyen dia: ", Nhan, " tu ", Tu, " den ", Den).
        ChuyenDia(N, Nhan, Tu, Den, Tgian, Tong):-
                N1=N-1, NhanPhu=Nhan-1,
                ChuyenDia(N1, NhanPhu, Tu, TGian, Den, Tong1),
                ChuyenDia(1, Nhan, Tu, Den, Tgian, 1),
                ChuyenDia(N1, NhanPhu, TGian, Den, Tu, Tong2),
```

```
        Tong = Tong1 + Tong2 + 1.
Goal %trong
    makewindow(1, 3, 7, "Thap Ha Noi", 0, 0, 25, 80),
    write("Nhap so dia n (1<=n<=12): "), readint(N), HaNoi(N, Tong),
    write("Tong so lan di chuyen dia: ", Tong)..
```

Kết quả của chương trình với n=2 là:

```
        Chuyen dia: 1 tu a den b
        Chuyen dia: 2 tu a den c
        Chuyen dia: 1 tu b den c
        Tong so lan di chuyen dia: 3
```

IV.3.2. Bài toán xử lý vi phân ký hiệu *(Minh họa bài toán xử lý ký hiệu)*:

Tính đạo hàm của một hàm số theo một biến bất kỳ.

```
%Program Xử lý vi phân ký hiệu
Domains
    KyHieu = symbol
    So     = real
    BieuThuc = bien(KyHieu); hang(So);
               cong(BieuThuc,BieuThuc); tru(BieuThuc,BieuThuc);
               nhan(BieuThuc,BieuThuc); chia(BieuThuc,BieuThuc);
               cos(BieuThuc); sin(BieuThuc);
               tg(BieuThuc); cotg(BieuThuc);
               ln(BieuThuc); log(BieuThuc,BieuThuc);
               exp(BieuThuc); mu(BieuThuc,BieuThuc);
               luyThua(BieuThuc,BieuThuc)
Predicates
    d(BieuThuc,KyHieu,BieuThuc)
Clauses
    d(hang(_),_,hang(0)).
    d(bien(X),X,hang(1)):-!.
    d(bien(_),_,hang(0)).

    d(cong(U,V),X,cong(U1,V1)):- d(U,X,U1), d(V,X,V1).
    d(tru(U,V),X,tru(U1,V1)):- d(U,X,U1), d(V,X,V1).
    d(nhan(U,V),X,cong(nhan(U1,V),nhan(U,V1))):- d(U,X,U1), d(V,X,V1).
    d(chia(U,V),X,chia(tru(nhan(U1,V),nhan(U,V1)),nhan(V,V))):-
        d(U,X,U1), d(V,X,V1).

    d(sin(U),X,nhan(cos(U),U1)) :- d(U,X,U1).
    d(cos(U),X,tru(hang(0),nhan(sin(U),U1))) :- d(U,X,U1).
    d(tg(U),X,chia(U1, mu(cos(U),hang(2)))) :- d(U,X,U1).
```

```
d(cotg(U),X,nhan(hang(-1),chia(U1, mu(sin(U),hang(2))))) :- d(U,X,U1).

d(ln(U),X,chia(U1,U)):-d(U,X,U1).
d(log(U,V),X, Kq) :- d(chia(ln(V),ln(U)),X,Kq).

d(exp(U),X,nhan(exp(U),U1)) :- d(U,X,U1).
d(mu(U,V),X,nhan(mu(U,V),W)):- d(nhan(V,ln(U)),X,W).

d(luyThua(U,hang(A)),X,nhan(hang(A),nhan(U1,luyThua(U,hang(A_1))))):-
        A_1 = A-1, d(U,X,U1).
Goal %ngoài: tính vi phan của x.sin(x)
    d(nhan(bien(x),sin(bien(x))),x, Kq).
```

Kết quả (*chưa rút gọn*) của chương trình là:

```
Kq=cong(nhan(hang(1),sin(bien(x))),nhan(bien(x),
                                        nhan(cos(bien(x)),hang(1))))
```

IV.3.3. Bài toán suy luận lôgic *(Minh họa lập trình cho bài toán lập luận lôgic)*

Ví dụ 2: Trên một đảo nọ, chỉ có hai nhóm người: một nhóm chỉ gồm những người luôn nói thật và nhóm còn lại chỉ gồm những người luôn nói láo. Một nhà thông thái đến đảo này và gặp ba người A, B, C. A nói: "Trong ba người chúng tôi chỉ có đúng một người nói thật", C nói: "Cả ba người chúng tôi đều nói láo". Hỏi trong ba người A, B, C, ai nói thật, ai nói láo ?

Ta *biểu diễn mỗi phương án* của bài toán bằng danh sách [A, B, C], sao cho mỗi phần tử chỉ gồm một trong hai trạng thái 0, 1, tương ứng với trạng thái nói láo hoặc nói thật. Ta *mô tả câu nói của A và C* như sau: nếu A nói thật thì Tổng = A+B+C = 1, nếu A nói láo thì Tổng ≠ 1; nếu C nói thật thì Tổng = A+B+C = 0, nếu C nói láo thì Tổng > 0.

```
%Program Noi lao, noi that
Domains
    i, ptu = integer
    ds  = ptu*
Predicates
    TaoDS(i,ds)
    Quet(ptu)
    Giai(i,ds)
    SuKien(symbol,ds)
    Tong(ds,integer)
    Viet(ds,i)
Clauses
    Quet(0).
```

```
Quet(1).

TaoDS(N,[]) :- N <= 0, !.
TaoDS(N,[Dau|Duoi]) :- Quet(Dau),N1 = N - 1, TaoDS(N1,Duoi).

Giai(N,DS) :- TaoDS(N,DS), SuKien("A",DS), SuKien("C",DS).

SuKien("A",DS) :- DS = [1,_,_], Tong(DS,1).
SuKien("A",DS) :- DS = [0,_,_], Tong(DS,So), So <> 1.

SuKien("C",DS) :- DS = [_,_,1], Tong(DS,0).  % phi li !
SuKien("C",DS) :- DS = [_,_,0], Tong(DS,So), So > 0.

Tong([],0) :- !.
Tong([Dau|Duoi],Kq) :- Tong(Duoi,KqDuoi), Kq = KqDuoi + Dau.

Viet([],_) :- nl,!.
Viet([1|D],I):-!,I1=I+1,I2 =I1+64,char_int(C,I2), write(C," noi that\n"),
              Viet(D,I1).
Viet([0|D],I):-!,I1=I+1,I2 =I1+64,char_int(C,I2), write(C," noi lao\n"),
              Viet(D,I1).
```

Goal *% trong*

```
Giai(3,DS),Viet(DS,0),nl.
```

Kết quả của chương trình là:

A noi that
B noi lao
C noi lao

IV.4. PHỤ LỤC: VÀI VỊ TỪ CHUẨN CỦA PROLOG

IV.4.1. NHẬP VÀ XUẤT DỮ LIỆU

Các vị từ xuất và nhập dữ liệu xét ở đây sẽ được thao tác từ các thiết bị vào ra hiện thời, ngầm định là màn hình và bàn phím.

1. Các vị từ nhập dữ liệu

Các ký hiệu sẽ sử dụng trong biểu diễn cú pháp các vị từ có ý nghĩa như sau: Sau tên vị từ các biến có các kiểu được liệt kê tiếp theo. Sau đó, dòng vào ra đối với các biến của vị từ sẽ được chỉ ra, trong đó các biến có vị trí tương ứng với ký tự:

. 'o' phải là biến tự do (sau khi vị từ được thực hiện các kết qủa của nó sẽ được gán vào các biến này);

. 'i' phải là biến ràng buộc hoặc đối tượng hằng (chúng cung cấp dữ liệu ban đầu để các vị từ này thực hiện).

a) **readln(StringVariable)**, (string) – (o): trong đó StringVariable là biến tự do có kiểu string. Vị từ này có tác dụng đọc một chuỗi từ thiết bị vào hiện thời (và tất nhiên được gán cho biến StringVariable) cho đến khi gặp ký tự xuống dòng (phím Enter, có mã ASCII tương ứng là '\13') được ấn.

b) **readint(IntgVariable)**, (integer) – (o): đọc một số nguyên từ thiết bị vào hiện thời cho đến khi gặp ký tự xuống dòng được ấn.

c) **readreal(RealVariable)**, (real) – (o): đọc một số thực từ thiết bị vào hiện thời cho đến khi gặp ký tự xuống dòng được ấn.

d) **readchar(CharVariable)**, (char) – (o): đọc một ký tự (nhưng không xuất hiện ký tự trên màn hình) từ thiết bị vào hiện thời và không cần kết thúc bằng ký tự xuống dòng.

e) **file_str(DosFileName, StringVariable)**, (string, string) – (i, o) (i, i): chuyển đổi nội dung văn bản giữa file có tên thực là DosFileName và biến chuỗi có tên StringVariable (cho đến khi gặp ký tự kết thúc file eof, tương ứng với tổ hợp phím Ctl-z hay mã ASCII là '\26').

f) **inkey(CharVariable)**, (char) – (o): đọc một ký tự từ bàn phím (gán cho biến CharVariable). Vị từ này chỉ đúng khi có một ký tự được ấn từ bàn phím.

g) **keypressed**: kiểm tra việc một phím được bấm hay chưa nhưng không đọc ký tự này vào. Vị từ này chỉ đúng khi có một ký tự được ấn từ bàn phím.

h) **keypressed**: kiểm tra việc một phím được bấm hay chưa nhưng không đọc ký tự này vào. Vị từ này chỉ đúng khi có một ký tự được ấn từ bàn phím.

2. Các vị từ xuất dữ liệu

a) **write(Variable|Constant *)**: trong đó dấu * chỉ ra rằng các biến Variable (ràng buộc) hay hằng Constant có thể được lặp lại và chúng được viết cách nhau bằng các dấu phẩy ‘,’. Vị từ này có tác dụng xuất đến thiết bị ra một số đối tượng.

b) **nl**: xuống dòng.

c) **writef(FormatString, Variable|Constant *)**: xuất ra một số đối có định khuôn dạng theo cú pháp sau: %-m_p$, trong đó:

. ‘-’: canh lề bên trái, nếu không có dấu ‘-’ sẽ là canh lề bên phải;

. ‘m’: xác định độ rộng vùng nhỏ nhất cho đối;

. ‘p’: xác định độ chính xác các số sau dấu chấm thập phân hay số lớn nhất các ký tự được in ra của chuỗi;

. ‘$’ là một trong các ký hiệu quy ước chuẩn:

d: số thập phân dạng chuẩn.

x: số thập lục phân.

s: chuỗi (symbols hay strings).

c: ký tự (chars hay integers).

g: số thực dạng ngắn nhất ngầm định.

e: số thực dạng mũ.

f: số thực dạng dấu phẩy động.

d) **“\n”**: ký tự xuống dòng.

“\t”: ký tự lùi vào một số khoảng trắng.

“\nnn”: ký tự đưa vào bảng mã ASCII tương ứng.

IV.4.2. CÁC THAO TÁC FILE

a) **openread(SymbolicFileName, DosFileName)**, (file, string) – (i, i): mở một file trên đĩa DosFileName để đọc và gán cho tên file hình thức (trong phần khai báo kiểu file) SymbolicFileName.

b) **openwrite(SymbolicFileName, DosFileName)**, (file, string) – (i, i): mở một file trên đĩa để ghi và gán cho tên file hình thức; nếu file đó đã tồn tại thì nó sẽ được xóa trước khi gọi.

c) **openappend(SymbolicFileName, DosFileName)**, (file, string) – (i, i):mở một file trên đĩa để nối thêm dữ liệu và gán cho tên file hình thức.

d) **openmodify(SymbolicFileName, DosFileName)**, (file, string) – (i, i):mở một file trên đĩa để sửa đổi (đọc và ghi) và gán cho tên file hình thức. Khi dùng vị từ này, nên kết hợp với vị từ filepos để cập nhật file tại vị trí tùy chọn.

e) **readdevice(SymbolicFileName)**, (file) – (i) (o): đặt lại hay hiển thị thiết bị đọc vào file đang mở (để đọc hay sửa đổi), ngầm định là bàn phím.

f) **writedevice(SymbolicFileName)**, (file) – (i) (o): đặt lại hay hiển thị thiết bị ghi vào file đang mở (để đọc ghi hoặc nối thêm hay sửa đổi), ngầm định là màn hình.

g) **closefile(SymbolicFileName)**, (file) – (i): đóng file (dù nó đã mở hay chưa).

h) **filepos(SymbolicFileName, FilePosition, Mode)**, (file, real, integer) – (i, i, i) (i, o, i): đưa con trỏ đến hay trả lại giá trị của vị trí FilePosition trong file tính từ chế độ Mode: 0: đầu file, 1: vị trí hiện tại trong file, 2: cuối file.

i) **eof(SymbolicFileName)**, (file) – (i): kiểm tra con trỏ có ở cuối file hay không. Vị từ này đúng nếu con trỏ ở vị trí cuối file.

j) **existfile(DosFileName)**, (string) – (i): vị từ này thành công nếu file có tên chỉ ra tồn tại trong thư mục hiện thời.

k) **deletefile(DosFileName)**, (string) – (i): xóa file có tên chỉ ra.

l) **renamefile(OldDosFileName, NewDosFileName)**, (string, string) – (i, i): đổi tên file cũ thành tên file mới.

m) **disk(DosPath)**, (string) – (i) (o): thiết lập hay trả lại ổ đĩa và đường tìm kiếm thư mục mặc định.

IV.4.3. CÁC THAO TÁC ĐỐI VỚI MÀN HÌNH VÀ CỬA SỔ

1. ***Màn hình***

Màu nền	**Giá trị**	**Màu chữ**	**Giá trị**
Black	0	Black	0
Blue	16	Blue	1
Green	32	Green	2
Cyan	48	Cyan	3
Red	64	Red	4
Magenta	80	Magenta	5
Brown	96	Brown	6

White	112	White	7
		Grey	8
		Light Blue	9
		Light Green	10
		Light Cyan	11
		Light Red	12
		Light Magenta	13
		Yellow	14
		White (High Intensity)	15

a) **attribute(Attr)**, (integer) – (i) (o): cho (đặt hay trả lại) giá trị thuộc tính của màn hình. Giá trị đặc trưng cho thuộc tính của màn hình, dựa vào bảng trên đây, được tính như sau:
   1. Chọn một màu chữ và một màu nền;
   2. Cộng những giá trị nguyên tương ứng với màu trong bảng 1;
   3. Cộng thêm giá trị đó với 128 nếu muốn các chữ hiển thị nhấp nháy.

b) **scr_char(Row, Column, Char)**, (integer, integer, char) – (i, i, i) (i, i, o): cho ký tự Char trên màn hình tại vị trí có tọa độ (Row, Column).

c) **scr_attr(Row, Column, Char)**, (integer, integer, char) – (i, i, i) (i, i, o): cho thuộc tính của màn hình tại vị trí có tọa độ (Row, Column).

d) **filed_str(Row, Column, Length, String)**, (integer, integer, integer, string) – (i, i, i, i) (i, i, i, o): cho chuỗi String trên màn hình tại một vùng bắt đầu ở vị trí có tọa độ (Row, Column) và có độ dài là Length ký tự.

e) **filed_attr(Row, Column, Length, Attr)**, (integer, integer, integer, integer) – (i, i, i, i) (i, i, i, o): cho thuộc tính của màn hình tại một vùng bắt đầu ở vị trí có tọa độ (Row, Column) và có độ dài là Length ký tự.

f) **cursor(Row, Column)**, (integer, integer) – (i, i) (i, o): dịch con trỏ đến vị trí có tọa độ (Row, Column) hay trả lại tọa độ của con trỏ hiện thời.

2. Hệ thống cửa sổ

a) **makewindow(WindowNo, ScrAtt, FrameAtt, FrameStr, Row, Column, Height, Width)**, (integer, integer, integer, string, integer, integer, integer, integer) – (i, i, i, i, i, i, i, i) (o, o, o, o, o, o, o, o): xác định một vùng màn hình làm cửa sổ.

b) **makewindow(WindowNo, ScrAtt, FrameAtt, FrameStr, Row, Column, Height, Width, ClearWindow, FrameStrPos, BorderChars)**, (integer, integer, integer, string, integer, integer, integer, integer, integer, integer, string) – (i, i, i, i, i, i, i, i, i, i) (o, o, o, o, o, o, o, o, o, o): xác định một vùng màn hình làm cửa sổ với các thuộc tính sau:

ClearWindow	= 0	không xóa cửa sổ sau khi tạo,
	= 1	xóa cửa sổ sau khi tạo;
FrameStrPos	= 255	canh tít ở giữa,
	<> 255	đặt tít tại vị trí FrameStrPos;
BorderChars	một chuỗi gồm 6 ký tự để vẽ khung:	
	ký tự thứ 1: góc trên bên trái,	
	ký tự thứ 2: góc trên bên phải,	
	ký tự thứ 3: góc dưới bên trái,	
	ký tự thứ 4: góc dưới bên phải,	
	ký tự thứ 5: đường kẻ ngang,	
	ký tự thứ 6: đường kẻ dọc.	

c) **shiftwindow(WindowNo),** (integer) – (i) (o): đổi cửa sổ làm việc đến cửa sổ thứ WindowNo (vẫn giữ nguyên trạng thái của cửa sổ trước đó) hay trả lại số của cửa sổ đang làm việc.

d) **gotowindow(WindowNo),** (integer) – (i): di chuyển nhanh đến cửa sổ thứ WindowNo mà không lưu nội dung của cửa sổ cũ vào bộ đệm cửa sổ.

e) **resizewindow**: điều chỉnh lại kích thước cửa sổ như kiểu thực đơn.

f) **resizewindow(StartRow, NoOfRows, StartCol, NoOfCols)**, (integer, integer, integer, integer) – (i, i, i, i): điều chỉnh lại kích thước cửa sổ một cách trực tiếp.

g) **colorsetup(Main_Frame)**, (integer) – (i): thay đổi màu cửa sổ hay khung tùy theo giá trị của:

Main_Frame	= 0	đổi màu cửa sổ
	= 1	đổi màu khung

h) **existwindow(WindowNo)**, (integer) – (i): vị từ này thành công nếu tồn tại cửa sổ số WindowNo.

i) **removewindow**: loại bỏ cửa sổ hiện thời.

j) **removewindow(WindowNo, Refresh)**, (integer, integer) – (i, i): loại bỏ cửa sổ số WindowNo với chế độ xóa nền hay không xóa nền (nếu Refresh bằng 1 hoặc 0).

k) **clearwindow**: xóa cửa sổ hiện thời về màu nền của nó.

l) **window_str(ScreenString)**, (string) – (i) (o): chuyển đổi nội dung giữa cửa sổ hiện thời và chuỗi ScreenString.

m) **window_at(Attribute**), (integer) – (i): xác định thuộc tính cho cửa sổ hiện thời.

n) **scroll(NoOfRows, NoOfCols**), (integer, integer) – (i, i): cuộn nội dung của cửa sổ lên (hay xuống) X=NoOfRows dòng tùy theo nó > 0 (hay < 0) và sang trái (hay phải) X=NoOfCols cột tùy theo nó > 0 (hay < 0).

o) **framewindow(FrameAttr**), (integer) – (i): thay đổi thuộc tính của khung cửa sổ.

p) **framewindow(FrameAttr, FrameStr, FrameStrPos, FrameTypeStr**), (integer, string, integer, integer) – (i, i, i, i): thay đổi thuộc tính của khung cửa sổ theo các chế độ sau:

FrameAttr	= thuộc tính khung
FrameStr	= tít của khung
FrameStrPos	= nhận giá trị từ 0 đến độ rộng cửa sổ. Nó đặt vị trí cho tít trên khung. Tít sẽ đặt ở giữa nếu nó bằng 255
FrameTypeStr	= một chuỗi gồm 6 ký tự để chỉ dạng của khung.

IV.4.4. CÁC VỊ TỪ CHUẨN KHÁC

1. Thao tác chuỗi

a) **frontchar(String, FrontChar, RestString)**, (string, char, string) – (i, o, o) (i, i, o) (i, o, i) (i, i, i) (o, i, i): tách một chuỗi thành ký tự đầu và chuỗi còn lại.

b) **fronttoken(String, Token, RestString)**, (string, string, string) – (i, o, o) (i, i, o) (i, o, i) (i, i, i) (o, i, i): tách một chuỗi thành một 'từ' và chuỗi còn lại.

c) **frontstr(Length, InpString, StartString, RestString)**, (integer, string, string, string) – (i, i, o, o): tách từ một chuỗi cho trước InpString ra một chuỗi con StartString bắt đầu từ ký tự thứ nhất có độ dài là Length và chuỗi còn lại RestString.

d) **concat(String1, String2, String3)**, (string, string, string) – (i, i, o) (i, o, i) (o, i, i) (i, i, i): nối String1 và String2 thành String3: String3 = String1 + String2.

e) **strlen(String, Length)**, (string, integer) – (i, i) (i, o) (o, i): chiều dài của String là Length. Với dòng vào ra (o, i) ta được một chuỗi gồm Length ký tự trắng.

f) **isname(StringParam)**, (string) – (i): kiểm tra chuỗi có là một 'name' (một dãy các ký tự chữ hoặc số hoặc dấu '_' nhưng bắt đầu bằng chữ hay ký tự '_' trong Turbo Prolog không.

2. Chuyển đổi kiểu

a) **char_int(CharParam, IntgParam)**, (char, integer) – (i, o) (o, i) (i, i): chuyển đổi giữa một ký tự và mã ASCII tương ứng với nó.

b) **str_int(StringParam, IntgParam)**, (string, integer) – (i, o) (o, i) (i, i): chuyển đổi giữa một chuỗi và một số nguyên tương ứng với nó.

c) **str_char(StringParam, CharParam)**, (string, char) – (i, o) (o, i) (i, i): chuyển đổi giữa một chuỗi và một ký tự tương ứng với nó.

d) **str_real(StringParam, RealParam)**, (string, real) – (i, o) (o, i) (i, i): chuyển đổi giữa một chuỗi và một số thực tương ứng với nó.

e) **upper_lower(StringInUpperCase, StringInLowerCase)**, (string, string) – (i, o) (o, i) (i, i): chuyển đổi giữa một chuỗi ký tự hoa và một chuỗi ký tự thường.

f) **upper_lower(CharInUpperCase, CharInLowerCase)**, (char, char) – (i, o) (o, i) (i, i): chuyển đổi giữa một ký tự hoa và một ký tự thường.

3. Thao tác dối với cơ sở dữ liệu trong

a) **consult(DosFileName)**, (string) – (i): gọi ra và bổ sung một file cơ sở dữ liệu trong (file đó được khởi tạo và lưu một cơ sở dữ liệu trong không tên bằng vị từ save(?)).

b) **consult(DosFileName, InternalDatabaseName)**, (string, InternalDatabaseName) – (i, i): gọi ra và bổ sung một file cơ sở dữ liệu trong (file đó được khởi tạo và lưu một cơ sở dữ liệu trong không tên bằng vị từ save(?, ?)).

c) **save(DosFileName)**, (string) – (i): lưu mọi mệnh đề của cơ sở dữ liệu trong không tên.

d) **save(DosFileName, InternalDatabaseName)**, (string, InternalDatabaseName) – (i, i): lưu mọi mệnh đề của cơ sở dữ liệu trong có tên.

e) **assert(Term)**, (InternalDatabaseDomain) – (i) hay **asserta(Term)**, (InternalDatabaseDomain) – (i): chèn một sự kiện vào đuôi một cơ sở dữ liệu trong không tên.

f) **assertz(Term)**, (InternalDatabaseDomain) – (i): chèn một sự kiện vào đuôi một cơ sở dữ liệu trong không tên.

g) **retract(Term)**, (InternalDatabaseDomain) – (_): vị từ không tất định (nondeterm) này loại bỏ sự kiện chỉ ra của cơ sở dữ liệu trong không tên.

h) **retract(Term, InternalDatabaseDomain)**, (InternalDatabaseDomain) – (_, i): vị từ không tất định (nondeterm) này loại bỏ sự kiện chỉ ra của cơ sở dữ liệu trong có tên.

i) **retractall(Term)**, (InternalDatabaseDomain) – (_): vị từ này loại bỏ mọi sự kiện của cơ sở dữ liệu trong không tên. Nên sử dụng biến vô danh trong vị từ này. Vị từ này không bao giờ sai.

j) **retractall(_, InternalDatabaseDomain)**, (_, InternalDatabaseDomain) – (_, i): vị từ này loại bỏ mọi sự kiện của cơ sở dữ liệu trong có tên. Nên sử dụng biến vô danh trong vị từ này.

4. Các lệnh liên quan đến hệ điều hành DOS

a) **system(DosCommandString)**, (string) – (i): thực hiện một lệnh của hệ điều hành DOS trong môi trường Turbo Prolog.

b) **dir(Path, Filespec, Filename)**, (string, string, string) – (i, i, o): xem các files trong thư mục và chọn một file bằng menu để xem nội dung.

c) **exit**: thoát ra một chương trình và trở về môi trường làm việc của Prolog.

5. Các lệnh linh tinh khác

a) **random(RealVariable)**, (real) – (o): tạo một số ngẫu nhiên thực trong khoảng [0, 1).

b) **random(MaxValue, RandomInt)**, (integer, integer) – (i, o): tạo một số nguyên ngẫu nhiên trong khoảng [0, RandomInt).

c) **sound(Duration, Frequency)**, (integer, integer) – (i, i): tạo ra âm thanh có trường độ Duration và tần số Frequency.

d) **beep**: tạo ra một tiếng còi ngắn.

e) **date(Year, Month, Day)**, (integer, integer, integer) – (o, o, o) (i, i, i): cho ngày, tháng, năm của đồng hồ hệ thống.

f) **time(Hours, Minutes, Seconds, Hundredths)**, (integer, integer, integer, integer) – (o, o, o, o) (i, i, i, i): cho giờ của đồng hồ hệ thống.

g) **trace(on/off)**, (string) – (o) (i): đặt chế độ lần vết hay không cho chương trình.

h) **findall(Variable, Atom, ListVariable)**: ghi các giá trị của biến trong Atom trùng tên với Variable vào danh sách ListVariable.

i) **not(Atom)**: vị từ này có giá trị logic phủ định của vị từ Atom. Lưu ý: không được dùng biến tự do trong vị từ Atom.

j) **free(Variable)**: vị từ này chỉ đúng khi Variable là biến tự do.

k) **bound(Variable)**: vị từ này chỉ đúng khi Variable là biến ràng buộc.

l) **fail**: vị từ này luôn luôn sai.

m) **true**: vị từ này luôn luôn đúng.

n) **Các phép toán số học trong Turbo Prolog**: +, -, *, /, mod, div.

o) **Các phép toán quan hệ trong Turbo Prolog** (có giá trị không chỉ đối với các số mà còn đối với các ký tự, chuỗi, symbol): >, <, =, >=, <=, <>, ><.

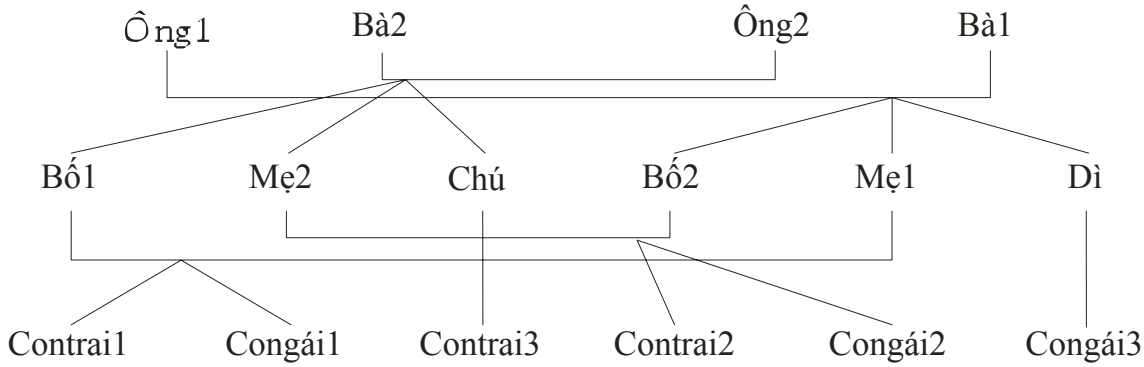
p) **Các hàm trong Turbo Prolog**: sin, cos, tan, arctan, ln, log, exp, sqrt, round (số nguyên làm tròn), trunc (số nguyên chặt cụt), abs.

BÀI TẬP CHƯƠNG IV

(Phần bài tập làm quen với vị từ trong Prolog)

Bài 1:

Xét cây gia phả sau:



Cây được mô tả bởi các vị từ quan hệ sau: cha(Cha, Con), mẹ(Mẹ, Con), nam(DO), nữ (DB).

Hãy liệt kê các quan hệ còn lại như: anhEm(Ng1, Ng2), con(Con, Cha, Mẹ), chú(Chú, Cháu), bà(Bà, Cháu), anhEmHọ(Ng1, Ng2)... từ những vị từ trên.

Suy ra mối quan hệ giữa hai người bất kỳ.

Bài 2:

Một dịch vụ du lịch phục vụ những chuyến đi trong 1 tuần hoặc 2 tuần ở Rome, London, Tunis.

Trong một quyển sách giới thiệu về dịch vụ du lịch, với mỗi địa điểm có giá cả đi lại (tùy vào thời gian của chuyến đi) và chi phí cho 1 tuần, tùy vào địa điểm và mức độ tiện nghi: khách sạn, nhà trọ, cắm trại ngoài trời.

a) Nhập vào các sự kiện mô tả quyển sách trên (giá cả do bạn nhập vào).
b) Hãy mô tả quan hệ ChuyếnĐi(TP, SốTuần, TiệnNghi, ChiPhí).
c) Mô tả quan hệ ChuyếnĐi_ÍtTốnKém(TP, SốTuần, TiệnNghi, ChiPhí, ChiPhíMax) sao cho chi phí của chuyến đi thấp hơn ChiPhíMax.

Bài 3:

Trong một dịch vụ tư vấn hôn nhân, liệt kê một danh sách các cặp xứng đôi có thể làm quen theo các nhận định sau:

- nam(n, t, c, a), nữ(n, t, c, a).
  n: tên; t: chiều cao.
  c: màu tóc (vàng, hung, nâu, hạt dẻ).
  a: tuổi (trẻ, trung niên, già).

- SởThích(n, m, l, s) (sở thích) có nghĩa là: Người n thích loại nhạc m (classique, pop, jazz), thích văn học loại l (phiêu lưu, văn học viễn tưởng, trinh thám), và chơi môn thể thao s(quần_vợt, bơi, chạy bộ).
- tìm(n, t, c, a): Người n tìm bạn có các tính chất t, c, a.

Hai người x và y gọi là hợp nhau nếu có cùng ngoại hình (t, c, a) và sở thích về: l, m, s.

a) Nhập dữ liệu về các khách hàng.
b) Mô tả qui tắc hợp_ngoạihình(x, y) và cùng_sở thích(x, y).
c) Viết chương trình xác định những cặp xứng đôi.

(Phần bài tập về danh sách, đệ qui, lát cắt)

Bài 4:

Xây dựng các vị từ đệ quy (có hoặc không sử dụng nhát cắt và sửa đổi phần đuôi của đệ quy, nếu có thể) để:

a) Tìm (hoặc xóa) phần tử thứ 1, thứ 2, phần tử cuối cùng trong một danh sách.
b) Tìm (hoặc lấy ra) phần tử đầu tiên trong danh sách trùng với một giá trị cho trước.
c) Tìm và lấy ra tất cả các phần tử trong danh sách trùng với một giá trị cho trước.
d) Tính giá trị trung bình cộng của một danh sách kiểu số thực.
e) Xuất ra tất cả các cặp phần tử kế tiếp nhau của danh sách.
f) Tạo danh sách tất cả các phần tử ở vị trí chẵn của danh sách dữ liệu L:
    - Giữ nguyên trật tự xuất hiện trong L.
    - Đảo ngược trật tự.
g) Viết chương trình trộn(u, v, w) nhận xen kẽ từng phần của u và v để tạo thành danh sách w.
h) Tạo danh sách chứa các số chẵn của một danh sách số nguyên cho trước.
i) Đảo ngược trật tự mọi phần tử của một danh sách.
j) Xuất ra mọi hoán vị của những phần tử của danh sách cho trước.
k) Xây dựng vị từ Thuộc trong đó có thêm một đối để trả về trị TRUE nếu phần tử x thuộc danh sách l cho trước và FALSE trong trường hợp ngược lại.
l) Xây dựng qui tắc NằmNgoài(x, L) để kiểm tra xem x không thuộc danh sách L.
m) Xây dựng qui tắc KhácNhau(L) để kiểm tra danh sách L không có phần tử trùng nhau.
n) Xây dựng qui tắc PhầnĐầu(m, L) để kiểm tra danh sách m là phần đầu của danh sách L.
o) Xây dựng qui tắc NằmTrongLiênTiếp(m, L) để kiểm tra mọi phần tử của m đều nằm liên tiếp trong danh sách L.

p) Xây dựng qui tắc PhânHoạch(L, u, v, w) để phân hoạch danh sách L thành 3 danh sách u, v, w.
q) Thêm một phần tử vào cuối danh sách.
r) Tìm và xóa phần tử thứ k của danh sách l cho trước.
s) Chèn một phần tử x sau phần tử thứ k của danh sách L (0 <= k <= length(L)).
t) Xây dựng danh sách mà các phần tử là căn bậc hai của các phần tử tương ứng trong danh sách L các số thực cho trước.

Bài 5:

Tìm n bộ 3 số tự nhiên (x, y, z) đầu tiên thỏa tính chất Pythagore sau:

$0 < x < y < z$

$x^2 + y^2 = z^2$

UCLN(x, y, 1): Ước chung lớn nhất của x và y bằng 1.

Bài 6 (+):

Tìm n_k số tự nhiên đầu tiên $\overline{a_1, a_2, ..., a_n}$ thỏa:

$$\overline{a_1.a_2.a_n} = \sum_{i=1}^{n} a_i^k \qquad (k = 3, 4, 5)$$

VD: k = 3, $153 = 1^3 + 5^3 + 3^3$.

Bài 7:

Kiểm tra số tự nhiên N có phải là số nguyên tố không? Tìm n số nguyên tố đầu tiên.

Bài 8:

Tìm các số nguyên tố <= n (sàng Eratosthene).

Bài 9:

a) Nhập 2 danh sách số nguyên dương không có phần tử trùng lặp từ bàn phím.
b) Xây dựng các phép toán và các quan hệ trên tập hợp: Hội, Giao, Hiệu, Thuộc, Bao hàm.

Bài 10:

Cài đặt các thuật toán sắp xếp: chọn, chèn, tráo đổi đơn giản và sắp xếp nhanh.

Bài 11 (+):

Các đỉnh của một đồ thị được đánh số từ 0 -> n. Mỗi cung của đồ thị tương ứng với qui tắc Cung(i, j) nối gốc i với ngọn j.

Xây dựng qui tắc ĐườngĐi(x, y, L) với L là danh sách biểu diễn đường đi không lặp từ x đến y.

Bài 12 (+):

Cài đặt cây nhị phân tìm kiếm và các thao tác cơ bản.

Bài 13:

Giải quyết bài toán "Tháp Hà Nội" bằng đệ quy.

Bài 14 (*): (8 hậu)

Đặt 8 con hậu lên bàn cờ sao cho không con nào ăn con nào.

Bài 15 (*): (Dominos)

Có một danh sách các quân cờ đôminô. Mỗi quân cờ đôminô được biểu diễn bởi Cặp(i, j) ứng với 2 con số của nó. Người ta tìm cách sắp xếp chúng theo tính chất sau: Mỗi con đôminô có thể xếp vào một trong hai đầu của hàng đã xếp.

Xuất phát từ một con đôminô trong số các con được cho, hãy xây dựng một dãy hợp lệ chứa tất cả các đôminô.

Bài 16 (*):

Giải quyết bài toán ma phương cấp n lẻ.

(Phần bài tập về trò chơi, suy luận lôgic)

Bài 17: (Ngựa vằn)

Có 5 người quốc tịch khác nhau sống trong 5 căn nhà màu khác nhau. Mỗi người có một con thú riêng, một thức uống riêng, và hút thuốc của những hãng khác nhau.

- Người Anh sống trong căn nhà màu đỏ.
- Con chó thuộc về người Tây Ban Nha.
- Người uống cà phê trong căn nhà màu xanh lá cây.
- Người Ukrain uống trà.
- Ngôi nhà xanh lá cây ở cạnh ngôi nhà trắng, bên phải.
- Người hút thuốc OldGold nuôi những con sên.
- Người uống sữa ở ngôi nhà chính giữa.
- Người hút thuốc Kool sống trong căn nhà màu vàng.
- Người Nauy sống trong căn nhà đầu tiên bên trái.
- Người hút thuốc Chesterfield sống cạnh người có nuôi cáo.
- Người hút thuốc Kool sống cạnh người có nuôi ngựa.
- Người hút thuốc Gitanes uống rượu.
- Người Nhật hút thuốc Craven
- Người Nauy sống cạnh ngôi nhà màu xanh da trời.

Hỏi: Ai uống nước? Ai là chủ con ngựa vằn?

Bài 18 (*): (Trò chơi tốt đen và tốt trắng)

Bắt đầu từ 3 tốt đen và 3 tốt trắng đã được sắp xếp và cách nhau bởi một khoảng trắng.

VD: BBB_NNN.

Mỗi lần đánh chỉ có 1 trong 4 phép dịch chuyển sau được thi hành:

Tốt đen trượt sang trái:

BNB_NNB -> BNBN_NB

Tốt đen nhảy sang trái:

BN_BNNB -> BNNB_NB

Tốt trắng trượt sang phải:

BNB_NNB -> BN_BNNB

Tốt trắng nhảy sang phải:

BBN_NNB -> B_NBNNB

Viết 1 chương trình để chuyển từ trạng thái đầu sang trạng thái kết (mọi tốt trắng đều nằm bên phải).

Bài 19 (*):

Cho 1 biểu thức với phép * (Nhan) và + (Cong). Hãy sửa đổi để được 1 tổng các tích bằng cách thêm vào dấu ngoặc.

VD: a * (b + c) + (d + e) * (f + g) = (((((a * b + a * c) + d * f) + e * f) + d * g) + e * g)

Có nghĩa là:

Cong(Nhan(a, Cong(b, c), Nhan(Cong(d, e), Cong(f, g)))) cho ra Cong(Cong(Cong(Cong(Cong(Nhan(a, b), Nhan(a, c)), Nhan(d, f)), Nhan(e, f)), Nhan(d, g)), Nhan(e, g))

Hãy xây dựng chương trình trên theo 2 giai đoạn:

- Phân phối Nhan sao cho không có Nhan nào có Cong làm đối.
- Chuyển thành cây sao cho nhánh phải của Cong không chứa Cong khác, nhánh phải của Nhan không chứa Nhan khác.

Bài 20 (*):

Cho biểu thức dạng And(p, q), Or(p, q), Not(q), trong đó p, q là biến hoặc là biểu thức cùng dạng. Ta có:

And(Or(p, q), r) <=> Or(And(p, r), And(q, r))
And(p, Or(q, r)) <=> Or(And(p, q), And(p, r))
Not(Or(p, q)) <=> And(Not(p), Not(q))
Not(And(p, q)) <=> Or(not(p), not(q))
Not(Not(p)) <=> p

Hãy chuyển biểu thức theo yêu cầu sau:

Mỗi nút Or chỉ có tổ tiên là Or.
Mỗi nút And chỉ có tổ tiên là And.

a) Viết chương trình thực hiện công việc trên.
b) Thay đổi chương trình để mỗi nhánh trái Or không chứa nút Or khác, và mỗi nhánh trái And không chứa And khác.

Bài 21 (*): (Những ông chồng hay ghen)

Có ba cặp vợ chồng phải qua sông nhưng chỉ có một chiếc thuyền chỉ chở được tối đa hai người. Hãy lập trình cho các chuyến đi để sao cho không có bà vợ nào ở trên bờ hay trên thuyền với những người đàn ông X khác mà không có chồng mình hoặc không có đồng thời với vợ của X.

Bài 22 (*): (Sói, dê và bắp cải)

Có một con sói, một con dê và một bắp cải phải qua sông. Chỉ có một người chèo thuyền, chỉ chở được mỗi lần một trong số chúng. Nếu không có mặt người chèo thuyền thì dê sẽ ăn bắp cải và sói sẽ ăn thịt dê. Hãy lập trình cho các chuyến đi.

TÀI LIỆU THAM KHẢO

[1] Bạch Hưng Khang, Hoàng Kiếm. *Trí tuệ nhân tạo - Các phương pháp và ứng dụng*. NXB Khoa học Kỹ thuật, 1989.
[2] Hoàng Kiếm. *Giải một bài toán trên máy tính như thế nào (tập 1 và 2).* NXB Giáo dục, 2001.
[3] Nguyễn Thanh Thủy. *Trí tuệ nhân tạo - Các phương pháp giải quyết vấn đề và kỹ thuật xử lý tri thức.* NXB Giáo dục, 1995.
[4] A. Thayse. *Approche logique de l'intelligence artificielle (T1, T2, T3).* Dunod, Paris, 1990.
[5]. Ivan Bratko, *Prolog - Programming for artificial intelligence,* Addison-Wesley, 1990.