Chương 2

GIẢI QUYẾT VẤN ĐỀ BẰNG TÌM KIẾM

Nhiều vấn đề (bài toán) có thể phát biểu và giải quyết dưới dạng tìm kiếm. Chương 2 trình bày những kiến thức về giải quyết vấn đề bằng tìm kiếm; cụ thể như sau: các chiến lược tìm kiếm (tìm kiếm theo chiều rộng, tìm kiếm theo chiều sâu,...); phân rã bài toán thành các vấn đề con; tìm kiếm trên đồ thị và/hoặc; các chiến lược tìm kiếm tối ưu.

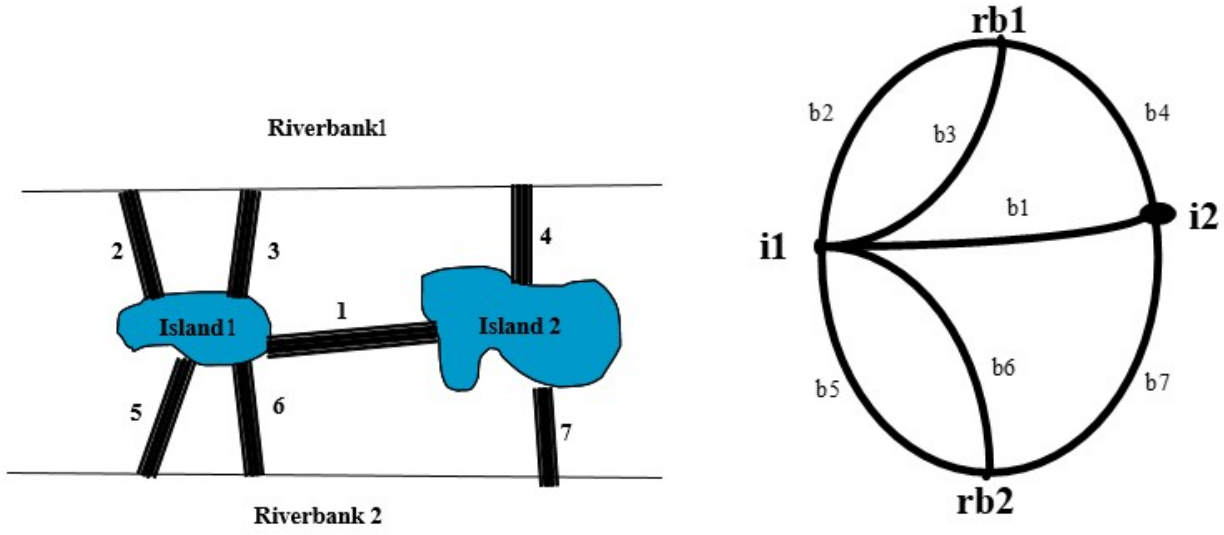
2.1. CÁC CHIẾN LƯỢC TÌM KIẾM MÙ

2.1.1. Biểu diễn vấn đề trong không gian trạng thái

Khi giải quyết một vấn đề, thông thường ta xác định trực tiếp lời giải thông qua một thủ tục tính toán hoặc các bước căn bản để có được lời giải. Có ba phương pháp chính để xác định trực tiếp lời giải. Phương pháp thứ nhất được áp dụng để giải các bài toán đã biết cách giải bằng các công thức toán học chính xác. Phương pháp thứ hai được dùng cho các bài toán đã biết cách giải bằng các xây dựng các công thức xấp xỉ. Phương pháp thứ ba được áp dụng vào các bài toán đã biết cách giải không tường minh thông qua việc truy hồi hay kỹ thuật đệ qui. Tuy nhiên không phải lúc nào cũng có thể giải quyết được bài toán bằng phương pháp xác định trực tiếp lời giải. Khi đó ta sẽ sử dụng một số phương pháp khác như tìm kiếm, thử - sai, kinh nghiệm, …

Khi muốn giải quyết một vấn đề nào đó bằng tìm kiếm, cần phải xác định không gian tìm kiếm. Không gian tìm kiếm bao gồm tất cả các đối tượng mà ta cần quan tâm tìm kiếm. Nó có thể là không gian liên tục, chẳng hạn không gian các véctơ thực n chiều; nó cũng có thể là không gian các đối tượng rời rạc.

Khi biểu diễn một vấn đề như là một đồ thị không gian trạng thái, chúng ta có thể sử dụng lý thuyết đồ thị để phân tích cấu trúc và độ phức tạp của các vấn đề cũng như các thủ tục tìm kiếm.



Hình 2.1: Bản đồ và biểu diễn đồ thị tương ứng

Toán tử: mô tả hành động hoặc phép biến đổi để đưa một trạng thái tới trạng thái khác

Ví dụ 2.1: Trong bài toán tìm đường đi, các con đường nối các thành phố sẽ được biểu diễn bởi các toán tử. Giải bài toán bằng tìm một dãy các toán tử để đưa trạng thái ban đầu (điểm xuất phát) về trạng thái kết thúc (điểm đích).

Như vậy muốn biểu diễn một vấn đề trong không gian trạng thái, ta cần xác định các yếu tố sau:

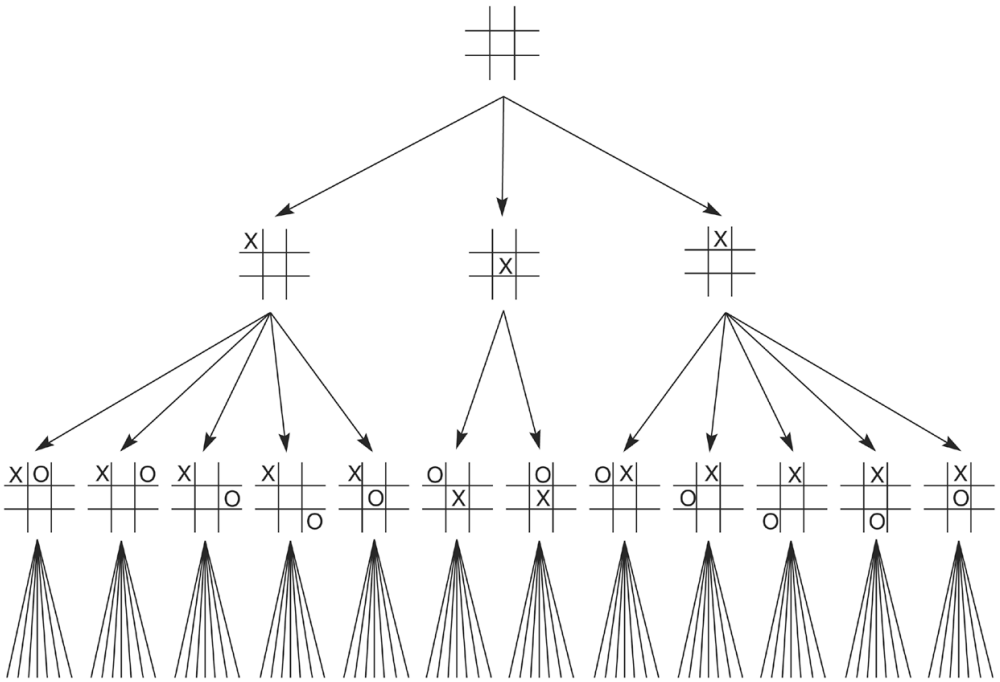
- ✓ Trạng thái ban đầu.
- ✓ Một tập hợp các toán tử. Trong đó mỗi toán tử mô tả một hành động hoặc một phép biến đổi có thể đưa một trạng thái tới một trạng thái khác. Tập hợp tất cả các trạng thái có thể đạt tới từ trạng thái ban đầu bằng cách áp dụng một dãy toán tử, lập thành không gian trạng thái của vấn đề. Ta sẽ ký hiệu không gian trạng thái là U, trạng thái ban đầu là $u_0 (u_0 U)$. Mỗi toán tử R có thể xem như một ánh xạ $R: U U$. Nói chung R là một ánh xạ không xác định khắp nơi trên U.
- ✓ Một tập hợp T các trạng thái kết thúc (trạng thái đích). T là tập con của không gian U. Trong nhiều vấn đề (chẳng hạn khi chơi cờ) có thể có nhiều trạng thái đích và ta không thể xác định trước được các trạng thái đích. Do đó, ta chỉ có thể mô tả các trạng thái đích là các trạng thái thỏa mãn một số điều kiện nào đó.

Khi biểu diễn một vấn đề thông qua các trạng thái và các toán tử, thì việc tìm nghiệm của bài toán được quy về việc tìm đường đi từ trạng thái ban đầu tới trạng thái đích (Một đường đi trong không gian trạng thái là một dãy toán tử dẫn một trạng thái tới một trạng thái khác).

Chúng ta có thể biểu diễn không gian trạng thái bằng đồ thị định hướng, trong đó mỗi đỉnh của đồ thị tương ứng với một trạng thái. Nếu có toán tử R biến đổi trạng thái u thành trạng thái v, thì có cung gán nhãn R đi từ đỉnh u tới đỉnh v. Khi đó một đường đi trong không gian trạng thái sẽ là một đường đi trong đồ thị này.

Sau đây chúng ta sẽ xét một số ví dụ về các không gian trạng thái được xây dựng cho một số vấn đề.

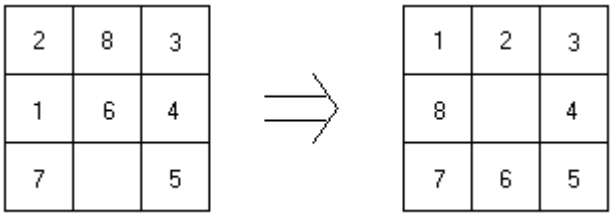
Ví dụ 2.2: Tic-tac-toe là một trò chơi phổ biến dùng viết trên bàn cờ giấy có chín ô, 3x3. Hai người chơi, một người dùng ký hiệu O, một người dùng ký hiệu X, lần lượt điền ký hiệu của mình vào các ô. Người thắng là người có thể tạo được đầu tiên một dãy ty ký hiệu của mình, ngang dọc hay chéo đều được.



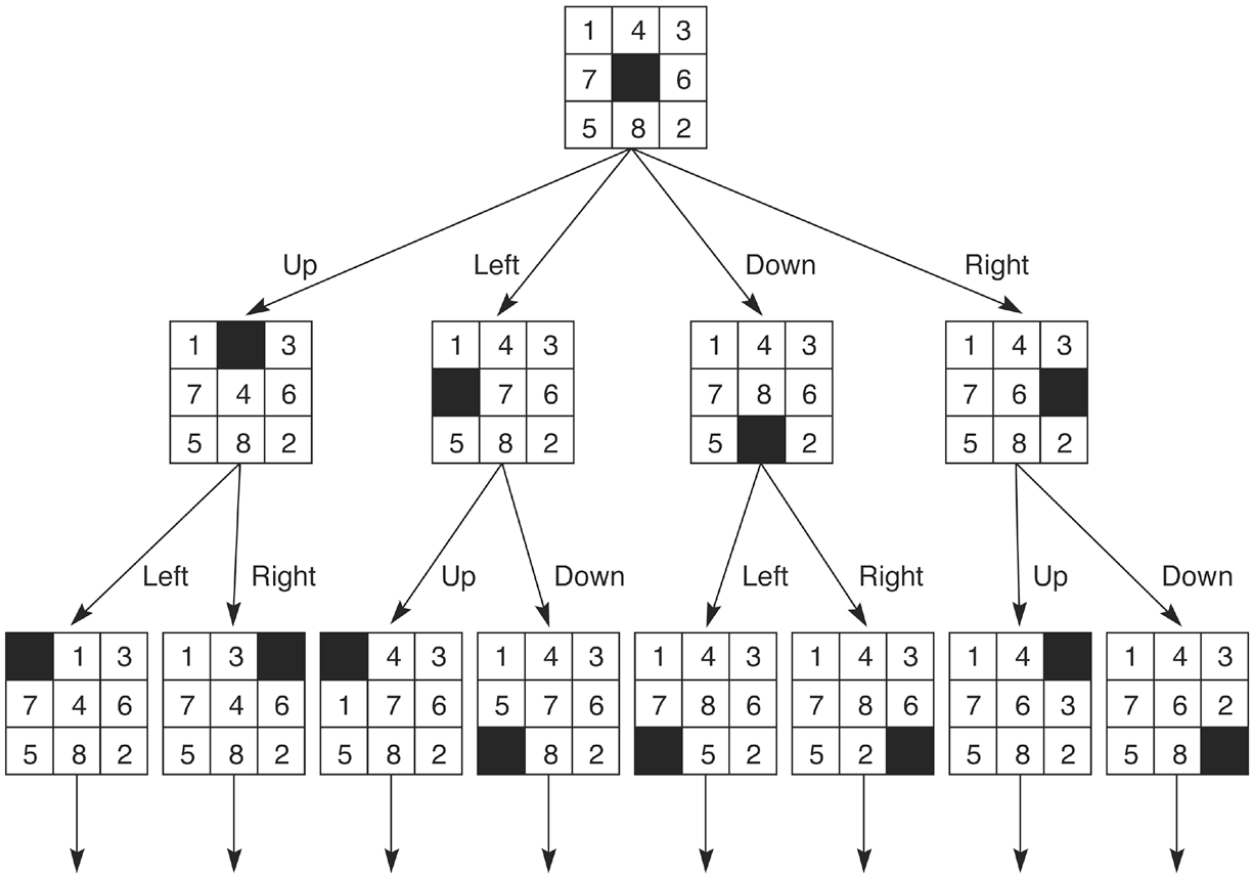
Hình 2.2: Một phần không gian trạng thái của trò chơi Tic-tac-toe

Ví dụ 2.3: Bài toán 8 số. Có bảng 3x3 ô và tám quân mang số hiệu từ 1 đến 8 được xếp vào tám ô, còn lại một ô trống, chẳng hạn như trong Hình 2.2. Trong trò chơi này, có thể chuyển dịch các quân ở cạnh ô trống tới ô trống đó. Cần tìm ra một dãy các chuyển dịch để biến đổi hình ban đầu (bên trái) thành hình bên phải.

Trong bài toán này, trạng thái ban đầu được mô tả ở hình bên trái, còn trạng thái kết thúc ở bên phải hình. Tương ứng với các quy tắc chuyển dịch các quân, ta có bốn toán tử: *up* (đẩy quân lên trên), *down* (đẩy quân xuống dưới), *left* (đẩy quân sang trái), *right* (đẩy quân sang phải). Rõ ràng là, các toán tử này chỉ là các toán tử bộ phận; chẳng hạn, từ trạng thái ban đầu (hình bên trái), ta chỉ có thể áp dụng các toán tử *down, left, right*.

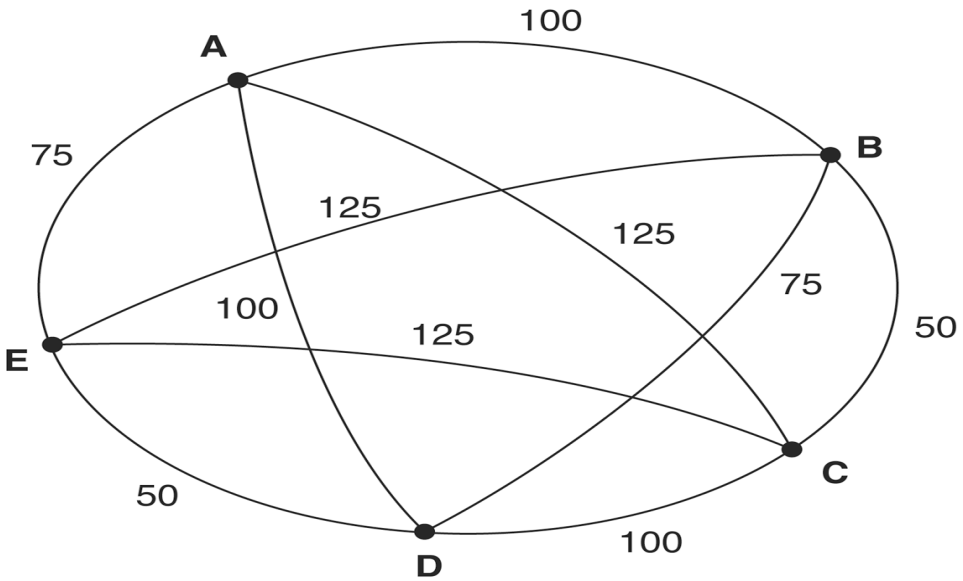


Hình 2.3: Bài toán 8 số

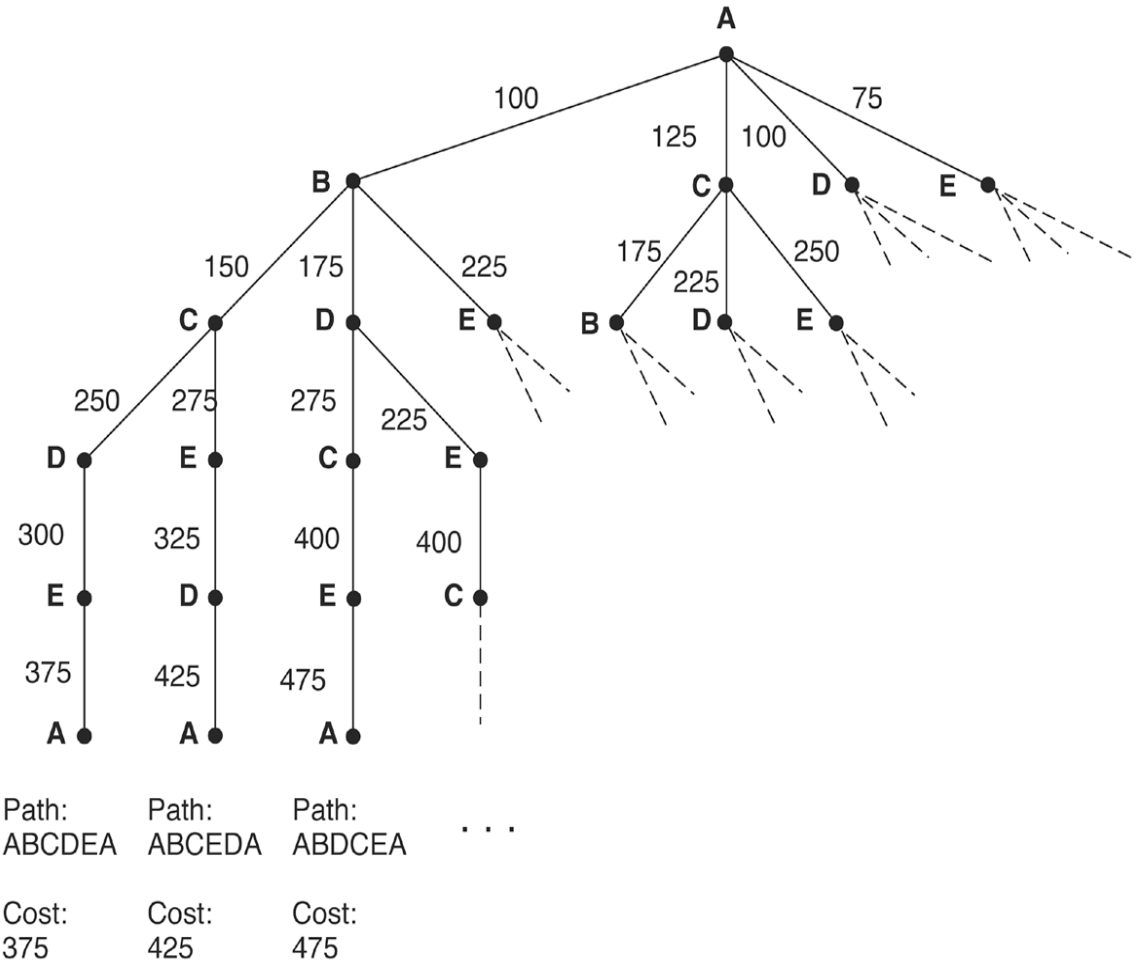


Hình 2.4: Một ví dụ của không gian trạng thái bằng cách dịch chuyển ô trống

Ví dụ 2 4: Bài toán TSP. Biểu diễn không gian trạng thái cho bài toán TSP như trong hình 2.5.



Hình 2.5: Bài toán TSP



Hình 2.6: Không gian trạng thái của bài toán TSP

Trong Hình 2.6, mỗi cung được đánh dấu bằng tổng chi phí của con đường từ nút bắt đầu đến nút hiện tại.

Ví dụ 2.5: Cho hai bình đựng chất lỏng, một bình có dung tích 4 lít và một bình có dung tích 3 lít. Cả hai bình không có dấu dung tích. Có thể dùng một đường ống để làm đầy nước ở hai bình. Làm thế nào để có chính xác 2 lít nước trong bình 4 lít. Hãy biểu diễn không gian của bài toán bằng đồ thị?

Không gian của bài toán có thể được mô tả bằng các cặp số nguyên (x, y), trong đó x = 0, 1, 2, 3, 4 biểu diễn số lít nước trong bình 4 lít và y = 0, 1, 2, 3 biểu diễn số lít nước trong bình 3 lít. Trạng thái ban đầu của bài toán là hai bình đều rỗng, do đó ta có cặp số nguyên (0, 0). Trạng thái đích của bài toán là có 2 lít nước trong bình 4 lít, do đó ta sẽ có (2, n), với n là giá trị bất kỳ từ 0→3.

Không gian của bài toán được định nghĩa bằng tri thức thủ tục của bài toán đó chính là các luật để giải bài toán được thiết kế như sau:

- Luật 1: Nếu x < 4 thì làm đầy bình 4 lít: $(x, y \mid x < 4) \rightarrow (4, y)$
- Luật 2: Nếu y < 3 thì làm đầy bình 3 lít: $(x, y \mid y < 3) \rightarrow (x, 3)$
- Luật 3: Nếu x > 0 thì làm rỗng bình 4 lít: $(x, y \mid x > 0) \rightarrow (0, y)$
- Luật 4: Nếu y > 0 thì làm rỗng bình 3 lít: $(x, y \mid y > 0) \rightarrow (x, 0)$

- Luật 5: Nếu x + y >=4 và y > 0 thì đưa nước từ bình 3 lít sang bình 4 lít cho đến khi bình 4 lít đầy: (x, y | x + y >=4 và y > 0) → (4, y – (4 – x))

- Luật 6: Nếu x + y >=3 và x > 0 thì đưa nước từ bình 4 lít sang bình 3 lít cho đến khi bình 3 lít đầy: (x, y | x + y >=3 và x > 0) → (x – (3 – y), 3)

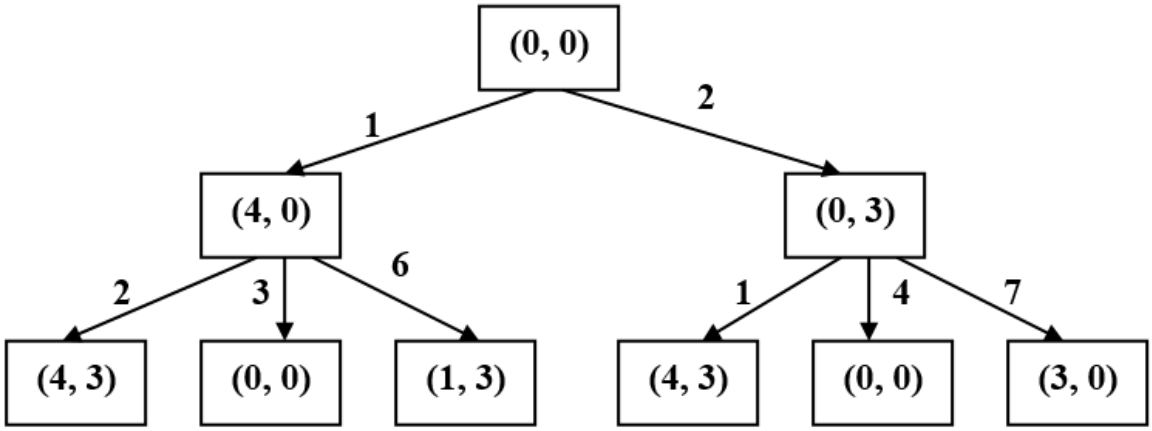
- Luật 7: Nếu x + y <=4 và y > 0 thì đưa tất cả nước từ bình 3 lít sang bình 4 lít: (x, y | x + y <=4 và y > 0) → (x + y, 0)

- Luật 8: Nếu x + y <=3 và x > 0 thì đưa tất cả nước từ bình 4 lít sang bình 3 lít: (x, y | x + y <=3 và x > 0) → (0, x + y)

Từ trạng thái ban đầu của bài toán là (0, 0) thoả mãn hai luật 1 và 2 phát sinh 2 trạng thái mới (4, 0) và (0, 3).

Tại trạng thái mới (4, 0) thoả mãn ba luật 2, 3, 6 phát sinh ra 3 trạng thái mới hơn là (4, 3), (0, 0) và (1, 3).

Tại trạng thái mới (0, 3) cũng thoả mãn ba luật 1, 4, 7 phát sinh ra 3 trạng thái mới hơn là (4, 3), (0, 0) và (3, 0). Quá trình phát sinh như thế cứ tiếp diễn cho đến khi có một trạng thái bất kỳ (2, n) xuất hiện thì dừng. Số trạng thái được phát sinh kể cả trạng thái ban đầu và trạng thái đích được gọi là không gian của bài toán.



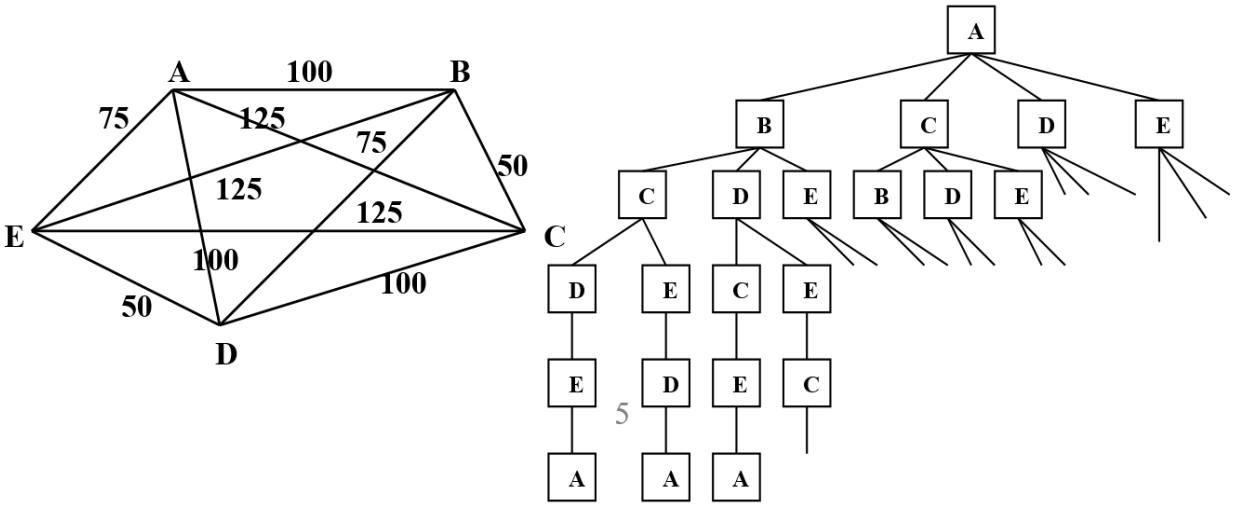
Hình 2.7: Một phần không gian trạng thái của bài toán đựng nước (biểu diễn bằng đồ thị)

Ví dụ 2.6: Xét bài toán hành trình người bán hàng. Bài toán người bán hàng (Travelling Salesman Problem - TSP) là một bài toán thuộc thể loại tối ưu rời rạc hay tổ hợp được nghiên cứu trong vận trù học hoặc lý thuyết khoa học máy tính. Bài toán được phát biểu như sau. Cho trước một danh sách các thành phố và khoảng cách giữa chúng, tìm chu trình ngắn nhất thăm mỗi thành phố đúng một lần.

Giả sử người bán hàng có năm thành phố cần đến giao hàng và sau đó phải trở về nhà. Mục đích của bài toán là tìm đường đi ngắn nhất cho cuộc hành trình người bán hàng để đi đến tất cả các thành phố, mỗi thành phố ông ta chỉ đến một lần và sau đó trở về lại thành phố bắt đầu cuộc hành trình. Hình 2.8 là một ví dụ cụ thể của bài toán trên. Hãy biểu diễn không gian trạng thái của bài toán

Giả sử cuộc hành trình của người bán hàng bắt đầu từ thành phố A và trở về lại A. Không gian của bài toán này là số đường đi khác nhau, trong đó sẽ có một đường đi ngắn

nhất cho cuộc hành trình. Nếu cuộc hành trình đi qua n thành phố, ta sẽ có số $(n-1)!$ đường đi khác nhau. Hình 2.7 (phải) mô tả một phần không gian trạng thái của bài toán.



Hình 2.8: Ví dụ bài toán hành trình người bán hàng và một phần không gian trạng thái

Trong các ví dụ trên việc tìm ra một biểu diễn thích hợp để mô tả các trạng thái của vấn đề là khá dễ dàng và tự nhiên. Song trong nhiều vấn đề việc tìm hiểu được biểu diễn thích hợp cho các trạng thái của vấn đề là hoàn toàn không đơn giản. Việc tìm ra dạng biểu diễn tốt cho các trạng thái đóng vai trò hết sức quan trọng trong quá trình giải quyết một vấn đề. Có thể nói rằng, nếu ta tìm được dạng biểu diễn tốt cho các trạng thái của vấn đề, thì vấn đề hầu như đã được giải quyết.

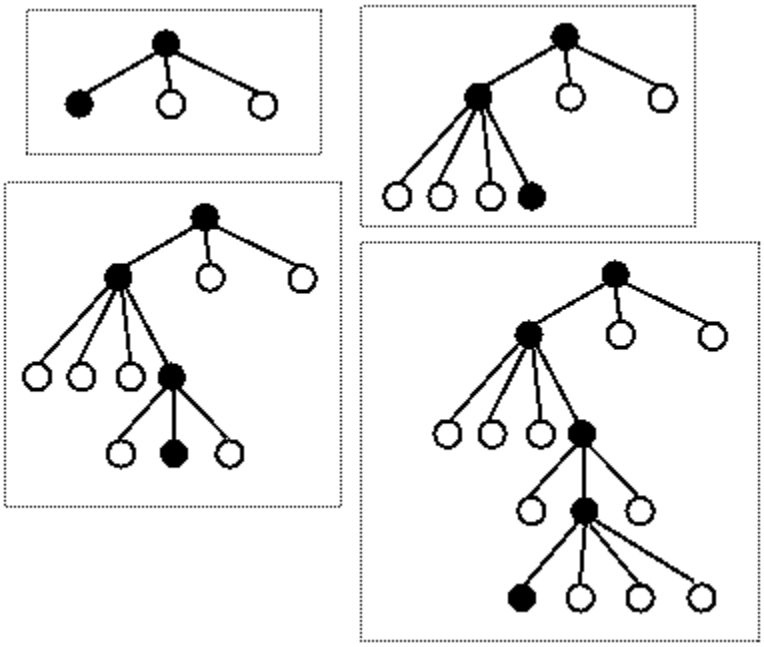
2.1.2. Các chiến lược tìm kiếm

Phân loại

Có thể phân các chiến lược tìm kiếm thành 02 loại:

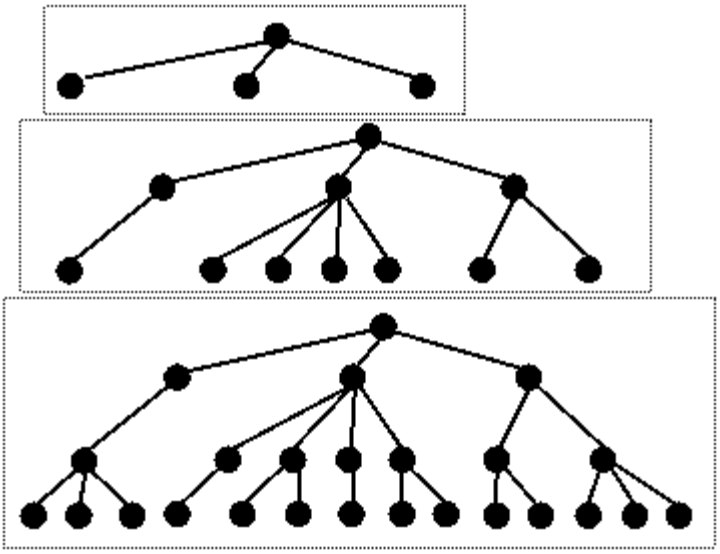
- ✓ Tìm kiếm mù (blind searches): không có sự hướng dẫn nào cho tìm kiếm, chỉ phát triển các trạng thái ban đầu cho tới khi gặp một trạng thái đích nào đó. Có hai kỹ thuật tìm kiếm mù, đó là tìm kiếm theo chiều rộng và tìm kiếm theo chiều sâu.

Trong tìm kiếm theo chiều sâu, tại trạng thái (đỉnh) hiện hành, ta chọn một trạng thái kế tiếp (trong tập các trạng thái có thể biến đổi thành từ trạng thái hiện tại) làm trạng thái hiện hành cho đến lúc trạng thái hiện hành là trạng thái đích. Trong trường hợp tại trạng thái hiện hành, ta không thể biến đổi thành trạng thái kế tiếp thì ta sẽ quay lui (back-tracking) lại trạng thái trước trạng thái hiện hành (trạng thái biến đổi thành trạng thái hiện hành) để chọn đường khác. Nếu ở trạng thái trước này mà cũng không thể biến đổi được nữa thì ta quay lui lại trạng thái trước nữa và cứ thế. Nếu đã quay lui đến trạng thái khởi đầu mà vẫn thất bại thì kết luận là không có lời giải. Hình 2.6 sau minh họa hoạt động của tìm kiếm theo chiều sâu.



Hình 2.9: Tìm kiếm theo chiều sâu

Tìm kiếm theo chiều rộng là các trạng thái được phát triển theo thứ tự mà chúng được sinh ra, tức là trạng thái nào được sinh ra trước sẽ được phát triển trước. Ngược lại với tìm kiếm theo kiểu chiều sâu, tìm kiếm chiều rộng mang hình ảnh của vết dầu loang. Từ trạng thái ban đầu, ta xây dựng tập hợp S bao gồm các trạng thái kế tiếp (mà từ trạng thái ban đầu có thể biến đổi thành). Sau đó, ứng với mỗi trạng thái T_k trong tập S, ta xây dựng tập S_k bao gồm các trạng thái kế tiếp của T_k rồi lần lượt bổ sung các Sk vào S. Quá trình này cứ lặp lại cho đến lúc S có chứa trạng thái kết thúc hoặc S không thay đổi sau khi đã bổ sung tất cả S_k.



Hình 2.10: Tìm kiếm theo chiều rộng

Trong nhiều vấn đề, dù phát triển các trạng thái theo chiều rộng hoặc theo chiều sâu thì số lượng các trạng thái được sinh ra trước khi ta gặp trạng thái đích thường là cực kỳ lớn. Do đó các thuật toán tìm kiếm mù kém hiệu quả, đòi hỏi rất nhiều không gian và thời gian. Trong thực tế, nhiều vấn đề không thể giải quyết được bằng tìm kiếm mù.

Bảng 1.1: Kỹ thuật tìm kiếm theo chiều rộng và chiều sâu

	Chiều sâu	**Chiều rộng**
Tính hiệu quả	Hiệu quả khi lời giải nằm sâu trong cây tìm kiếm và có một phương án chọn hướng đi chính xác. Hiệu quả của chiến lược phụ thuộc vào phương án chọn hướng đi. Phương án càng kém hiệu quả thì hiệu quả của chiến lược càng giảm. Thuận lợi khi muốn tìm chỉ một lời giải.	Hiệu quả khi lời giải nằm gần gốc của cây tìm kiếm. Hiệu quả của chiến lược phụ thuộc vào độ sâu của lời giải. Lời giải càng xa gốc thì hiệu quả của chiến lược càng giảm. Thuận lợi khi muốn tìm nhiều lời giải.
Lượng bộ nhớ sử dụng để lưu trữ các trạng thái	Chỉ lưu lại các trạng thái chưa xét đến.	Phải lưu toàn bộ các trạng thái.
Trường hợp xấu nhất	Vét cạn toàn bộ	Vét cạn toàn bộ.
Trường hợp tốt nhất	Phương án chọn hướng đi *tuyệt đối* chính xác. Lời giải được xác định một cách trực tiếp.	Vét cạn toàn bộ.

Tìm kiếm chiều sâu và tìm kiếm chiều rộng đều là các phương pháp tìm kiếm có hệ thống và chắc chắn tìm ra lời giải. Tuy nhiên, do bản chất là vét cạn nên với những bài toán có không gian lớn thì ta không thể dùng hai chiến lược này được. Hơn nữa, hai chiến lược này đều có tính chất "mù quáng" vì chúng không chú ý đến những thông tin (tri thức) ở trạng thái hiện thời và thông tin về đích cần đạt tới cùng mối quan hệ giữa chúng. Các tri thức này vô cùng quan trọng và rất có ý nghĩa để thiết kế các thuật giải hiệu quả hơn mà ta sắp sửa bàn đến.

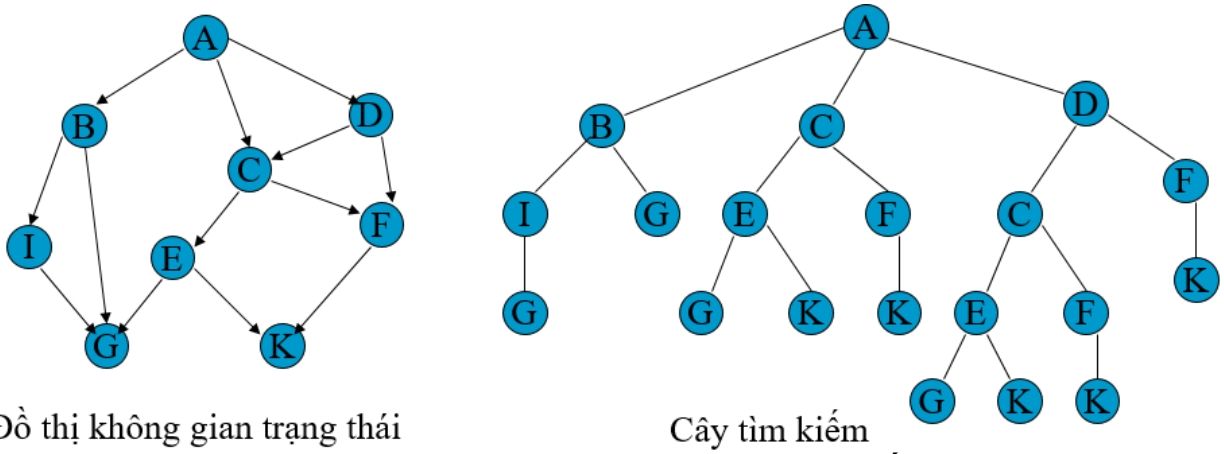
✓ Tìm kiếm kinh nghiệm (heuristic): tìm kiếm dựa vào hiểu biết về các vấn đề, dựa vào kinh nghiệm, trực giác để đánh giá các trạng thái.

Như vậy chiến lược tìm kiếm được xác định bởi chiến lược chọn trạng thái để phát triển ở mỗi bước. Trong tìm kiếm mù, ta chọn trạng thái để phát triển theo thứ tự mà đúng được sinh ra; còn trong tìm kiếm kinh nghiệm ta chọn trạng thái dựa vào sự đánh giá các trạng thái.

Cây tìm kiếm

Quá trình tìm kiếm được xem như quá trình xây dựng cây tìm kiếm. Gốc của cây tìm kiếm tương ứng với trạng thái ban đầu. Nếu một đỉnh ứng với trạng thái u, thì các

đỉnh con của nó ứng với các trạng thái *v* kề *u*. Hình 2.11 biểu diễn một không gian trạng thái với trạng thái ban đầu là A, hình bên phải là cây tìm kiếm tương ứng với không gian trạng thái đó.



Đồ thị không gian trạng thái Cây tìm kiếm

Hình 2.11: Không gian trạng thái và cây tìm kiếm tương ứng

Mỗi chiến lược tìm kiếm trong không gian trạng thái tương ứng với một phương pháp xây dựng cây tìm kiếm. Quá trình xây dựng cây bắt đầu từ cây chỉ có một đỉnh là trạng thái ban đầu. Giả sử tới một bước nào đó trong chiến lược tìm kiếm, ta đã xây dựng được một cây nào đó, các lá của cây tương ứng với các trạng thái chưa được phát triển. Bước tiếp theo phụ thuộc vào chiến lược tìm kiếm mà một đỉnh nào đó trong các lá được chọn để phát triển. Khi phát triển đỉnh đó, cây tìm kiếm được mở rộng bằng cách thêm vào các đỉnh con của đỉnh đó. Kỹ thuật tìm kiếm theo bề rộng (theo độ sâu) tương ứng với phương pháp xây dựng cây tìm kiếm theo bề rộng (theo độ sâu).

2.1.3 Các chiến lược tìm kiếm mù

Tìm kiếm theo chiều sâu (Depth First Search - DFS)

Thuật toán Depth First Search (DFS – Tìm kiếm theo chiều sâu) là một dạng thuật toán duyệt hoặc tìm kiếm trên cây hoặc đồ thị. Trong lý thuyết khoa học máy tính, thuật toán DFS nằm trong chiến lược tìm kiếm mù (tìm kiếm không có định hướng, không chú ý đến thông tin, giá trị được duyệt) được ứng dụng để duyệt hoặc tìm kiếm trên đồ thị.

Ý tưởng thuật toán

Từ đỉnh (nút) gốc ban đầu.

- Duyệt đi xa nhất theo từng nhánh.
  - Khi nhánh đã duyệt hết, lùi về từng đỉnh để tìm và duyệt những nhánh tiếp theo.
- Quá trình duyệt chỉ dừng lại khi tìm thấy đỉnh cần tìm hoặc tất cả đỉnh đều đã được duyệt qua.

Thuật giải

Một số quy ước:

- Open: là tập hợp các đỉnh chờ được xét ở bước tiếp theo theo ngăn xếp (ngăn xếp: dãy các phần tử mà khi thêm phần tử vào sẽ thêm vào đầu dãy, còn khi lấy phần tử ra sẽ lấy ở phần tử đứng đầu dãy).
- Close: là tập hợp các đỉnh đã xét, đã duyệt qua.
- s: là đỉnh xuất phát, đỉnh gốc ban đầu trong quá trình tìm kiếm.
- g: đỉnh đích cần tìm.
- p: đỉnh đang xét, đang duyệt.

Trình bày thuật giải:

1. Tập Open chứa đỉnh gốc s chờ được xét.
2. Kiểm tra tập Open có rỗng không.
    - Nếu tập Open không rỗng, lấy một đỉnh ra khỏi tập Open làm đỉnh đang xét p.
        - Nếu p là đỉnh g cần tìm, kết thúc tìm kiếm.
    - Nếu tập Open rỗng, tiến đến bước 4.
3. Đưa đỉnh p vào tập Close, sau đó xác định các đỉnh kề với đỉnh p vừa xét.
    - Nếu các đỉnh kề không thuộc tập Close, đưa chúng vào đầu tập Open. Quay lại bước 2.
4. Kết luận không tìm ra đỉnh đích cần tìm.

Tìm kiếm theo chiều rộng (Breadth First Search - BFS)

Thuật toán Breadth First Search (BFS - Tìm kiếm theo chiều rộng) là thuật toán xét (duyệt) hoặc tìm kiếm trên cây và đồ thị, có chiến lược tìm kiếm mù (tìm kiếm không có định hướng, không chú ý đến thông tin, giá trị được duyệt).

Ý tưởng thuật toán:

Từ một đỉnh (nút) gốc ban đầu.

- Xác định và lần lượt duyệt các đỉnh kề xung quanh đỉnh gốc vừa xét.
    - Tiếp tục quá trình duyệt qua các đỉnh kề đỉnh vừa xét cho đến khi đạt được kết quả cần tìm hoặc duyệt qua tất cả các đỉnh.

Thuật giải

Một số quy ước:

- Open: là tập hợp các đỉnh chờ được xét ở bước tiếp theo theo hàng đợi (hàng đợi: dãy các phần tử mà khi thêm phần tử vào sẽ thêm vào cuối dãy, còn khi lấy phần tử ra sẽ lấy ở phần tử đứng đầu dãy).
- Close: là tập hợp các đỉnh đã xét, đã duyệt qua.
- s: là đỉnh xuất phát, đỉnh gốc ban đầu trong quá trình tìm kiếm.
- g: đỉnh đích cần tìm.
- p: đỉnh đang xét, đang duyệt.

Trình bày thuật giải:

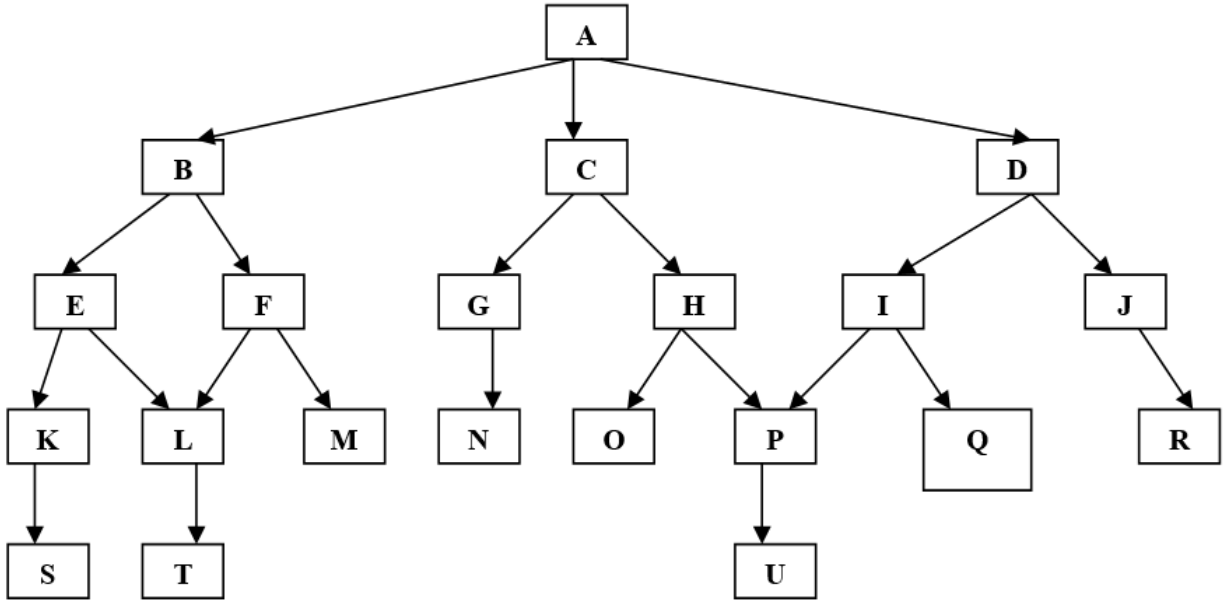
- Bước 1: Tập Open chứa đỉnh gốc *s* chờ được xét.
- Bước 2: Kiểm tra tập Open có rỗng không.
  - Nếu tập Open không rỗng, lấy một đỉnh ra khỏi tập Open làm đỉnh đang xét *p*. Nếu *p* là đỉnh g cần tìm, kết thúc tìm kiếm.
  - Nếu tập Open rỗng, tiến đến bước 4.
- Bước 3: Đưa đỉnh *p* vào tập Close, sau đó xác định các đỉnh kề với đỉnh p vừa xét. Nếu các đỉnh kề không thuộc tập Close, đưa chúng vào cuối tập Open. Quay lại bước 2.
- Bước 4: Kết luận không tìm ra đỉnh đích cần tìm.

Tìm kiếm sâu dần (Iterative Deepening Search - IDS)

Kết hợp của tìm kiếm rộng và tìm kiếm sâu trên cơ sở cho biết mức sâu *n* rồi tìm kiếm rộng ứng mới mức sâu đó

Một số ví dụ:

Ví dụ 2.7: Cho không gian trạng thái của bài toán như sơ đồ dưới đây:



Hình 2.12: Không gian trạng thái của ví dụ 2.7

Giả sử U là trạng thái đích của bài toán. Hãy sử dụng giải thuật tìm kiếm theo chiều rộng và chiều sâu để giải bài toán?

Gọi:

+ Danh sách open chứa các đỉnh đang chờ duyệt qua
+ Danh sách closed chứa các đỉnh đã được duyệt qua

Sử dụng giải thuật tìm kiếm theo chiều rộng để tìm kiếm trạng thái đích U trên đồ thị không gian trạng thái của hình vẽ cho kết quả của các vòng lặp như sau:

1- open = [A]; closed = [ ]

2- open = [B, C, D]; closed = [A]
3- open = [C, D, E, F]; closed = [B, A]
4- open = [D, E, F, G, H]; closed = [C, B, A]
5- open = [E, F, G, H, I, J]; closed = [D, C, B, A]
6- open = [F, G, H, I, J, K, L]; closed = [E, D, C, B, A]
7- open = [G, H, I, J, K, L, M] (L đã có sẵn trên danh sách open);
closed = [F, E, D, C, B, A]
8- open = [H, I, J, K, L, M, N]; closed = [G, F, E, D, C, B, A]
9- tiếp tục cho đến khi U được tìm thấy hoặc open = [ ]

Ví dụ 2.8: Hãy sử dụng giải thuật tìm kiếm theo chiều sâu để tìm kiếm trạng thái đích U trên đồ thị.

Kết quả các vòng lặp như sau:

1- open = [A]; closed = [ ]
2- open = [B, C, D]; closed = [A]
3- open = [E, F, C, D]; closed = [B, A]
4- open = [K, L, F, C, D]; closed = [E, B, A]
5- open = [S, L, F, C, D]; closed = [K, E, B, A]
6- open = [L, F, C, D]; closed = [S, K, E, B, A]
7- open = [T, F, C, D]; closed = [L, S, K, E, B, A]
8- open = [F, C, D]; closed = [T, L, S, K, E, B, A]
9- open = [M, C, D]; closed = [F, T, L, S, K, E, B, A]
10- open = [C, D]; closed = [M, F, T, L, S, K, E, B, A] tiếp tục cho đến khi U được tìm thấy hoặc open = [ ]

Ví dụ 2.9: Bài toán đong nước. Cho 2 bình có dung tích lần lượt là m và n (lit). Với nguồn nước không hạn chế, dùng 2 bình trên để đong k lit nước. Không mất tính tổng quát có thể giả thiết $k \le min(m,n)$.

Tại mỗi thời điểm xác định, lượng nước hiện có trong mỗi bình phản ánh bản chất hình trạng của bài toán ở thời điểm đó.

- Gọi x là lượng nước hiện có trong bình dung tích m và y là lượng nước hiện có trong bình dung tích n. Như vậy bộ có thứ tự (x,y) có thể xem là trạng thái của bài toán. Với cách mô tả như vậy, các trạng thái đặc biệt của bài toán sẽ là:

Trạng thái đầu: (0,0)

Trạng thái cuối: (x,k) hoặc (k,y), $0 \le x \le m$, $0 \le y \le n$

Bài toán đong nước với $m=5$, $n=4$, $k=3$. Giải bài toán theo phương pháp tìm kiếm theo chiều rộng.

Mức 1: Trạng thái đầu (0;0)

Mức 2: Các trạng thái (5;0), (0;4), Mức 3: (5;4), (1;4), (4,0)

Mức 4: (1;0), (4;4) Mức 5: (0;1), (5;3)

Ở mức 5 ta gặp trạng thái đích là (5;3) vì vậy có được lời giải như sau:

(0;0)→ (0;4) → (4;0) → (4;4) → (5;3)

Để có được lời giải này ta phải lưu lại vết của đường đi, có thể trình bày quá trình tìm kiếm dưới dạng bảng sau:

i	T(i)	↑MO ↓	DONG
		(0;0)	
(0;0)	(5;0) **(0;4)**	(5;0) (0;4)	(0;0)
(5;0)	(5;4) (0;0) (1;4)	(0;4) (5;4) (1;4)	(0;0) (5;0)
(0;4)	(5;4) (0;0) **(4;0)**	(5;4) (1;4) (4;0)	(0;0) (5;0) (0;4)
(5;4)	(0;4) (5;0)	(1;4) (4;0)	(0;0) (5;0) (0;4) (5;4)
(1;4)	(5;4) (0;4) (1;0) (5;0)	(4;0) (1;0)	(0;0) (5;0) (0;4) (5;4) (1;4)
(4;0)	(5;0) (4;4) (0;0) **(0;4)**	(1;0) (4;4)	(0;0) (5;0) (0;4) (5;4) (1;4) (4;0)
(1;0)	(5;0) (1;4) (0;1)	(4;4) (0;1)	(0;0) (5;0) (0;4) (5;4) (1;4) (4;0) (1;0)
(4;4)	(5;4) (0;4) (4;0) **(5;3)**	(0;1) (5;3)	(0;0) (5;0) (0;4) (5;4) (1;4) (4;0) (1;0) (4;4)
(0;1)	(5;1) (0;4) (0;0) (1;0)	(5;3) (5;1)	(0;0) (5;0) (0;4) (5;4) (1;4) (4;0) (1;0) (0;1)
(5;3)			

Ví dụ 2.10: Bài toán đong nước với m = 5, n = 4, k = 3. Giải bài toán theo phương pháp tìm kiếm sâu. Nếu ta chọn nhánh ưu tiên đổ đầy bình thứ hai thì sẽ tìm thấy lời giải rất nhanh. Quá trình tìm kiếm có thể trình bày bằng bảng dưới đây:

i	T(i)	MO ↓↑	DONG
		(0;0)	
(0;0)	(5;0) **(0;4)**	(5;0) (0;4)	(0;0)
(0;4)	(5;4) (0;0) **(4;0)**	(5;0) (5;4) (4;0)	(0;0) (0;4)
(4;0)	(5;0) (4;4) (0;0) **(0;4)**	(5;0) (5;4) (4;4)	(0;0) (0;4) (4;0)
(4;4)	(5;4) (0;4) (4;0) **(5;3)**	(5;0) (5;4) (5;3)	(0;0) (0;4) (4;0) (4;4)
(5;3)			

Lời giải tìm được: (0;0) → (0;4) → (4;0) → (4;4) → (5;3)

Ví dụ 2.11: Bài toán trò chơi 8 số. Giải bài toán với phương pháp tìm kiếm rộng

Bảng xuất phát

2	8	3
1	6	4
7		5

Bảng kết thúc

1	2	3
8		4

7	6	5

Mức 1: Có một trạng thái

2	8	3
1	6	4
7		5

Mức 2: Có ba trạng thái

2	8	3
1		4
7	6	5

2	8	3
1	6	4
	7	5

2	8	3
1	6	4
7	5	

Mức 3: Có năm trạng thái

2	8	3
	1	4
7	6	5

2	8	3
1	4	
7	6	5

2		3
1	8	4
7	6	5

2	8	3
	6	4
1	7	5

2	8	3
1	6	
7	5	4

Mức 4: Có mười trạng thái

	8	3
2	1	4
7	6	5

2	8	3
7	1	4
	6	5

2	8	
1	4	3

2	8	3
1	4	5

1	7	5

7	6	

	2	3
1	8	4
7	6	5

2	3	
1	8	4
7	6	5

	8	3
2	6	4
1	7	5

2	8	3
6		4
1	7	5

2	8	
1	6	3
7	5	4

2	8	3
1	6	4
7	5	

Mức 6: Có 12 trạng thái

1	2	3
	8	4
7	6	5

2	3	4
1	8	
7	6	5

8		3
2	1	4
7	6	5

2	8	3
7	1	4
6		5

2		8
1	4	3
7	6	5

2	8	3
1	4	5
7		6

8		3
2	6	4
1	7	5

2		3
6	8	4
1	7	5

2	8	3
6	7	4
1		5

2		8
1	6	3
7	5	4

2		3
1	8	3
7	5	4

2	8	3
1	5	6
7		4

Mức 6: Có 24 trạng thái

1	2	3
8		4
7	6	5

1	2	3
7	8	4
	6	5

. . .

Ở mức này ta gặp được trạng thái đích.

1	2	3
8		4
7	6	5

Ví dụ 2.12: Bài toán tháp Hà Nội

Cho ba cọc 1, 2, 3. Ở cọc 1 ban đầu có n đĩa sắp xếp theo thứ tự to dần từ dưới lên trên. Hãy dịch chuyển n đĩa đó sang cọc thứ 3 sao cho:

Mỗi lần chỉ chuyển một đĩa.

Trong mỗi cọc không cho phép đĩa to nằm trên đĩa nhỏ hơn.

Bài toán xác định khi biết được từng đĩa đang nằm ở cọc nào. Hay nói cách khác, có hai cách xác định:

1- Cọc 1 hiện đang chứa những đĩa nào? Cọc 2 hiện đang chứa những đĩa nào? Và cọc 3 đang chứa những đĩa nào.

2- Đĩa lớn thứ i hiện đang nằm ở cọc nào? (i = 1 … n)

Như vậy cách mô tả trạng thái bài toán không duy nhất, vấn đề là chọn cách mô tả nào để đạt được mục đích dễ dàng nhất.

Theo trên, với cách thứ nhất ta phải dùng 3 danh sách động vì số đĩa trên mỗi cọc là khác nhau trong từng thời điểm khác nhau.

Cách thứ hai, nhìn qua thì khó mô tả nhưng dựa vào khái niệm về bộ có thứ tự trong toán học, cách này mô tả bài toán hiệu quả hơn. Thật vậy, nếu gọi xi là cọc chứa đĩa lớn thứ i, trong đó xi$\in${1, 2, 3}, i$\in${1 ..n}. Khi đó bộ có thứ tự (x1, x2, . . ,xn) có thể dùng làm dạng mô tả trạng thái đang xét của bài toán. Với cách mô tả này,

Trạng thái đầu là (1,1,. . .,1)

Trạng thái cuối là (3,3,. . .,3)

Ví dụ 2.13: Bài toán Tháp Hà nội với $n=3$. Giải bài toán tháp Hà nội với phương pháp tìm kiếm sâu

Nhắc lại, dùng bộ ba (x1; x2; x3) biểu diễn trạng thái bài toán, với x_i là cọc chứa đĩa lớn thứ i.

i	T(i)	OPENED ↓↑	CLOSED
		(1;1;1)	
(1;1;1)	(1;1;2) (1;1;3)	(1;1;2) (1;1;3)	(1;1;1)
(1;1;3)	(1;1;1)(1;1;2) (1;2;3)	(1;1;2)(1;2;3)	(1;1;1)(1;1;3)
(1;2;3)	(1;1;3) (1;2;1) (1;2;2)	(1;1;2)(1;2;1)(1;2;2)	(1;1;1)(1;1;3)(1;2;3)
(1;2;2)	(1;2;3) (1;2;1) (3;2;2)	(1;1;2)(1;2;1)(3;2;2)	(1;1;1)(1;1;3)(1;2;3)(1;2;2)
(3;2;2)	(1;2;2) (3;2;3) (3;2;1)	(1;1;2)(1;2;1)(3;2;1)	(1;1;1)(1;1;3)(1;2;3)(1;2;2) (3;2;2)
(3;2;1)	(3;2;2) (3;2;3) (3;3;1)	(1;1;2)(1;2;1)(3;3;1)	(1;1;1)(1;1;3)(1;2;3)(1;2;2) (3;2;2)

			(3;2;1)
(3;3;1)	(3;2;1) (3;3;2) (3;3;3)	(1;1;2)(1;2;1)(3;3;3)	(1;1;1)(1;1;3)(1;2;3)(1;2;2) (3;2;2) (3;2;1) (3;3;3)
(3;3;3)			

Lời giải của bài toán:

(1;1;1) → (1;1;3) → (1;2;3) → (1;2;2) → (3;2;2) → (3;2;1) → (3;3;1) → (3;3;3)

Qua các ví dụ trên, ta thấy tìm kiếm theo chiều sâu đều cho lời giải tốt và nhanh.

2.1.4. Quy vấn đề về các vấn đề con. Tìm kiếm trên đồ thị VÀ/HOẶC

Phân rã bài toán

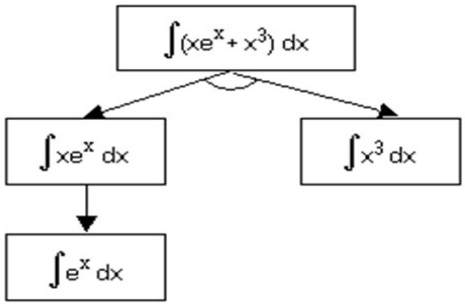
Việc biểu diễn bài toán được thông qua các trạng thái và các toán tử. Khi đó việc tìm lời giải của bài toán được quy về việc tìm đường đi trong không gian trạng thái. Phần này sẽ cung cấp một phương pháp khác để giải quyết vấn đề, đó là dựa trên việc quy vấn đề về các vấn đề con.

Quy vấn đề về các vấn đề con (còn gọi là rút gọn vấn đề) là một phương pháp được sử dụng rộng rãi nhất để giải quyết các vấn đề. Trong đời sống hàng ngày, cũng như trong khoa học kỹ thuật, mỗi khi gặp một vấn đề cần giải quyết, ta vẫn thường cố gắng tìm cách đưa nó về các vấn đề đơn giản hơn. Quá trình rút gọn vấn đề sẽ được tiếp tục cho tới khi ta dẫn tới các vấn đề con có thể giải quyết được dễ dàng.

Ý tưởng chủ yếu là xuất phát từ bài toán ban đầu, tách ra các bài toán con, quá trình này tiếp tục đối với các bài toán con cho đến khi gặp các bài toán sơ cấp (bài toán có lời giải ngay).

Ví dụ 2.14: Tích phân bất định:

Giả sử ta cần tính một tích phân bất định, chẳng hạn $\int (x e^x + x^3) dx$. Quá trình chúng ta vẫn thường làm để tính tích phân bất định là như sau: Sử dụng các quy tắc tính tích phân (quy tắc tính tích phân của một tổng, quy tắc tính tích phân từng phần...), sử dụng các phép biến đổi biến số, các phép biến đổi các hàm để đưa tích phân cần tính về tích phân của các hàm số sơ cấp mà chúng ta đã biết cách tính. Chẳng hạn, đối với tích phân trên, áp dụng quy tắc tích phân của tổng ta đưa về hai tích phân $\int xe^x dx$ và $\int x^3 dx$. áp dụng quy tắc tích phân từng phần ta đưa tích phân $\int xe^x dx$ về tích phân $\int e^x dx$. Quá trình trên có thể biểu diễn bởi đồ thị trong hình vẽ.



Hình 2.13: Tích phân bất định

Các tích phân $\int e^x dx$ và $\int x^3 dx$ là các tích phân cơ bản đã có trong bảng tích phân.

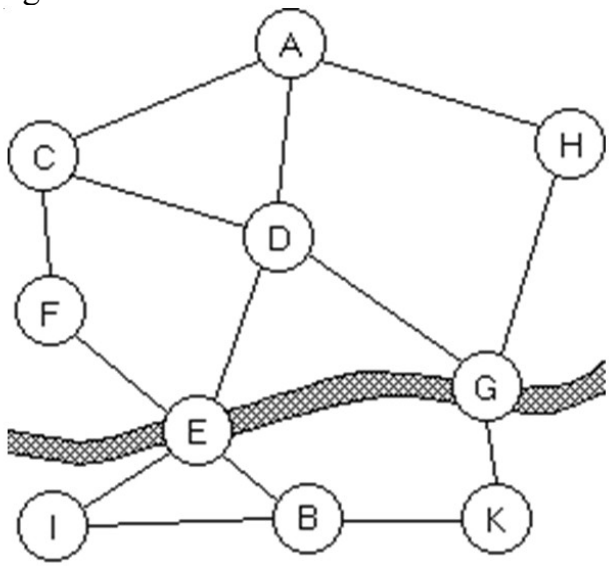
Kết hợp các kết quả của các tích phân cơ bản, ta nhận được kết quả của tích phân đã cho.

Chúng ta có thể biểu diễn việc quy một vấn đề về các vấn đề con cơ bởi các trạng thái và các toán tử. Ở đây, bài toán cần giải là trạng thái ban đầu. Mỗi cách quy bài toán về các bài toán con được biểu diễn bởi một toán tử, toán tử A→B, C biểu diễn việc quy bài toán A về hai bài toán B và C. Chẳng hạn, đối với bài toán tính tích phân bất định, ta có thể xác định các toán tử dạng:

$\int (f1 + f2)\, dx \rightarrow \int f1\, dx, \int f2\, dx$ và $\int u\, dv \rightarrow \int v\, du$

Các trạng thái kết thúc là các bài toán sơ cấp (các bài toán đã biết cách giải). Chẳng hạn, trong bài toán tính tích phân, các tích phân cơ bản là các trạng thái kết thúc. Một điều cần lưu ý là, trong không gian trạng thái biểu diễn việc quy vấn đề về các vấn đề con, các toán tử có thể là đa trị, nó biến đổi một trạng thái thành nhiều trạng thái khác.

Ví dụ 2.15: Tìm đường trên bản đồ



Hình 2.14: Tìm đường trên bản đồ

Bài toán tìm đường đi trong không gian trạng thái, trong đó mỗi trạng thái ứng với một thành phố, mỗi toán tử ứng với một con đường nối, nối thành phố này với thành phố khác. Bây giờ ta đưa ra một cách biểu diễn khác dựa trên việc quy vấn đề về các vấn đề con. Giả sử ta có bản đồ giao thông trong một vùng lãnh thổ, cần tìm đường đi từ thành phố A tới thành phố B. Có con sông chảy qua hai thành phố E và G và có cầu qua sông ở mỗi thành phố đó. Mọi đường đi từ A đến B chỉ có thể qua E hoặc G. Như vậy bài toán tìm đường đi từ A đến B được quy về:

- Bài toán tìm đường đi từ A đến B qua E (hoặc)
- Bài toán tìm đường đi từ A đến B qua G.

Mỗi một trong hai bài toán trên lại có thể phân nhỏ như sau

- Bài toán tìm đường đi từ A đến B qua E được quy về:

+ Tìm đường đi từ A đến E (và)

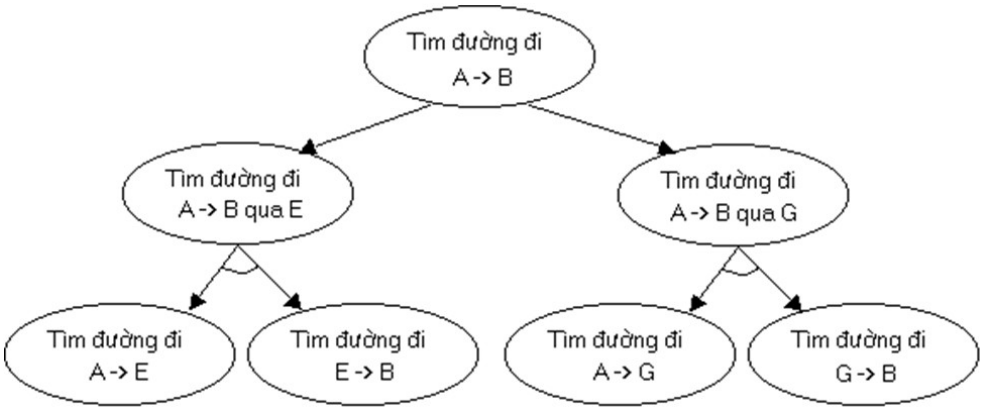
+ Tìm đường đi từ E đến B.

- Bài toán tìm đường đi từ A đến B qua G được quy về:

+ Tìm đường đi từ A đến G (và)

+ Tìm đường đi từ G đến B.

Quá trình rút gọn vấn đề như trên có thể biểu diễn dưới dạng đồ thị (đồ thị và/hoặc) trong hình vẽ. Ở đây mỗi bài toán tìm đường đi từ một thành phố tới một thành phố khác ứng với một trạng thái. Các trạng thái kết thúc là các trạng thái ứng với các bài toán tìm đường đi, chẳng hạn từ A đến C, hoặc từ D đến E, bởi vì đã có đường nối A với C, nối D với E.

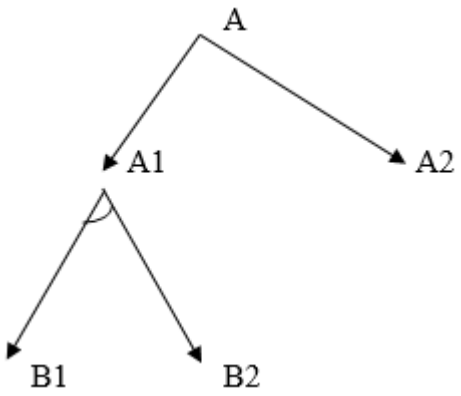


Hình 2.15: Bài toán tìm đường

Đồ thị và/hoặc

Không gian trạng thái mô tả việc quy vấn đề về các vấn đề con có thể biểu diễn dưới dạng đồ thị định hướng đặc biệt gọi là đồ thị và/hoặc. Đồ thị này được xây dựng như sau:

Mỗi bài toán ứng với một đỉnh của đồ thị. Nếu có một toán tử quy bài toán về các bài toán tương đương thì sẽ có các cung đi từ bài toán xuất phát đến các bài toán tương đương đó. Nếu một toán tử quy bài toán về các bài toán con thì cũng có các cung nối từ bài toán xuất phát đến các bài toán con, ngoài ra giữa các cung này cũng có đường nới với nhau. Chẳng hạn, giả sử bài toán A được đưa về hai bài toán tương đương A1 và A2. Bài toán A1 lại được quy về hai bài toán con B1 và B2, ta có biểu diễn như hình vẽ.



Hình 2.16: Phân ra bài toán

Một cách hình thức ta có thể định nghĩa đồ thị và/hoặc như sau:

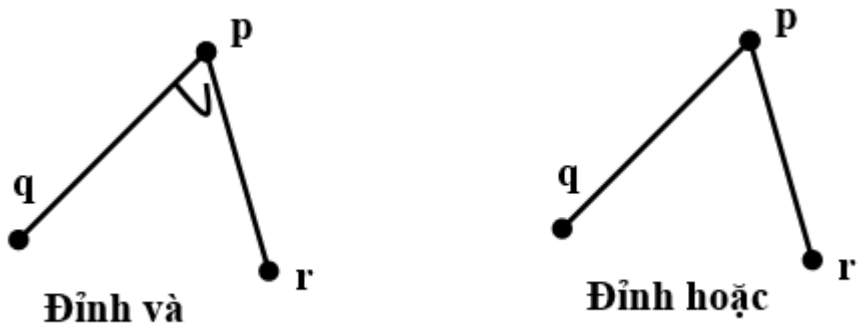
Đồ thị dùng để biểu diễn không gian bài toán mà các luật của nó được thiết kế với dạng p → q, gọi là đồ thị hoặc như ta đã thảo luận ở trên.

Một cấu trúc khác dùng để biểu diễn không gian trạng thái của bài toán mà các luật của nó được thiết kế với các dạng q ¿ r → p và q ¿ r → p, trong đó ¿,∪¿¿ là các toán tử và, hoặc, cấu trúc này được gọi là đồ thị và-hoặc.

Trong đồ thị và-hoặc, luật với dạng q ¿ r → p, giá trị chân lý của p phụ thuộc vào cả hai giá trị chân lý của q và r, trong khi đó luật với dạng q ¿ r → p, giá trị chân lý của p chỉ phụ thuộc vào một trong hai giá trị chân lý q hoặc r. Luật với dạng q ¿ r → p có thể được tách ra thành hai luật đó là q → p và r → p

Nói chung nếu luật được thể hiện dưới dạng if-then mà vế bên trái của luật có nhiều toán tử kết nối và thì các toán tử và này tạo thành một cung, và cung này được gọi là cung và (hay còn gọi là đỉnh và). Mỗi đỉnh và có nhiều đỉnh kế thừa và mỗi kế thừa của nó được gọi là đỉnh thành phần. Hình bên trái ở bên biểu diễn luật với dạng q ¿ r → p, trong đó q, r được gọi là các đỉnh thành phần và p được gọi là đỉnh và (hay còn gọi là cung và).

Nếu luật được thiết kế dưới dạng if-then mà vế bên trái của luật có nhiều toán tử kết nối hoặc, thì các toán tử hoặc này tạo thành một cung và cung này được gọi là cung hoặc (hay còn gọi là đỉnh hoặc). Mỗi đỉnh hoặc có nhiều đỉnh kế thừa và mỗi kế thừa được gọi là đỉnh thành phần. Hình bên phải biểu diễn luật với dạng q ¿ r → p, với q, r là các đỉnh thành phần của cung hoặc p .



Hình 2.17: Đỉnh và và đỉnh hoặc

Ta có sự tương quan như sau:

Phân rã bài toán	Đồ thị VÀ/HOẶC
Bài toán	Đỉnh
Chuyển bài toán thành các bài toán con	Cung
Bài toán sơ cấp	Đỉnh cuối
Các bài toán con phụ	Đỉnh VÀ
Các bài toán con độc lập	Đỉnh HOẶC
Giải bài toán	Tìm đồ thị con lời giải bài toán

Tìm kiếm trên đồ thị và/hoặc

Sau khi lựa chọn mô tả bài toán và các toán tử quy bài toán về bài toán con, ta có thể xây dựng đồ thị Và/hoặc để giải quyết bài toán ban đầu hoặc chứng tỏ tính không giải được của nó. Cũng như đồ thị trong không gian trạng thái, đồ thị và/hoặc có thể cho dưới dạng tường minh hoặc không tường minh trên cơ sở toán tử xây dựng.

Các phương pháp tìm kiếm trên đồ thị và/hoặc khác nhau chủ yếu ở phương pháp lựa chọn và sắp xếp đỉnh trước khi tháo chúng. Tương tự như trong không gian trạng thái, ta cũng có các phương pháp sau:

Tìm kiếm theo chiều rộng.

Tìm kiếm theo chiều sâu.

Các quá trình này khác hẳn với các quá trình lựa chọn trong không gian trạng thái. Sự khác biệt chủ yếu là do việc kiểm tra tính kết thúc của quá trình tìm kiếm và các phương pháp sắp xếp và lựa chọn đỉnh để xét phức tạp hơn nhiều. Thay cho việc tìm kiếm đỉnh thoả mãn điều kện đích, chúng ta phải tiến hành tìm kiếm đồ thị lời giải. Do đó, ở những thời điểm nhất định, ta phải kiểm tra xem đỉnh đầu có giải được hay không, nếu đỉnh đầu giải được thì kết thúc công việc, trong trường hợp ngược lại thì tiếp tục xét các nút. Nếu đỉnh đang xét không phải là đỉnh kết thúc và nó là đỉnh lá, tức là đỉnh không giải được. Lúc này phải kiểm tra đỉnh đàu có phải không giải được hay không, nếu đúng thì dừng, ngược lại, tiếp tục tìm kiếm.

Ta sẽ sử dụng kỹ thuật tìm kiếm theo độ sâu trên đồ thị và/hoặc để đánh dấu các đỉnh. Các đỉnh sẽ được đánh dấu giải được hoặc không giải được theo định nghĩa đệ quy về đỉnh giải được và không giải được. Xuất phát từ đỉnh ứng với bài toán ban đầu, đi xuống theo độ sâu, nếu gặp đỉnh u là đỉnh kết thúc thì nó được đánh dấu giải được. Nếu gặp đỉnh u không phải là đỉnh kết thúc và từ u không đi tiếp được, thì u được đánh dấu không giải được. Khi đi tới đỉnh u, thì từ u ta lần lượt đi xuống các đỉnh v kề u theo một toán tử R nào đó. Nếu đánh dấu được một đỉnh v không giải được thì không cần đi tiếp xuống các đỉnh v còn lại. Tiếp tục đi xuống các đỉnh kề u theo một toán tử khác. Nếu tất cả các đỉnh kề u theo một toán tử nào đó được đánh dấu giải được thì u sẽ được đánh dấu giải được và quay lên cha của u. Còn nếu từ u đi xuống các đỉnh kề nó theo mọi toán tử đều gặp các đỉnh kề được đánh dấu không giải được, thì u được đánh dấu không giải được và quay lên cha của u.

2.2. Các chiến lược tìm kiếm kinh nghiệm

2.2.1. Hàm đánh giá và tìm kiếm kinh nghiệm

Trong nhiều vấn đề, ta có thể sử dụng kinh nghiệm, tri thức của chúng ta về vấn đề để đánh giá các trạng thái của vấn đề. Với mỗi trạng thái u, chúng ta sẽ xác định một giá trị số h(u), số này đánh giá "sự gần đích" của trạng thái u. Hàm $h(u)$ được gọi là hàm đánh giá. Chúng ta sẽ sử dụng hàm đánh giá để hướng dẫn sự tìm kiếm. Trong quá trình tìm kiếm, tại mỗi bước ta sẽ chọn trạng thái để phát triển là trạng thái có giá trị hàm đánh giá nhỏ nhất, trạng thái này được xem là trạng thái có nhiều hứa hẹn nhất hướng tới đích.

Các kỹ thuật tìm kiếm sử dụng hàm đánh giá để hướng dẫn sự tìm kiếm được gọi chung là các kỹ thuật tìm kiếm kinh nghiệm (heuristic search). Các giai đoạn cơ bản để giải quyết vấn đề bằng tìm kiếm kinh nghiệm như sau:

- ✓ Tìm biểu diễn thích hợp mô tả các trạng thái và các toán tử của vấn đề.

- ✓ Xây dựng hàm đánh giá.
- ✓ Thiết kế chiến lược chọn trạng thái để phát triển ở mỗi bước.

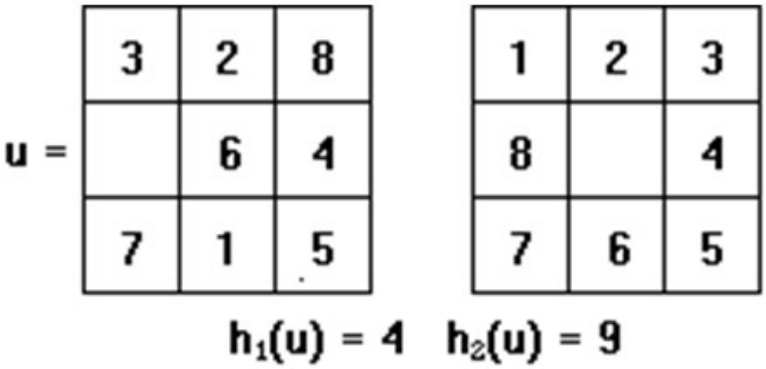
Hàm đánh giá

Trong tìm kiếm kinh nghiệm, hàm đánh giá đóng vai trò cực kỳ quan trọng. Chúng ta có xây dựng được hàm đánh giá cho ta sự đánh giá đúng các trạng thái thì tìm kiếm mới hiệu quả. Nếu hàm đánh giá không chính xác, nó có thể dẫn ta đi chệch hướng và do đó tìm kiếm kém hiệu quả.

Hàm đánh giá được xây dựng tùy thuộc vào vấn đề. Sau đây là một số ví dụ về hàm đánh giá:

- ✓ Trong bài toán tìm kiếm đường đi trên bản đồ giao thông, ta có thể lấy độ dài của đường chim bay từ một thành phố tới một thành phố đích làm giá trị của hàm đánh giá.
- ✓ Bài toán 8 số. Chúng ta có thể đưa ra hai cách xây dựng hàm đánh giá.

Hàm h1: Với mỗi trạng thái u thì $h_1(u)$ là số quân không nằm đúng vị trí của nó trong trạng thái đích. Chẳng hạn trạng thái đích ở bên phải hình, và u là trạng thái ở bên trái hình, thì $h_1(u)=4$, vì các quân không đúng vị trí là 3, 8, 6 và 1.



Hình 2.18: Đánh giá trạng thái u

Hàm h2: h2(u) là tổng khoảng cách giữa vị trí của các quân trong trạng thái u và vị trí của nó trong trạng thái đích. ở đây khoảng cách được hiểu là số ít nhất các dịch chuyển theo hàng hoặc cột để đưa một quân tới vị trí của nó trong trạng thái đích. Chẳng hạn với trạng thái u và trạng thái đích như trong hình 2.1, ta có: $h_2(u) = 2 + 3 + 1 + 3 = 9$

Vì quân 3 cần ít nhất 2 dịch chuyển, quân 8 cần ít nhất 3 dịch chuyển, quân 6 cần ít nhất 1 dịch chuyển và quân 1 cần ít nhất 3 dịch chuyển.

Hai chiến lược tìm kiếm kinh nghiệm quan trọng nhất là tìm kiếm tốt nhất - đầu tiên (best-first search) và tìm kiếm leo đồi (hill-climbing search). Có thể xác định các chiến lược này như sau:

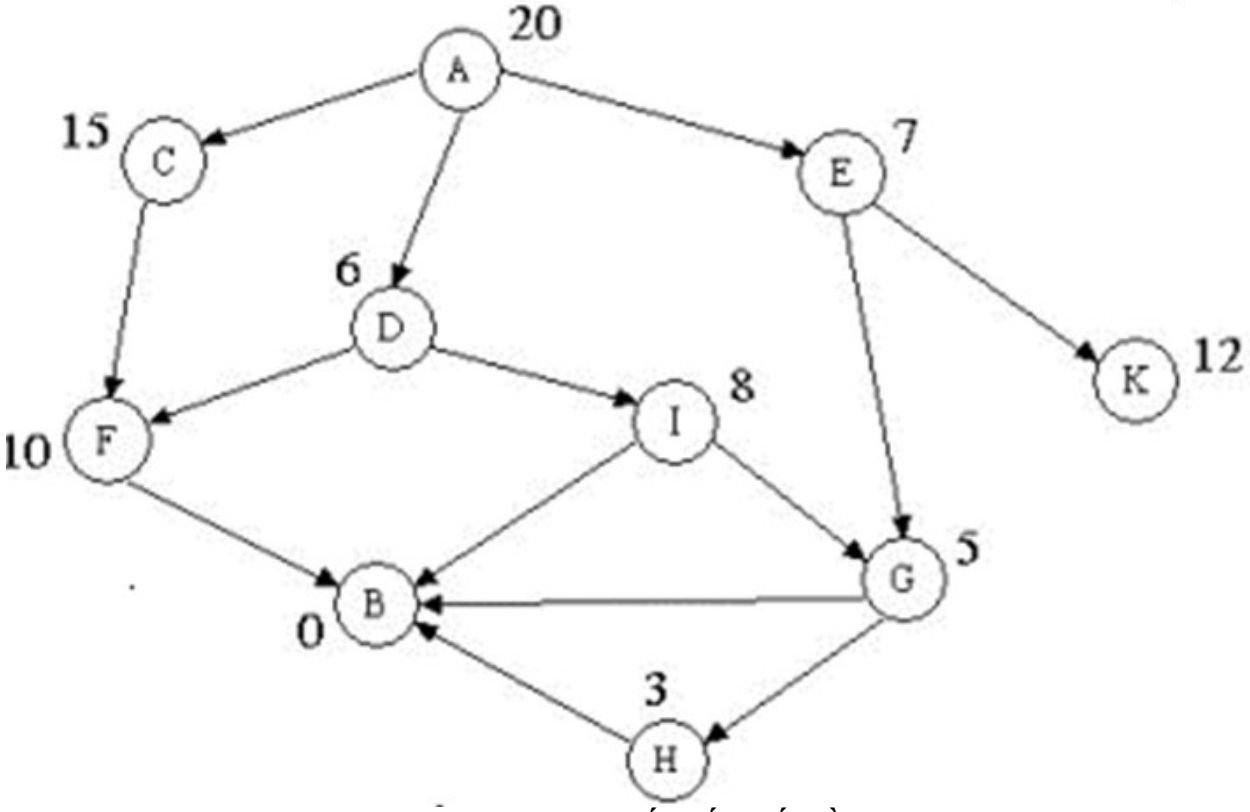
Tìm kiếm tốt nhất đầu tiên = Tìm kiếm theo bề rộng + Hàm đánh giá
Tìm kiếm leo đồi = Tìm kiếm theo độ sâu + Hàm đánh giá

Chúng ta sẽ lần lượt nghiên cứu các kỹ thuật tìm kiếm này trong các mục sau.

2.2.2. Tìm kiếm tốt nhất - đầu tiên

Tìm kiếm tốt nhất - đầu tiên (best-first search) là tìm kiếm theo bề rộng được hướng dẫn bởi hàm đánh giá. Nhưng nó khác với tìm kiếm theo bề rộng ở chỗ, trong tìm

kiếm theo bề rộng ta lần lượt phát triển tất cả các đỉnh ở mức hiện tại để sinh ra các đỉnh ở mức tiếp theo, còn trong tìm kiếm tốt nhất - đầu tiên ta chọn đỉnh để phát triển là đỉnh tốt nhất được xác định bởi hàm đánh giá (tức là đỉnh có giá trị hàm đánh giá là nhỏ nhất), đỉnh này có thể ở mức hiện tại hoặc ở các mức trên.



Hình 2.19: Cây tìm kiếm tốt nhất đầu tiên

Ví dụ 2.16: Xét không gian trạng thái được biểu diễn bởi đồ thị trong hình 2.2, trong đó trạng thái ban đầu là A, trạng thái kết thúc là B. Giá trị của hàm đánh giá là các số ghi cạnh mỗi đỉnh. Quá trình tìm kiếm tốt nhất - đầu tiên diễn ra như sau: Đầu tiên phát triển đỉnh A sinh ra các đỉnh kề là C, D và E. Trong ba đỉnh này, đỉnh D có giá trị hàm đánh giá nhỏ nhất, nó được chọn để phát triển và sinh ra F, I. Trong số các đỉnh chưa được phát triển C, E, F, I thì đỉnh E có giá trị đánh giá nhỏ nhất, nó được chọn để phát triển và sinh ra các đỉnh G, K. Trong số các đỉnh chưa được phát triển thì G tốt nhất, phát triển G sinh ra B, H. Đến đây ta đã đạt tới trạng thái kết thúc. Cây tìm kiếm tốt nhất - đầu tiên được biểu diễn trong hình 2.3.

Sau đây là thủ tục tìm kiếm tốt nhất - đầu tiên. Trong thủ tục này, chúng ta sử dụng danh sách L để lưu các trạng thái chờ phát triển, danh sách được sắp theo thứ tự tăng dần của hàm đánh giá sao cho trạng thái có giá trị hàm đánh giá nhỏ nhất ở đầu danh sách.

procedure *Best_First_Search*;

begin

1. Khởi tạo danh sách L chỉ chứa trạng thái ban đầu;

2. **loop do**

2.1 **if** *L rỗng* **then**

{*thông báo thất bại*; **stop**};

2.2 Loại trạng thái u ở đầu danh sách L;

2.3 **if** *u là trạng thái kết thúc* **then**

{*thông báo thành công*; **stop**}

2.4 **for** *mỗi trạng thái v kề u* **do**

Xen v vào danh sách L sao cho L được sắp theo thứ tự tăng dần của hàm đánh giá;

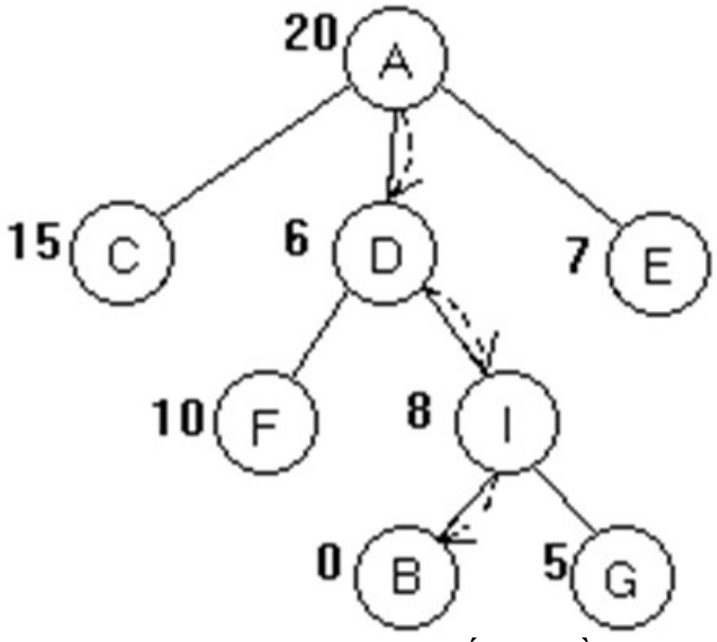
end;

2.2.3. Tìm kiếm leo đồi

Tìm kiếm leo đồi (hill-climbing search) là tìm kiếm theo chiều sâu được hướng dẫn bởi hàm đánh giá. Song khác với tìm kiếm theo chiều sâu, khi ta phát triển một đỉnh u thì bước tiếp theo, ta chọn trong số các đỉnh con của u, đỉnh có nhiều hứa hẹn nhất để phát triển, đỉnh này được xác định bởi hàm đánh giá.

Ví dụ 2.17: Ta lại xét đồ thị không gian trạng thái trong hình 2.2. Quá trình tìm kiếm leo đồi được tiến hành như sau. Đầu tiên phát triển đỉnh A sinh ra các đỉnh con C, D, E. Trong các đỉnh này chọn D để phát triển, và nó sinh ra các đỉnh con B, G. Quá trình tìm kiếm kết thúc. Cây tìm kiếm leo đồi được cho trong hình 2.4.

Trong thủ tục tìm kiếm leo đồi được trình bày dưới đây, ngoài danh sách L lưu các trạng thái chờ được phát triển, chúng ta sử dụng danh sách L1 để lưu giữ tạm thời các trạng thái kề trạng thái u, khi ta phát triển u. Danh sách L1 được sắp xếp theo thứ tự tăng dần của hàm đánh giá, rồi được chuyển vào danh sách L sao trạng thái tốt nhất kề u đứng ở danh sách L.



Hình 2.20: Cây tìm kiếm leo đồi

procedure *Hill_Climbing_Search*;

begin

1. Khởi tạo danh sách L chỉ chứa trạng thái ban đầu;

2. **loop do**

2.1 **if** *L rỗng* **then**

{*thông báo thất bại*; **stop**};

2.2 Loại trạng thái u ở đầu danh sách L;

2.3 **if** *u là trạng thái kết thúc* **then**

{*thông báo thành công*; **stop**};

2.3 **for** *mỗi trạng thái v kề u do đặt v vào* L_1;

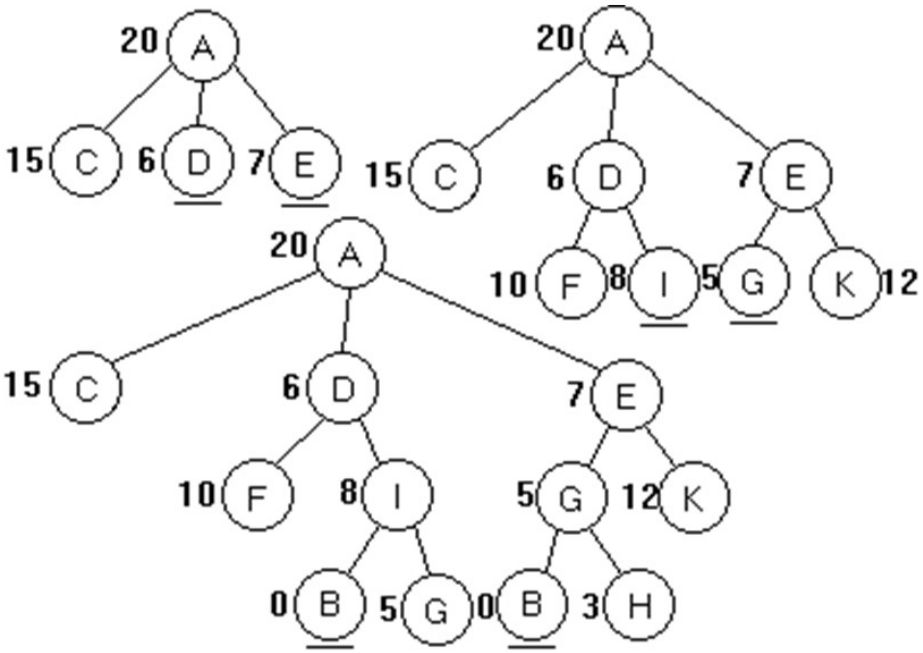
2.5 Sắp xếp L_1 *theo thứ tự tăng dần của hàm đánh giá*;

2.6 Chuyển danh sách L_1 *vào đầu danh sách L*;

end;

2.2.4. Tìm kiếm beam

Tìm kiếm beam (beam search) giống như tìm kiếm theo chiều rộng, nó phát triển các đỉnh ở một mức rồi phát triển các đỉnh ở mức tiếp theo. Tuy nhiên, trong tìm kiếm theo bề rộng, ta phát triển tất cả các đỉnh ở một mức, còn trong tìm kiếm beam, ta hạn chế chỉ phát triển k đỉnh tốt nhất (các đỉnh này được xác định bởi hàm đánh giá). Do đó trong tìm kiếm beam, ở bất kỳ mức nào cũng chỉ có nhiều nhất k đỉnh được phát triển, trong khi tìm kiếm theo bề rộng, số đỉnh cần phát triển ở mức d là bd (b là nhân tố nhánh).



Hình 2.21: Cây tìm kiếm Beam

Ví dụ: Chúng ta lại xét đồ thị không gian trạng thái trong hình 2.2. Chọn k = 2. Khi đó cây tìm kiếm beam được cho như hình 2.5. Các đỉnh được gạch dưới là các đỉnh được chọn để phát triển ở mỗi mức.

2.3. CÁC CHIẾN LƯỢC TÌM KIẾM TỐI ƯU

2.3.1. Tìm đường đi ngắn nhất

Chúng ta đã nghiên cứu vấn đề tìm kiếm đường đi từ trạng thái ban đầu tới trạng thái kết thúc trong không gian trạng thái. Trong mục này, ta giả sử rằng, giá trị phải trả để đưa trạng thái a tới trạng thái b (bởi một toán tử nào đó) là một số $k(a,b) \geq 0$, ta sẽ gọi số này là độ dài cung (a,b) hoặc giá trị của cung (a,b) trong đồ thị không gian trạng thái. Độ dài của các cung được xác định tùy thuộc vào vấn đề. Chẳng hạn, trong bài toán tìm

đường đi trong bản đồ giao thông, giá của cung (a,b) chính là độ dài của đường nối thành phố a với thành phố b. Độ dài đường đi được xác định là tổng độ dài của các cung trên đường đi. Vấn đề của chúng ta trong mục này, tìm đường đi ngắn nhất từ trạng thái ban đầu tới trạng thái đích. Không gian tìm kiếm ở đây bao gồm tất cả các đường đi từ trạng thái ban đầu tới trạng thái kết thúc, hàm mục tiêu được xác định ở đây là độ dài của đường đi.

Chúng ta có thể giải quyết vấn đề đặt ra bằng cách tìm tất cả các đường đi có thể có từ trạng thái ban đầu tới trạng thái đích (chẳng hạn, sử sụng các ký thuật tìm kiếm mù), sau đó so sánh độ dài của chúng, ta sẽ tìm ra đường đi ngắn nhất. Thủ tục tìm kiếm này thường được gọi là thủ tục bảo tàng Anh Quốc (British Museum Procedure). Trong thực tế, kỹ thuật này không thể áp dụng được, vì cây tìm kiếm thường rất lớn, việc tìm ra tất cả các đường đi có thể có đòi hỏi rất nhiều thời gian. Do đó chỉ có một cách tăng hiệu quả tìm kiếm là sử dụng các hàm đánh giá đề hướng dẫn sử tìm kiếm. Các phương pháp tìm kiếm đường đi ngắn nhất mà chúng ta sẽ trình bày đều là các phương pháp tìm kiếm heuristic.

Giả sử u là một trạng thái đạt tới (có dường đi từ trạng thái ban đầu u0 tới u). Ta xác định hai hàm đánh giá sau:

- ✓ g(u) là đánh giá độ dài đường đi ngắn nhất từ u_0 tới u (Đường đi từ u_0 tới trạng thái u không phải là trạng thái đích được gọi là đường đi một phần, để phân biệt với đường đi đầy đủ, là đường đi từ u_0 tới trạng thái đích).
- ✓ h(u) là đánh giá độ dài đường đi ngắn nhất từ u tới trạng thái đích.

Hàm h(u) được gọi là chấp nhận được (hoặc đánh giá thấp) nếu với mọi trạng thái u, h(u) ≤ độ dài đường đi ngắn nhất thực tế từ u tới trạng thái đích. Chẳng hạn trong bài toán tìm đường đi ngắn nhất trên bản đồ giao thông, ta có thể xác định h(u) là độ dài đường chim bay từ u tới đích.

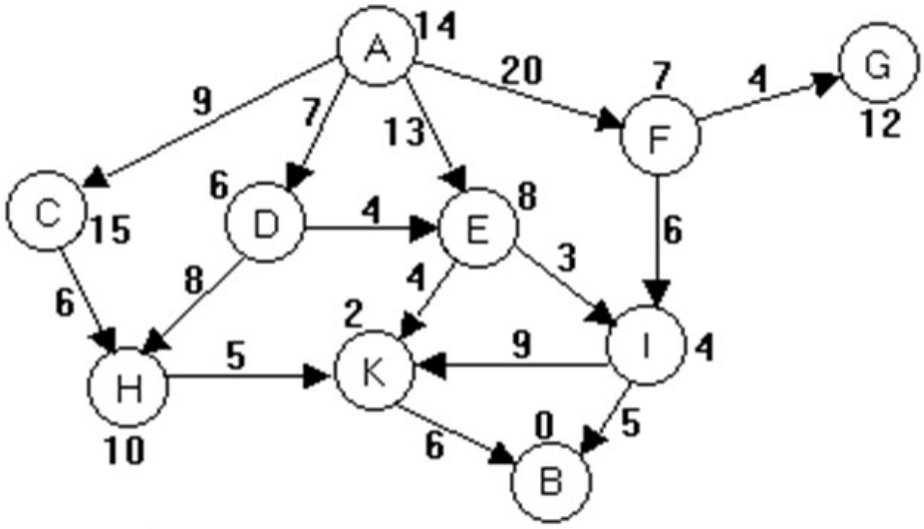
Ta có thể sử dụng kỹ thuật tìm kiếm leo đồi với hàm đánh giá h(u). Tất nhiên phương pháp này chỉ cho phép ta tìm được đường đi tương đối tốt, chưa chắc đã là đường đi tối ưu.

Ta cũng có thể sử dụng kỹ thuật tìm kiếm tốt nhất đầu tiên với hàm đánh giá g(u). Phương pháp này sẽ tìm ra đường đi ngắn nhất, tuy nhiên nó có thể kém hiệu quả.

Để tăng hiệu quả tìm kiếm, ta sử dụng hàm đánh giá mới:

f(u) = g(u) + h(u)

Tức là, f(u) là đánh giá độ dài đường đi ngắn nhất qua u từ trạng thái ban đầu tới trạng thái kết thúc.



Hình 2.22: Đồ thị không gian trạng thái với hàm đánh giá

Thuật toán A*

Thuật toán A* là thuật toán sử dụng kỹ thuật tìm kiếm tốt nhất đầu tiên với hàm đánh giá f(u).

Để thấy được thuật toán A* làm việc như thế nào, ta xét đồ thị không gian trạng thái trong hình 2.23. Trong đó, trạng thái ban đầu là trạng thái A, trạng thái đích là B, các số ghi cạnh các cung là độ dài đường đi, các số cạnh các đỉnh là giá trị của hàm h. Đầu tiên, phát triển đỉnh A sinh ra các đỉnh con C, D, E và F. Tính giá trị của hàm f tại các đỉnh này ta có:

g(C) =9, f(C) = 9 + 15 = 24, g(D) = 7, f(D) = 7 + 6 = 13,
g(E) = 13, f(E) = 13 + 8 = 21, g(F) = 20, f(F) = 20 +7 = 27

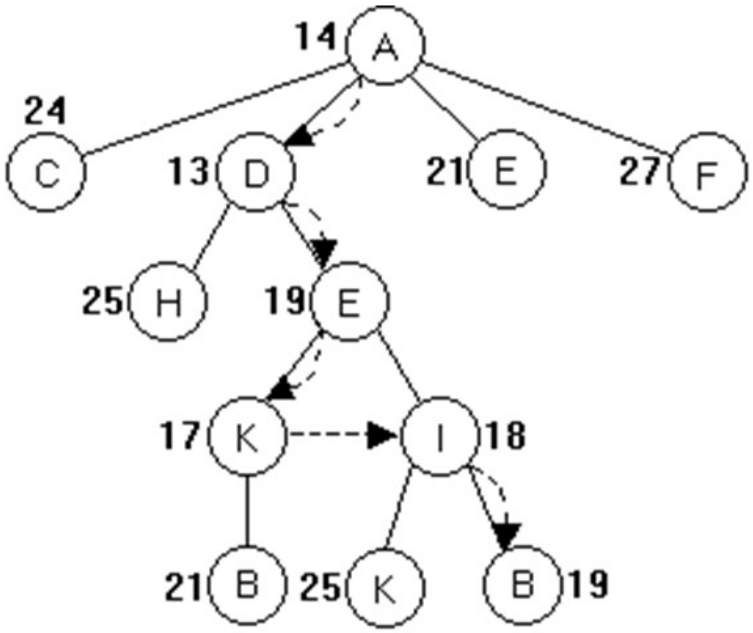
Như vậy đỉnh tốt nhất là D (vì f(D) = 13 là nhỏ nhất). Phát triển D, ta nhận được các đỉnh con H và E. Ta đánh giá H và E (mới):

g(H) = g(D) + Độ dài cung (D, H) = 7 + 8 = 15, f(H) = 15 + 10 = 25.

Đường đi tới E qua D có độ dài:

g(E) = g(D) + Độ dài cung (D, E) = 7 + 4 = 11.

Vậy đỉnh E mới có đánh giá là f(E) = g(E) + h(E) = 11 + 8 = 19. Trong số các đỉnh cho phát triển, thì đỉnh E với đánh giá f(E) = 19 là đỉnh tốt nhất. Phát triển đỉnh này, ta nhận được các đỉnh con của nó là K và I. Chúng ta tiếp tục quá trình trên cho tới khi đỉnh được chọn để phát triển là đỉnh kết thúc B, độ dài đường đi ngắn nhất tới B là g(B) = 19. Quá trình tìm kiếm trên được mô tả bởi cây tìm kiếm trong hình 3.2, trong đó các số cạnh các đỉnh là các giá trị của hàm đánh giá f(u).



Hình 2.23: Cây tìm kiếm theo thuật toán A*

procedure *A**;

begin

1. Khởi tạo danh sách L chỉ chứa trạng thái ban đầu;

2. **loop do**

2.1 **if** *L rỗng* **then**

{*thông báo thất bại*; **stop**};

2.2 Loại trạng thái u ở đầu danh sách L;

2.3 **if** *u là trạng thái đích* **then**

{*thông báo thành công*; **stop**}

2.4 **for** *mỗi trạng thái v kề u* **do**

{$g(v) \leftarrow g(u) + k(u,v)$;

$f(v) \leftarrow g(v) + h(v)$;

Đặt v vào danh sách L;}

2.5 Sắp xếp L theo thứ tự tăng dần của hàm f sao cho

trạng thái có giá trị của hàm f nhỏ nhất

ở đầu danh sách;

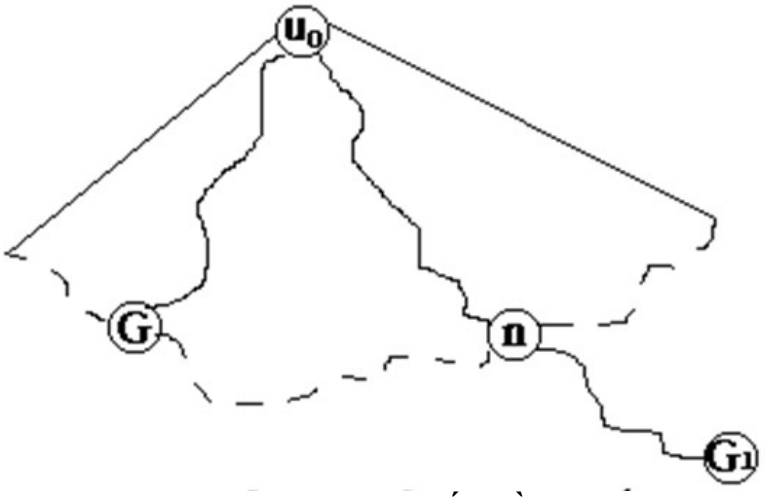
end;

Chúng ta đưa ra một số nhận xét về thuật toán A*.

Người ta chứng minh được rằng, nếu hàm đánh giá h(u) là đánh giá thấp nhất (trường hợp đặc biệt, h(u) = 0 với mọi trạng thái u) thì thuật toán A* là thuật toán *tối ưu*, tức là nghiệm mà nó tìm ra là nghiệm tối ưu. Ngoài ra, nếu độ dài của các cung không nhỏ hơn một số dương δ nào đó thì thuật toán A* là thuật toán *đầy đủ* theo nghĩa rằng, nó luôn dừng và tìm ra nghiệm.

Chúng ta chứng minh tính tối ưu của thuật toán A*.

Giả sử thuật toán dừng lại ở đỉnh kết thúc G với độ dài đường đi từ trạng thái ban đầu u0 tới G là g(G). Vì G là đỉnh kết thúc, ta có h(G) = 0 và f(G) = g(G) + h(G) = g(G). Giả sử nghiệm tối ưu là đường đi từ u0 tới đỉnh kết thúc G1 với độ dài l. Giả sử đường đi này "thoát ra" khỏi cây tìm kiếm tại đỉnh lá n (Xem hình 3.3). Có thể xẩy ra hai khả năng:



Hình 2.24: Đỉnh lá *n* của cây tìm kiếm nằm trên đường đi tối ưu

Giá trị của n trùng với G1 hoặc không. Nếu n là G1 thì vì G được chọn để phát triển trước G1, nên f(G) ≤ f(G1), do đó g(G) ≤ g(G1) = l. Nếu n ≠ G1 thì do h(u) là hàm đánh giá thấp, nên f(n) = g(n) + h(n) ≤ l. Mặt khác, cũng do G được chọn để phát triển trước n, nên f(G) ≤ f(n), do đó, g(G) ≤ l. Như vậy, ta đã chứng minh được rằng độ dài của đường đi mà thuật toán tìm ra g(G) không dài hơn độ dài l của đường đi tối ưu. Vậy nó là độ dài đường đi tối ưu.

Trong trường hợp hàm đánh giá h(u) = 0 với mọi u, thuật toán A* chính là thuật toán tìm kiếm tốt nhất đầu tiên với hàm đánh giá g(u) mà ta đã nói đến.

Thuật toán A* đã được chứng tỏ là thuật toán hiệu quả nhất trong số các thuật toán đầy đủ và tối ưu cho vấn đề tìm kiếm đường đi ngắn nhất.

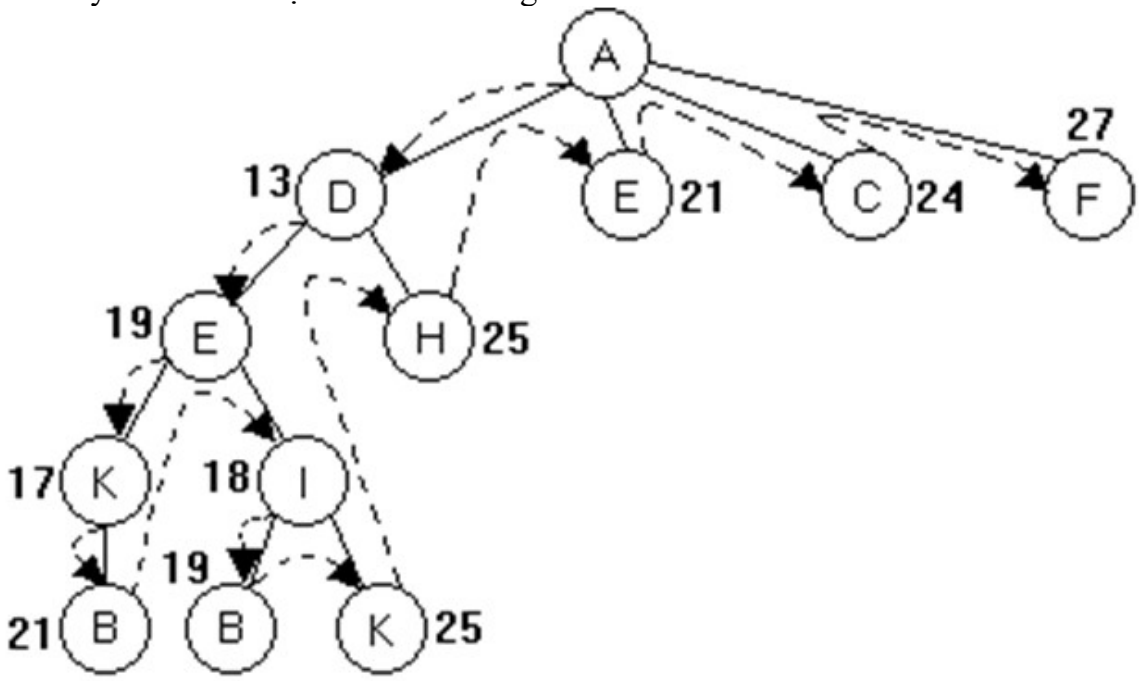
Thuật toán tìm kiếm nhánh-và-cận.

Thuật toán nhánh_và_cận là thuật toán sử dụng tìm kiếm leo đồi với hàm đánh giá f(u).

Trong thuật toán này, tại mỗi bước khi phát triển trạng thái u, thì ta sẽ chọn trạng thái tốt nhất v (f(v) nhỏ nhất) trong số các trạng thái kề u đề phát triển ở bước sau. Đi xuống cho tới khi gặp trạng thái v là đích, hoặc gặp trạng thái v không có đỉnh kề, hoặc gặp trạng thái v mà f(v) lớn hơn độ dài đường đi tối ưu tạm thời, tức là đường đi đầy đủ ngắn nhất trong số các đường đi đầy đủ mà ta đã tìm ra. Trong các trường hợp này, ta không phát triển đỉnh v nữa, hay nói cách khác, ta cắt đi các nhánh cây xuất phát từ v, và quay lên cha của v đề tiếp tục đi xuống trạng thái tốt nhất trong các trạng thái còn lại chưa được phát triển.

Ví dụ 2.18: Chúng ta lại xét không gian trạng thái trong hình 3.1. Phát triển đỉnh A, ta nhận được các đỉnh con C, D, E và F, f(C) = 24, f(D) = 13, f(E) = 21, f(F) = 27. Trong số này D là tốt nhất, phát triển D, sinh ra các đỉnh con H và E, f(H) = 25, f(E) = 19. Đi xuống phát triển E, sinh ra các đỉnh con là K và I, f(K) = 17, f(I) = 18. Đi xuống phát triển K sinh ra đỉnh B với f(B) = g(B) = 21. Đi xuống B, vì B là đỉnh đích, vậy ta tìm được đường đi tối ưu tạm thời với độ dài 21. Từ B quay lên K, rồi từ K quay lên cha nó là E.

Từ E đi xuống J, f(J) = 18 nhỏ hơn độ dài đường đi tạm thời (là 21). Phát triển I sinh ra các con K và B, f(K) = 25, f(B) = g(B) = 19. Đi xuống đỉnh B, vì đỉnh B là đích ta tìm được đường đi đầy đủ mới với độ dài là 19 nhỏ hơn độ dài đường đi tối ưu tạm thời cũ (21). Vậy độ dài đường đi tối ưu tạm thời bây giờ là 19. Bây giờ từ B ta lại quay lên các đỉnh còn lại chưa được phát triển. Song các đỉnh này đều có giá trị hàm đánh giá lớn hơn 19, do đó không có đỉnh nào được phát triển nữa. Như vậy, ta tìm được đường đi tối ưu với độ dài 19. Cây tìm kiếm được biểu diễn trong hình 3.4.



Hình 2.25: Cây tìm kiếm nhánh và cận

Thuật toán nhánh_và_cận sẽ được biểu diễn bởi thủ tục Branch_and_Bound. Trong thủ tục này, biến cost được dùng để lưu độ dài đường đi ngắn nhất. Giá trị ban đầu của cost là số đủ lớn, hoặc độ dài của một đường đi đầy đủ mà ta đã biết.

procedure *Branch_and_Bound*;

begin

1. Khởi tạo danh sách L chỉ chứa trạng thái ban đầu;

Gán giá trị ban đầu cho cost;

2. **loop do**

2.1 **if** *L rỗng* **then stop;**

2.2 Loại trạng thái u ở đầu danh sách L;

2.3 **if** *u là trạng thái kết thúc* **then**

if $g(u) \le y$ **then** {$y \leftarrow g(y)$; *Quay lại 2.1*};

2.4 **if** $f(u) > y$ **then** *Quay lại 2.1*;

2.5 **for** *mỗi trạng thái v kề u* **do**

{$g(v) \leftarrow g(u) + k(u,v)$;

$f(v) \leftarrow g(v) + h(v)$;

Đặt v vào danh sách L_1};

2.6 Sắp xếp L_1 *theo thứ tự tăng của hàm f*;

2.7 Chuyển L_1 vào đầu danh sách L sao cho trạng thái
ở đầu L_1 trở thành ở đầu L;

end;

Người ta chứng minh được rằng, thuật toán nhánh_và_cận cũng là thuật toán đầy đủ và tối ưu nếu hàm đánh giá h(u) là đánh giá thấp và có độ dài các cung không nhỏ hơn một số dương δ nào đó.

2.3.2. Tìm đối tượng tốt nhất

Trên không gian tìm kiếm U được xác định hàm giá (hàm mục tiêu) cost, ứng với mỗi đối tượng x ∈ U với một giá trị số cost(x), số này được gọi là giá trị của x. Chúng ta cần tìm một đối tượng mà tại đó hàm giá trị lớn nhất, ta gọi đối tượng đó là đối tượng tốt nhất. Giả sử không gian tìm kiếm có cấu trúc cho phép ta xác định được khái niệm lân cận của mỗi đối tượng. Chẳng hạn, U là không gian trạng thái thì lân cận của trạng thái u gồm tất cả các trạng thái v kề u; nếu U là không gian các vectơ thực n-chiều thì lân cận của vectơ x = (x1, x2, ... xn) gồm tất cả các vectơ ở gần x theo khoảng cách Ơcơlit thông thường.

Trong mục này, ta sẽ xét kỹ thuật tìm kiếm leo đồi để tìm đối tượng tốt nhất. Sau đó ta sẽ xét kỹ thuật tìm kiếm gradient (gradient search). Đó là kỹ thuật leo đồi áp dụng cho không gian tìm kiếm là không gian các vectơ thực n-chiều và hàm giá là là hàm khả vi liên tục. Cuối cùng ta sẽ nghiên cứu kỹ thuật tìm kiếm mô phỏng luyện kim (simulated annealing).

Tìm đối tượng tốt nhất bằng kỹ thuật tìm kiếm leo đồi

Kỹ thuật tìm kiếm leo đồi để tìm kiếm đối tượng tốt nhất hoàn toàn giống như kỹ thuật tìm kiếm leo đồi để tìm trạng thái kết thúc đã xét trong mục 2.3. Chỉ khác là trong thuật toán leo đồi, từ một trạng thái ta "leo lên" trạng thái kề tốt nhất (được xác định bởi hàm giá), tiếp tục cho tới khi đạt tới trạng thái đích; nếu chưa đạt tới trạng thái đích mà không leo lên được nữa, thì ta tiếp tục "tụt xuống" trạng thái trước nó, rồi lại leo lên trạng thái tốt nhất còn lại. Còn ở đây, từ một đỉnh u ta chỉ leo lên đỉnh tốt nhất v (được xác định bởi hàm giá cost) trong lân cận u nếu đỉnh này "cao hơn" đỉnh u, tức là cost(v) > cost(u). Quá trình tìm kiếm sẽ dừng lại ngay khi ta không leo lên đỉnh cao hơn được nữa.

Trong thủ tục leo đồi dưới đây, biến u lưu đỉnh hiện thời, biến v lưu đỉnh tốt nhất (cost(v) nhỏ nhất) trong các đỉnh ở lân cận u. Khi thuật toán dừng, biến u sẽ lưu trong đối tượng tìm được.

procedure *Hill_Climbing*;
begin
1. u ← một đối tượng ban đầu nào đó;
2. **if** *cost(v) > cost(u)* **then** *u ← v* **else stop;**
end;

Rõ ràng là, khi thuật toán leo đồi dừng lại tại đối tương u*, thì giá của nó cost(u*) lớn hơn giá của tất cả các đối tượng nằm trong lân cận của tất cả các đối tượng trên đường đi từ đối tượng ban đầu tới trạng thái u*. Do đó nghiệm u* mà thuật toán leo đồi tìm được là tối ưu địa phương. Cần nhấn mạnh rằng không có gì đảm bảo nghiệm đó là tối ưu toàn cục theo nghĩa là cost(u*) là lớn nhất trên toàn bộ không gian tìm kiếm.

Để nhận được nghiệm tốt hơn bằng thuật toán leo đồi, ta có thể áp dụng lặp lại nhiều lần thủ tục leo đồi xuất phát từ một dãy các đối tượng ban đầu được chọn ngẫu nhiên và lưu lại nghiệm tốt nhất qua mỗi lần lặp. Nếu số lần lặp đủ lớn thì ta có thể tìm được nghiệm tối ưu.

Kết quả của thuật toán leo đồi phụ thuộc rất nhiều vào hình dáng của "mặt cong" của hàm giá. Nếu mặt cong chỉ có một số ít cực đại địa phương, thì kỹ thuật leo đồi sẽ tìm ra rất nhanh cực đại toàn cục. Song có những vấn đề mà mặt cong của hàm giá tựa như lông nhím vậy, khi đó sử dụng kỹ thuật leo đồi đòi hỏi rất nhiều thời gian.

Tìm kiếm đối tượng tốt nhất bằng kỹ thuật gradient

Tìm kiếm gradient là kỹ thuật tìm kiếm leo đồi để tìm giá trị lớn nhất (hoặc nhỏ nhất) của hàm khả vi liên tục f(x) trong không gian các vectơ thực n-chiều. Như ta đã biết, trong lân cận đủ nhỏ của điểm x = (x1,...,xn), thì hàm f tăng nhanh nhất theo hướng của vectơ gradient:

Do đó tư tưởng của tìm kiếm gradient là từ một điểm ta đi tới điểm ở lân cận nó theo hướng của vectơ gradient.

$$\nabla f = \left(\frac{\partial f}{\partial x1}, \frac{\partial f}{\partial x2}, \ldots, \frac{\partial f}{\partial xn} \right)$$

procedure *Gradient_Search*;
begin
    $x \leftarrow$ *điểm xuất phát nào đó*;
    repeat
        $x \leftarrow x + \alpha \nabla f(x)$;
    until $|\nabla f| < \varepsilon$;
end;

Trong thủ tục trên, α là hằng số dương nhỏ nhất xác định tỉ lệ của các bước, còn ε là hằng số dương nhỏ xác định tiêu chuẩn dừng. Bằng cách lấy các bước đủ nhỏ theo hướng của vectơ gradient chúng ta sẽ tìm được điểm cực đại địa phương, đó là điểm mà tại đó ∇f = 0, hoặc tìm được điểm rất gần với cực đại địa phương.

Tìm kiếm đối tượng tốt nhất bằng kỹ thuật mô phỏng luyện kim

Như đã nhấn mạnh ở trên, tìm kiếm leo đồi không đảm bảo cho ta tìm được nghiệm tối ưu toàn cục. Để cho nghiệm tìm được gần với tối ưu toàn cục, ta áp dụng kỹ thuật leo đồi lặp xuất phát từ các điểm được lựa chọn ngẫu nhiên. Bây giờ thay cho việc luôn luôn "leo lên đồi" xuất phát từ các điểm khác nhau, ta thực hiện một số bước "tụt xuống" nhằm thoát ra khỏi các điểm cực đại địa phương. Đó chính là tư tưởng của kỹ thuật tìm kiếm mô phỏng luyện kim.

Trong tìm kiếm leo đồi, khi ở một trạng thái u ta luôn luôn đi tới trạng thái tốt nhất trong lân cận nó. Còn bây giờ, trong tìm kiếm mô phỏng luyện kim, ta chọn ngẫu nhiên một trạng thái v trong lân cận u. Nếu trạng thái v được chọn tốt hơn u (cost(v) > cost(u)) thì ta đi tới v, còn nếu không ta chỉ đi tới v với một xác suất nào đó. Xác suất này giảm theo hàm mũ của "độ xấu" của trạng thái v. Xác suất này còn phụ thuộc vào tham số nhiệt độ T. Nhiệt độ T càng cao thì bước đi tới trạng thái xấu càng có khả năng được thực hiện.

Trong quá trình tìm kiếm, tham số nhiệt độ T giảm dần tới không. Khi T gần không, thuật toán hoạt động gần giống như leo đồi, hầu như nó không thực hiện bước tụt xuống. Cụ thể ta xác định xác suất đi tới trạng thái xấu v từ u là eΔ/T, ở đây Δ = cost(v) - cost(u).

Sau đây là thủ tục mô phỏng luyện kim.

```
procedure Simulated_Anneaning;
begin
    t ← 0;
    u ← trạng thái ban đầu nào đó;
    T ← nhiệt độ ban đầu;
    repeat
        v ← trạng thái được chọn nhẫu nhiên trong lân cận u;
        if cost(v) > cost(u) then u ← v
        else u ← v với xác suất e^(Δ/T);
        T ← g(T, t);
        t ← t + 1;
    until T đủ nhỏ
end;
```

Trong thủ tục trên, hàm g(T, t) thỏa mãn điều kiện g(T, t) < T với mọi t, nó xác định tốc độ giảm của nhiệt độ T. Người ta chứng minh được rằng, nếu nhiêt độ T giảm đủ chậm, thì thuật toán sẽ tìm được nghiệm tối ưu toàn cục. Thuật toán mô phỏng luyện kim đã được áp dụng thành công cho các bài toán tối ưu cỡ lớn.

2.3.3. Thuật toán di truyền

Thuật toán di truyền (Genetic Algorithm - GA) là kỹ thuật chung giúp giải quyết vấn đề bài toán bằng cách mô phỏng sự tiến hóa của con người hay của sinh vật nói chung (dựa trên thuyết tiến hóa muôn loài của Darwin) trong điều kiện qui định sẵn của môi trường. GA là một thuật giải, nghĩa là mục tiêu của GA không nhằm đưa ra lời giải chính xác tối ưu mà là đưa ra lời giải tương đối tối ưu.

Theo đề xuất ban đầu của giáo sư John Holland, một vấn đề-bài toán đặt ra sẽ được mã hóa thành các chuỗi bit với chiều dài cố định. Nói một cách chính xác là các thông số của bài toán sẽ được chuyển đổi và biểu diễn lại dưới dạng các chuỗi nhị phân. Các thông số này có thể là các biến của một hàm hoặc hệ số của một biểu thức toán học. Người ta gọi các chuỗi bit này là mã genome ứng với mỗi cá thể, các genome đều có cùng chiều dài. Nói ngắn gọn, một lời giải sẽ được biểu diễn bằng một chuỗi bit, cũng giống như mỗi cá thể đều được quy định bằng gen của cá thể đó vậy. Như vậy, đối với thuật giải di truyền, một cá thể chỉ có một gen duy nhất và một gen cũng chỉ phục vụ cho một cá thể duy nhất.

Ban đầu, ta sẽ phát sinh một số lượng lớn, giới hạn các cá thể có gen ngẫu nhiên. Nghĩa là phát sinh một tập hợp các chuỗi bit ngẫu nhiên. Tập các cá thể này được gọi là quần thể ban đầu (initial population). Sau đó, dựa trên một hàm nào đó, ta sẽ xác định được một giá trị gọi là độ thích nghi - Fitness. Giá trị này, có thể hiểu chính là độ "tốt"

của lời giải. Vì phát sinh ngẫu nhiên nên độ "tốt" của lời giải hay tính thích nghi của các cá thể trong quần thể ban đầu là không xác định.

Để cải thiện tính thích nghi của quần thể, người ta tìm cách tạo ra quần thể mới. Có hai thao tác thực hiện trên thế hệ hiện tại để tạo ra một thế hệ khác với độ thích nghi tốt hơn. Thao tác đầu tiên là sao chép nguyên mẫu một nhóm các cá thể tốt từ thế hệ trước rồi đưa sang thế hệ sau (selection). Thao tác này đảm bảo độ thích nghi của thế hệ sau luôn được giữ ở một mức độ hợp lý. Các cá thể được chọn thông thường là các cá thể có độ thích nghi cao nhất.

Thao tác thứ hai là tạo các cá thể mới bằng cách thực hiện các thao tác sinh sản trên một số cá thể được chọn từ thế hệ trước – thông thường cũng là những cá thể có độ thích nghi cao. Có hai loại thao tác sinh sản: một là lai tạo (crossover), hai là đột biến (mutation). Trong thao tác lai tạo, từ gen của hai cá thể được chọn trong thế hệ trước sẽ được phối hợp với nhau (theo một số quy tắc nào đó) để tạo thành hai gen mới.

Thao tác chọn lọc và lai tạo giúp tạo ra thế hệ sau. Tuy nhiên, nhiều khi do thế hệ khởi tạo ban đầu có đặc tính chưa phong phú và chưa phù hợp nên các cá thể không rải đều được hết không gian của bài toán. Từ đó, khó có thể tìm ra lời giải tối ưu cho bài toán. Thao tác đột biến sẽ giúp giải quyết được vấn đề này. Đó là sự biến đổi ngẫu nhiên một hoặc nhiều thành phần gen của một cá thể ở thế hệ trước tạo ra một cá thể hoàn toàn mới ở thế thệ sau. Nhưng thao tác này chỉ được phép xảy ra với tần suất rất thấp (thường dưới 0.01), vì thao tác này có thể gây xáo trộn và làm mất đi những cá thể đã chọn lọc và lai tạo có tính thích nghi cao, dẫn đến thuật toán không còn hiệu quả.

Thế hệ mới được tạo ra lại được xử lý như thế hệ trước (xác định độ thích nghi và tạo thế hệ mới) cho đến khi có một cá thể đạt được giải pháp mong muốn hoặc đạt đến thời gian giới hạn.

Thuật giải di truyền sử dụng các toán tử cơ bản sau đây để biến đổi các thế hệ.

- ✓ Toán tử tái sinh (reproduction) (còn được gọi là toán tử chọn lọc (selection)). Các cá thể tốt được chọn lọc để đưa vào thế hệ sau. Sự lựa chọn này được thực hiện dựa vào độ thích nghi với môi trường của mỗi cá thể. Ta sẽ gọi hàm ứng mỗi cá thể với độ thích nghi của nó là hàm thích nghi (fitness function).
- ✓ Toán tử lai ghép (crossover). Hai cá thể cha và mẹ trao đổi các gien để tạo ra hai cá thể con.
- ✓ Toán tử đột biến (mutation). Một cá thể thay đổi một số gien để tạo thành cá thể mới.

Tất cả các toán tử trên khi thực hiện đều mang tính ngẫu nhiên. Cấu trúc cơ bản của TTDT là như sau:

procedure *Genetic_Algorithm*;
begin
    t ← *0*;
    Khởi tạo thế hệ ban đầu P(t);
    Đánh giá P(t) (theo hàm thích nghi);
  repeat
    t ← *t* + *1*;

Sinh ra thế hệ mới P(t) từ P(t-1) bởi

- *Chọn lọc*
- *Lai ghép*
- *Đột biến;*

Đánh giá P(t);

until *điều kiện kết thúc được thỏa mãn*;

end;

Trong thủ tục trên, điều kiện kết thúc vòng lặp có thể là một số thế hệ đủ lớn nào đó, hoặc độ thích nghi của các cá thể tốt nhất trong các thế hệ kế tiếp nhau khác nhau không đáng kể. Khi thuật toán dừng, cá thể tốt nhất trong thế hệ cuối cùng được chọn làm nghiệm cần tìm.

2.4. CÁC CHIẾN LƯỢC TÌM KIẾM CÓ ĐỐI THỦ

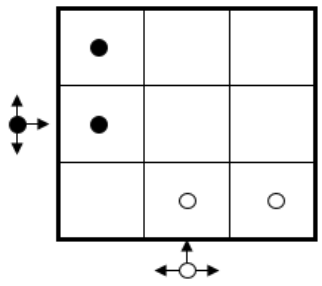
Bài toán tìm kiếm có đối thủ (chơi cờ) được biểu diễn trong không gian trạng thái:

- ✓ Trạng thái ban đầu: sự sắp xếp các quân cờ của hai bên lúc bắt đầu chơi.
- ✓ Các toán tử: các nước đi hợp lệ
- ✓ Các trạng thái kết thúc: các tình thế mà cuộc chơi dừng.
- ✓ Hàm kết cuộc: ứng mỗi trạng thái kết thúc với một giá trị nào đó.

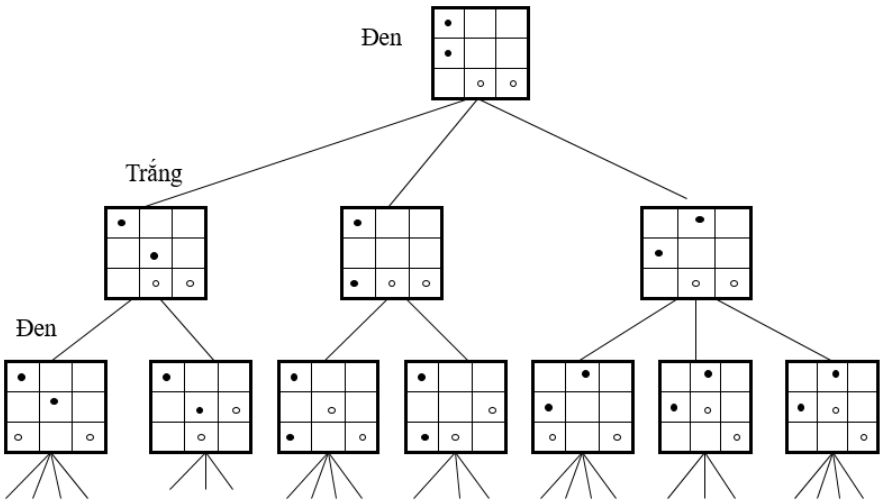
2.4.1. Cây trò chơi và tìm kiếm trên cây trò chơi

Cây trò chơi được xây dựng như sau. Gốc của cây ứng với trạng thái ban đầu. Ta sẽ gọi đỉnh ứng với trạng thái mà Trắng (Đen) đưa ra nước đi là đỉnh Trắng (Đen). Nếu một đỉnh là Trắng (Đen) ứng với trạng thái u, thì các đỉnh con của nó là tất cả các đỉnh biểu diễn trạng thái v, v nhận được từ u do Trắng (Đen) thực hiện nước đi hợp lệ nào đó. Do đó, trên cùng một mức của cây các đỉnh đều là Trắng hặc đều là Đen, các lá của cây ứng với các trnạg thái kết thúc.

Ví dụ 2.19: Xét trò chơi Dodgen (được tạo ra bởi Colin Vout). Có hai quân Trắng và hai quân Đen, ban đầu được xếp vào bàn cờ 3*3 (Hình vẽ). Quân Đen có thể đi tới ô trống ở bên phải, ở trên hoặc ở dưới. Quân Trắng có thể đi tới trống ở bên trái, bên phải, ở trên. Quân Đen nếu ở cột ngoài cùng bên phải có thể đi ra khỏi bàn cờ, quân Trắng nếu ở hàng trên cùng có thể đi ra khỏi bàn cờ. Ai đưa hai quân của mình ra khỏi bàn cờ trước sẽ thắng, hoặc tạo ra tình thế bắt đối phương không đi được cũng sẽ thắng.



Hình 2.26: Trò chơi Dodgen



Hình 2.27: Cây trò chơi Dodgen với đen đi trước

2.4.2. Chiến lược minimax

Quá trình chơi cờ là quá trình Trắng và Đen thay phiên nhau đưa ra quyết định, thực hiện một trong số các nước đi hợp lệ. Trên cây trò chơi, quá trình đó sẽ tạo ra đường đi từ gốc tới lá. Giả sử tới một thời điểm nào đó, đường đi đã dẫn tới đỉnh u. Nếu u là đỉnh Trắng (Đen) thì Trắng (Đen) cần chọn đi tới một trong các đỉnh Đen (Trắng) v là con của u. Tại đỉnh Đen (Trắng) v mà Trắng (Đen) vừa chọn, Đen (Trắng) sẽ phải chọn đi tới một trong các đỉnh Trắng (Đen) w là con của v. Quá trình trên sẽ dừng lại khi đạt tới một đỉnh là lá của cây.

Giả sử Trắng cần tìm nước đi tại đỉnh u. Nước đi tối ưu cho Trắng là nước đi dần tới đỉnh con của v là đỉnh tốt nhất (cho Trắng) trong số các đỉnh con của u. Ta cần giả thiết rằng, đến lượt đối thủ chọn nước đi từ v, Đen cũng sẽ chọn nước đi tốt nhất cho anh ta. Như vậy, để chọn nước đi tối ưu cho Trắng tại đỉnh u, ta cần phải xác định giá trị các đỉnh của cây trò chơi gốc u. Giá trị của các đỉnh lá (ứng với các trạng thái kết thúc) là giá trị của hàm kết cuộc. Đỉnh có giá trị càng lớn càng tốt cho Trắng, đỉnh có giá trị càng nhỏ càng tốt cho Đen. Để xác định giá trị các đỉnh của cây trò chơi gốc u, ta đi từ mức thấp nhất lên gốc u. Giả sử v là đỉnh trong của cây và giá trị các đỉnh con của nó đã được xác định. Khi đó nếu v là đỉnh Trắng thì giá trị của nó được xác định là giá trị lớn nhất trong các giá trị của các đỉnh con. Còn nếu v là đỉnh Đen thì giá trị của nó là giá trị nhỏ nhất trong các giá trị của các đỉnh con.

Hai đấu thủ trong trò chơi được gọi là MIN và MAX.

Mỗi nút lá có giá trị:

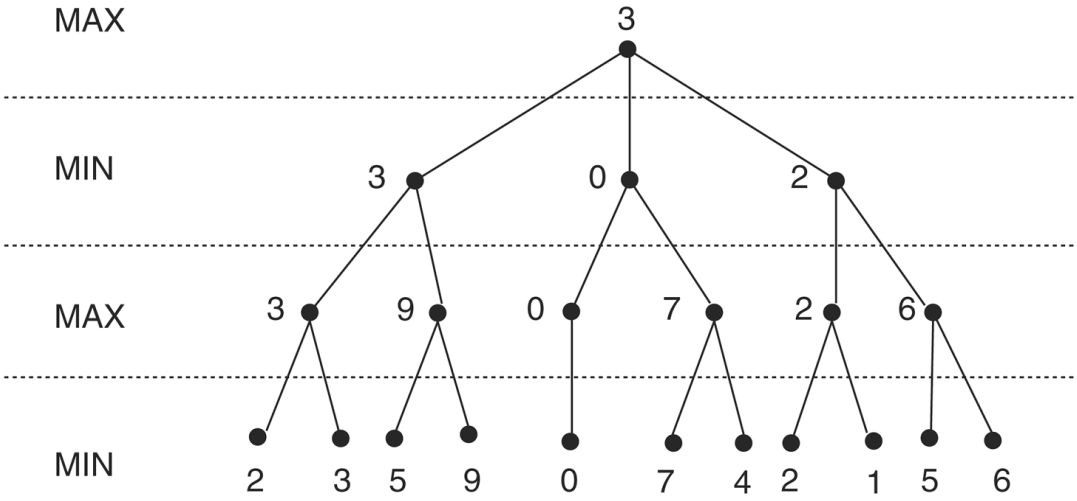
- ✓ 1 nếu là MAX thắng,
- ✓ 0 nếu là MIN thắng.

Minimax sẽ truyền các giá trị này lên cao dần trên đồ thị, qua các nút cha mẹ kế tiếp theo các luật sau:

Nếu trạng thái cha mẹ là MAX, gán cho nó giá trị lớn nhất có trong các trạng thái con.

Nếu trạng thái cha mẹ là MIN, gán cho nó giá trị nhỏ nhất có trong các trạng thái con.

Minimax đối với một KGTT giả định.



Hình 2.28: Minimax với độ sâu lớp cố định

Các nút lá được gán các giá trị heuristic
Còn giá trị tại các nút trong là các giá trị nhận được dựa trên giải thuật Minimax
Giải thuật Minimax

```
Function MaxVal(u);
begin
        if  u là đỉnh kết thúc  then  MinVal(u) ← f(u)
        else MinVal(u) ← min{MaxVal(v) | v là đỉnh con của u}
end;
Function MinVal(u);
begin
        if  u là đỉnh kết thúc  then  MaxVal(u) ← f(u)
        else MaxVal(u) ← max{MinVal(v) | v là đỉnh con của u}
end;
Procedure Minimax(u, v);
begin
        val ←-∝;
        for  mỗi w là đỉnh con của u  do
        if  val(u) <= MinVal(w)  then
                {val ← MinVal(w);  v ←  w}
end;
```

2.4.3. Phương pháp cắt cụt Anpha-Beta

Trong chiến lược tìm kiếm Minimax, để tìm kiếm nước đi tốt cho Trắng tại trạng thái u, cho dù ta hạn chế không gian tìm kiếm trong phạm vi cây trò chơi gốc u với độ cao h, thì số đỉnh của cây trò chơi này cũng còn rất lớn với h ≥ 3. Chẳng hạn, trong cờ vua, nhân tố nhánh trong cây trò chơi trung bình khoảng 35, thời gian đòi hỏi phải đưa ra nước đi là 150 giây, với thời gian này trên máy tính thông thường chương trình của bạn chỉ có thể xem xét các đỉnh trong độ sâu 3 hoặc 4. Một người chơi cờ trình độ trung bình cũng

có thể tính trước được 5, 6 nước hoặc hơn nữa, và do đó chương trình của bạn mới đạt trình độ người mới tập chơi.

Khi đánh giá đỉnh *u* tới độ sâu *h*, một thuật toán Minimax đòi hỏi ta phải đánh giá tất cả các đỉnh của cây gốc u tới độ sâu h. Song ta có thể giảm bớt số đỉnh cần phải dánh giá mà vẫn không ảnh hưởng gì đến sự đánh giá đỉnh u. Phương pháp cắt cụt alpha-beta cho phép ta cắt bỏ các nhánh không cần thiết cho sự đánh giá đỉnh u.

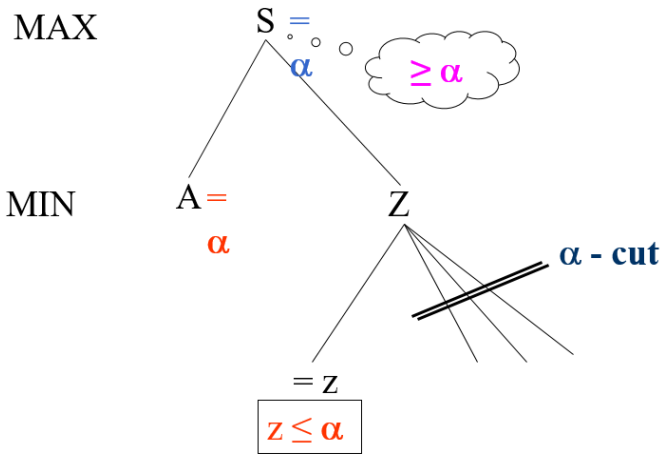
Tư tưởng của kỹ thuật cắt cụt alpha-beta là như sau: Nhớ lại rằng, chiến lược tìm kiếm Minimax là chiến lược tìm kiếm theo độ sâu. Giả sử trong quá trính tìm kiếm ta đi xuống đỉnh a là đỉnh Trắng, đỉnh a có người anh em v đã được đánh giá. Giả sử cha của đỉnh a là b và b có người anh em u dã được đánh giá, và giả sử cha của b là c (Xem hình 4.7). Khi đó ta có giá trị đỉnh c (đỉnh Trắng) ít nhất là giá trị của u, giá trị của đỉnh b (đỉnh Đen) nhiều nhất là giá trị v. Do đó, nếu eval(u) > eval(v), ta không cần đi xuống để đánh giá đỉnh a nữa mà vẫn không ảnh hưởng gì dến đánh giá đỉnh c. Hay nói cách khác ta có thể cắt bỏ cây con gốc a. Lập luận tương tự cho trường hợp a là đỉnh Đen, trong trường hợp này nếu eval(u) < eval(v) ta cũng có thể cắt bỏ cây con gốc a.

Để cài đặt kỹ thuật cắt cụt alpha-beta, đối với các đỉnh nằm trên đường đi từ gốc tới đỉnh hiện thời, ta sử dụng tham số α để ghi lại giá trị lớn nhất trong các giá trị của các đỉnh con đã đánh giá của một đỉnh Trắng, còn tham số β ghi lại giá trị nhỏ nhất trong các đỉnh con đã đánh giá của một đỉnh Đen. Giá trị của α và β sẽ được cập nhật trong quá trình tìm kiếm. α và β được sử dụng như các biến địa phương trong các hàm MaxVal(u, α, β) (hàm xác định giá trị của đỉnh Trắng u) và Minval(u, α, β) (hàm xác định giá trị của đỉnh Đen u).

Giải thuật cắt tỉa α-β

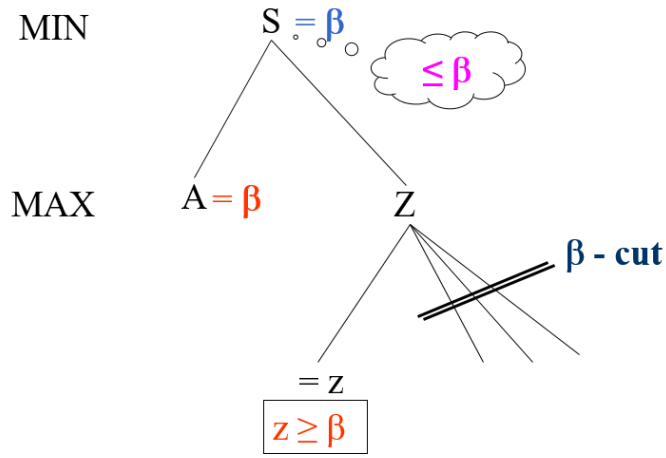
- ✓ Tìm kiếm theo kiểu depth-first.
- ✓ Nút MAX có 1 giá trị α (luôn tăng)
- ✓ Nút MIN có 1 giá trị β (luôn giảm)
- ✓ TK có thể kết thúc dưới bất kỳ:
  - Nút MIN nào có $\beta \leq \alpha$ của bất kỳ nút cha MAX nào.
  - Nút MAX nào có $\alpha \geq \beta$ của bất kỳ nút cha MIN nào.
- ✓ Giải thuật cắt tỉa α-β thể hiện *mối quan hệ giữa các nút ở lớp n và n+2*, mà tại đó toàn bộ cây có gốc tại lớp n+1 có thể cắt bỏ.

Cắt tỉa α



Hình: Cắt tỉa α

Cắt tỉa β



Hình 2.29: Cắt tỉa β

- **Function** MaxVal(u,α, β);
  begin
  if u là lá của cây hạn chế hoặc đỉnh kết thúc **then** MaxVal←eval(u)
  else for mỗi đỉnh v là con của u **do**
  {α ← max[α,MinVal(v,α,β)];
  if α ≥ β **then** exit};
  MaxVal ←α;
  end;
- **Function** MinVal(u,α, β);
  begin
  if u là lá của cây hạn chế hoặc đỉnh kết thúc **then** MinVal←eval(u)
  else for mỗi đỉnh v là con của u **do**
  {β ←min[β,MaxVal(v,α,β)];
  if α ≥ β **then** exit};
  MinVal ← β;
  end;

```
Procedure Alpha_beta(u,v);
begin
        α←-∝;
        β←∝;
        for mỗi đỉnh w là con của u  do
                if α ≤ MinVal(w, α, β)  then
                    {α ← MinVal(w, α, β);
                     v ←w; }
end;
```

CÂU HỎI, BÀI TẬP

1. Xây dựng không gian trạng thái gồm 8 địa điểm trong thực tế. Hãy xác định các đường đi từ một điểm bất kỳ đến các điểm khác và áp dụng thuật toán tìm kiếm tốt nhất, tìm kiếm leo đồi vào không gian trạng thái trên.

2. Xây dựng không gian trạng thái cho trò chơi xếp hình n = 2. Hãy chỉ ra trạng thái đầu, trạng thái đích, thứ tự ưu tiên. Áp dụng thuật toán tìm kiếm tốt nhất đầu tiên, tìm kiếm leo đồi vào không gian trạng thái trên.

3. Xây dựng không gian trạng thái gồm 8 địa điểm trong thực tế. Hãy xác định các đường đi từ một điểm bất kỳ đến các điểm khác và áp dụng thuật toán A*, tìm kiếm nhánh cận vào không gian trạng thái trên.

4. Xây dựng không gian trạng thái cho trò chơi xếp hình n = 2. Hãy chỉ ra trạng thái đầu, trạng thái đích, thứ tự ưu tiên. Áp dụng thuật toán A*, tìm kiếm nhánh cận vào không gian trạng thái trên.