Bài 2

CÁC THUẬT TOÁN TÌM KIẾM

2.1. Các thuật toán tìm kiếm không có thông tin

- Thuật toán tìm kiếm theo bề rộng.
- Thuật toán tìm kiếm theo độ sâu.
- Thuật toán tìm kiếm theo độ sâu lặp.

2.1.1 Thuật toán tìm kiếm theo bề rộng (Breadth First Search)

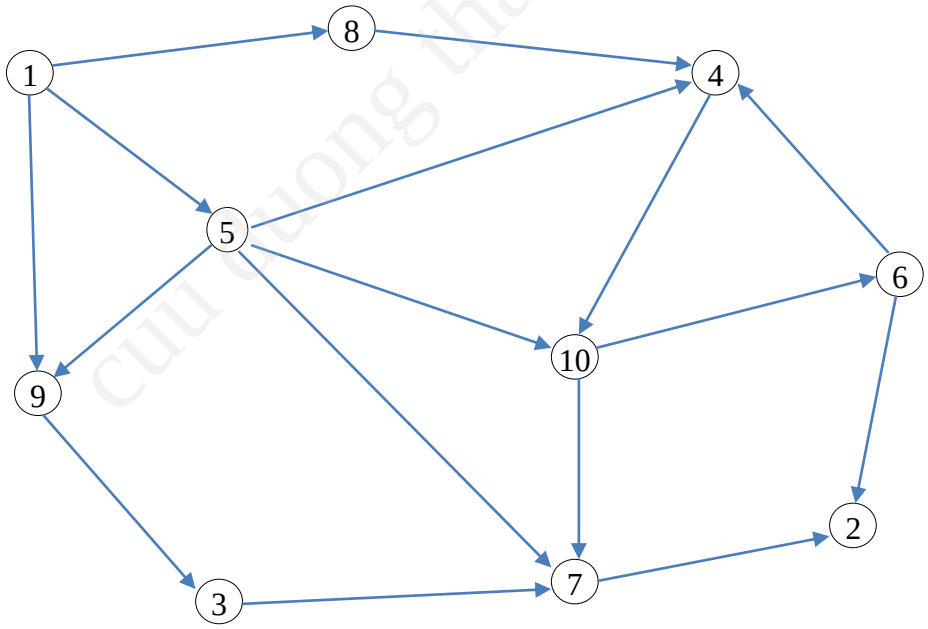
- Phát triển các đỉnh lần lượt theo mức 0,1,…. Khitất cả các đỉnh ở một mức được phát triển xong rồi mới phát triển những đỉnh ở mức tiếp theo.

- Tại mỗi mứcta sẽ chọn đỉnh để phát triển là đỉnh được sinh ra trước nhất so với các đỉnh chờ phát triển khác.

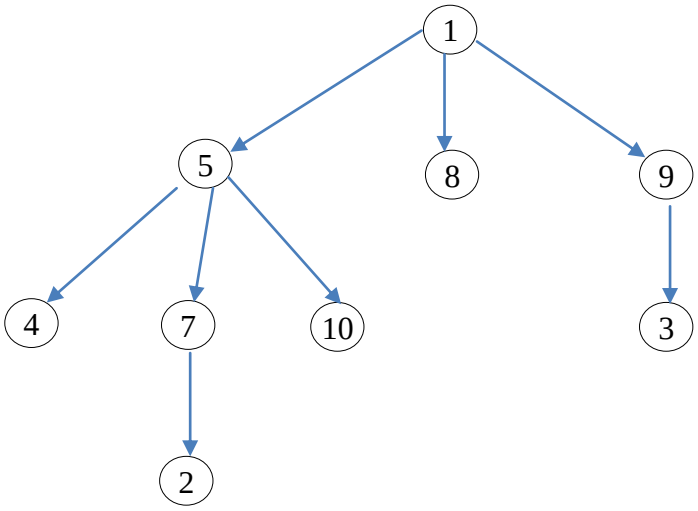
✓ **Ví dụ:**

Cho không gian trang thái như hình 2.1. Tìm dãy các phép biến đổi để biến đổi 1 thành 2.

Ta có cây tìm kiếm BFS như hình 2.2. Quá trình thuật toán thực hiện như hình 2.3



Hình 2.1: Không gian trang thái



Hình 2.2 : Cây tìm kiếm BFS

Bước lặp	Open	Close
Khởi tạo	(1,null) // đỉnh 1 có cha là null	Rỗng
1	(5,1),(8,1),(9,1) //5 cha là 1	1
2	(8,1),(9,1),(4,5),(7,5),(10,5)	1,5
3	(9,1),(4,5),(7,5),(10,5)	1,5,8
4	(4,5),(7,5),(10,5),(3,9)	1,5,8,9
5	(7,5),(10,5),(3,9)	1,5,8,9,4
6	(10,5),(3,9),(2,7)//mở được 2 => ngừng	1,5,8,9,4,7

Hình 2.3: các bước BFS thực hiện và cho kết quả: 1->5->7->2

✓ **Thuật toán:**

```
bool  Breadth_First_Search {
    if (trạng thái ban đầu là trạng thái kết thúc)  return true; //tìm kiếm thành công
    khởi tạo danh sách open chỉ chứa đỉnh bắt đầu;
    khởi tạo danh sách close rỗng;
    while (true){
        if (open rỗng)  return  false; //tìm kiếm thất bại
        chọn và loại đỉnh u ở đầu danh sách  open;
        thêm u vào danh sách close;
        for (mỗi đỉnh v kề u) {
            if (v không có trong close và v không có trong open) {
                father(v) =u;
                if (v là đỉnh kết thúc)  return true; //tìm kiếm thành công
```

thêm v vào cuối danh sách open;

}

}

}

}

✓ **Nhận xét:**

- Đỉnh sinh ra trước sẽ được phát triển trước, do đó danh sách open được xử lý như hàng đợi (queue).

- Nếu có đường đi từ trạng thái ban đầu tới trạng thái đích thì thuật toán tìm kiếm theo bề rộng sẽ luôn tìm ra với số phép biến đổi ít nhất.Nếukhông có đường đi từ trạng thái ban đầu tới trạng thái đích và không gian trạng thái hữu hạn, thì thuật toán sẽ dừng và thông báo tìm kiếm thất bại.

- Giả sử mỗi đỉnh khi được phát triển sẽ sinh ra b đỉnh kề (b gọi làhệ số nhánh) vàđỉnh kết thúcở mức d của cây tìm kiếm. Dođỉnh kết thúc có thể tìm được tại một đỉnh bất kỳ ở mức d, nên số đỉnh cần xét nhiều nhất để tìm ra đỉnh kết thúc là:

  $1 + b + b^2 + ... + b^{d-1} + b^d$

  Như vậy, độ phức tạp thời gian của thuật toán tìm kiếm theo bề rộng là $O(b^d)$. Độ phức tạp không gian (bộ nhớ) cũng là $O(b^d)$ vì ta cũng lưu vào danh sách open/close tất cả các đỉnh của cây tìm kiếm tới mức d.

- Để thấy rõ tìm kiếm theo bề rộng đòi hỏi thời gian và không gian lớn tới mức nào, ta xét trường hợp b = 10 và giả sử kiểm tra 1000 trạng thái cần 1 giây, và lưu giữ 1 trạng thái cần 100 bytes. Khi đó thời gian và không gian mà thuật toán đòi hỏi được cho trong bảng sau:

Độ sâu d	Thời gian	Không gian
4	11 giây	1 megabyte
6	18 giây	111 megabytes
8	31 giờ	11 gigabytes
10	128 ngày	1 terabyte
12	35 năm	111 terabytes
14	3500 năm	11.111 terabytes

Với độ sâu 10 trở lên là thuật toánkhông thể chấp nhận được!

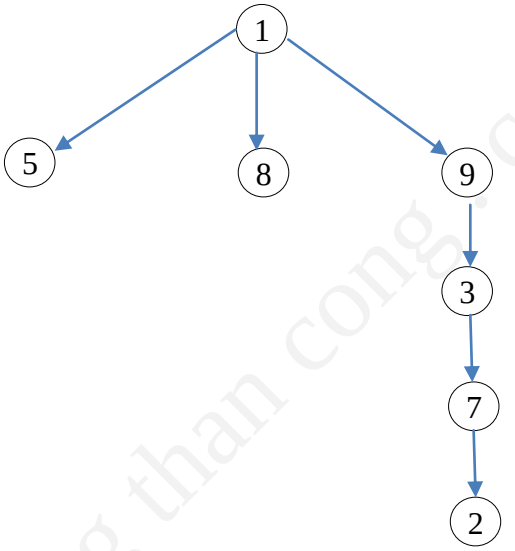
2.1.2 Thuật toán tìm kiếm theo độ sâu (Depth First Search)

- Tại mỗi bước, đỉnh được chọn để phát triển là đỉnh được sinh ra sau cùng trong số các đỉnh chờ phát triển.
- Trong thuật toán tìm kiếm theo bề rộng, ta sửa**"thêm v vào cuối danh sách open"**thành **"thêmv vào đầu danh sách open"** thì sẽ có Thuật toán tìm kiếm theo độ sâu.

✓ **Ví dụ:**

Cho không gian trang thái như hình 2.1. Tìm dãy các phép biến đổi để biến đổi 1 thành 2.

Ta có cây tìm kiếm DFS như hình 2.4. Quá trình thuật toán thực hiện như hình 2.5



Hình 2.4 : Cây tìm kiếm DFS

Bước lặp	Open	Close
Khởi tạo	(1,null) // đỉnh 1 có cha là null	Rỗng
1	(9,1),(8,1),(5,1) //5 cha là 1	1
2	(3,9),(8,1),(5,1)	1,9
3	(7,3),(8,1),(5,1)	1,9,3
4	(2,7),(8,1),(5,1) //mở được 2 => ngừng	1,9,3,7

Hình 2.5: các bước DFS thực hiện và cho kết quả: 1->9->3->7->2

✓ **Thuật toán:**

```
bool  Depth_First_Search {
    if (trạng thái ban đầu là trạng thái kết thúc)  return true; //tìm kiếm thành công
    khởi tạo danh sách open chỉ chứa đỉnh bắt đầu;
    khởi tạo danh sách close rỗng;
    while (true){
        if (open rỗng)  return  false; //tìm kiếm thất bại
        chọn và loại đỉnh u ở đầu danh sách  open;
        thêm u vào danh sách close;
        for (mỗi đỉnh v kề u) {
            if (v không có trong close và v không có trong open) {
                father(v) =u;
                if (v là đỉnh kết thúc)  return true; //tìm kiếm thành công
                thêm v vào đầu danh sách  open;
            }
        }
    }
}
```

✓ **Nhận xét:**

- Do đỉnh sinh ra sau sẽ được phát triển trước, nên danh sách open được xử lý như ngăn xếp (stack).

- Nếu có trạng thái kết thúc và không gian trạng thái hữu hạn, thì thuật toán tìm kiếm theo độ sâusẽ tìm ra. Nếu không gian trạng thái vô hạn thì có thể không tìm ra, vì nếu thuật toán đi theo một nhánh vô hạn mà trạng thái kết thúc không nằm trên nhánh này thì thuật toán sẽ không dừng.

- Giả sửtrạng thái kết thúc ở mức d và cây tìm kiếm có hệ số nhánh là b. Trường hợp xấu nhất làtrạng thái kết thúc ở đỉnh ngoài cùng của mức d, do đó độ phức tạpthời gian của tìm kiếm theo độ sâu trong trường hợp xấu nhất là $O(b^d)$.

- Khi một đỉnh u trên cây tìm kiếm theo độ sâu, có các đỉnh “hậu duệ”đã mở thì đỉnh u có thể hủy. Do đó độ phức tạp không gian của tìm kiếm theo độ sâu có thể rút gọn là O(db).

- Tìm kiếm theo độ sâu thường nhanh hơn tìm kiếm theo bề rộng,vì tìm kiếm theo bề rộng phải xem xét toàn bộ cây tìm kiếm tới mức d-1, rồi mới xem xét các đỉnh ở mức d. Còn trong tìm kiếm theo độ sâu, có thể ta chỉ cần xem xét một bộ phận nhỏ của cây tìm kiếm thì có thể đã tìm ra trạng thái kết thúc.

2.1.3. Thuật toán tìm kiếm sâu lặp (Depth Deepening Search)

- Nếu cây tìm kiếm chứa nhánh vô hạn, tìm kiếm theo độ sâu có thể mắc kẹt ở nhánh vô hạn và không tìm ratrạng thái kết thúc.

- Để khắc phục ta lần lượt tìm kiếm theo độ sâu mức 0, nếu không tìm ra trạng thái kết thúc, talại tìm kiếm theođộ sâu mức 1,2,…. Quá trình được lặp lại dếnmột độ sâu Max nào đó do ta chọn.Nếu vẫn không tìm thấy thì thông báo "tìm kiếm thất bại" hoặc tăng Max và tìm lại.

✓ **Thuật toán tìm kiếmđến độ sâud**

```
boolDepth_Limited_Search(d) {
    if (trạng thái ban đầu là trạng thái kết thúc)  return true;
    khởi tạo danh sách  open chỉ có đỉnh bắt đầu chứa trạng thái ban đầu u0;
    khởi tạo danh sách close rỗng;
    depth(u0)=0;
    while(true){
        if (open rỗng) return  false;
        chọn và loại đỉnh u ở đầu danh sách  open;
        thêm u vào danh sách close;
        if (depth(u) < d) {
            for (mỗi đỉnh v kề u) {
                if (v không có trong close và v không có trong open) {
                    father(v) =u;
                    depth(v)=depth(u) + 1;
                    if (v là đỉnh kết thúc) return true;
                    thêm v vào đầu danh sách open;
                }
            }
        }
    }
}
```

✓ **Thuật toán tìm kiếm sâu lặp**

```
boolDepth_Deepening_Search (max){
    for (d =0 to max){
            result= Depth_Limited_Search (d);
            if (result==true) return true;
    }
    return false;
}
```

✓ **Nhận xét**

- Kỹ thuật tìm kiếm sâu lặp kết hợp được các ưu điểm của tìm kiếmtheo bề rộng và tìm kiếm theo độ sâu.Cũng như tìm kiếm theo bề rộng, nếu có trạng thái kết thúc, tìm kiếm sâu lặp luôn tìm ramiễn là ta chọn độ sâu max đủ lớn.

- Tìm kiếm sâu lặp với độ sâu d cần bộ nhớ như tìm kiếm theo độ sâu tức là O(db).Tổng số đỉnh cần phát triển trong tìm kiếm sâu lặpvới độ sâu d là:$(d+1) + db + (d-1)b^2+... + 2b^{d-1}+ 1b^d$
  Do đó thời gian tìm kiếm sâu lặp cũng là $O(b^d)$.

- Nên áp dụng tìm kiếm sâu lặp cho không gian trạng thái lớn và độ sâu của trạng thái kết thúc không biết trước.

✓ **Tóm tắt**

Độ phức tạp	**BFS**	**DFS**	**DDS**
Thời gian	$O(b^d)$	$O(b^d)$	$O(b^d)$
Bộ nhớ	$O(b^d)$	O(bd)	O(bd)

2.2. Các thuật toán tìm kiếm có thông tin(informed search/heuristic search)

- Nếu không gian trạng thái rất lớn, các thuật toán“tìm kiếm không có thông tin” không thể áp dụng vì tốn rất nhiều bộ nhớ và thời gian.

- Các thuật toán“tìm kiếm có thông tin” giống với các thuật toán “tìm kiếm không có thông tin” nhưng khi chọn một đỉnh để phát triển,ta không chọn ngẫu nhiên mà chọndựa vào hàm đánh giá.Thông thường thuật toán “tìm kiếm có thông tin”nhanh chóng tìm ra lời giải nếu có.

- **Hàm đánh giá trạng thái:**thường có ba hàm đánh giá một trạng thái

- ✓ g(u):chi phí nhỏ nhất để chuyểntừ đỉnh ban đầu tới đỉnh u. g(u) gọi là hàm đánh giá dựa vào thông tin hiện tại.

- ✓ h(u):chi phí nhỏ nhất dự kiến để chuyển từ trạng thái u tới trạng thái kết thúc. h(u) gọi là hàm đánh giá dựa vào thông tin tương lai. Cần xây dựng h(u) sao cho h(u) ≤chi phí thực tế để chuyển từ trạng thái u tới trạng thái kết thúc và càng gần với chi phí thực tế thì càng tốt.

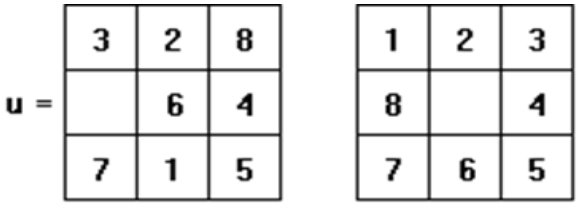
- ✓ f(u):chi phí nhỏ nhất để di chuyển từ trạng thái ban đầuqua u, tới trạng thái kết thúc . f(u) gọi là hàm đánh giá đỉnh u. Thông thường f(u) = g(u) + h(u).

- Hàm đánh giá cần xây dựng hợp lý thì tìm kiếm mới hiệu quả.Trong quá trình tìm kiếm, tại mỗi bước ta sẽ chọn đỉnh để phát triển là đỉnh có giá trị hàm đánh giá nhỏ nhất(hoặc lớn nhất), đỉnh này được xem là đỉnh có nhiều hứa hẹn dẫn tới đỉnh kết thúc nhanh nhất, ít tốn chi phí nhất.

Một sốví dụ về hàm đánh giá:

- **Tìm đường đi trên bản đồ**
  - ✓ g(u) có thể là là chiều dài đường đitừ thành phố xuất phát đến thành phố u.
  - ✓ h(u) có thể là độ dài của đường chim bay từ thành phố u tới thành phố đích.
  - ✓ f(u) = g(u) + h(u) và ta chọn đỉnh u có f(u) nhỏ nhất để phát triển.

- **Tròchơi 8 số:**
  - ✓ g(u) có thể là số lần dịch chuyểnđể biến đổi đỉnh ứng với trạng thái ban đầu đến đỉnh ứng với trạng thái u
  - ✓ h(u) có thể là hàm $h_1(u)$ hoặc $h_2(u)$ như sau:
    - $h_1(u)$: là số các số không nằm đúng vịtrí trong trạng thái đích. Như hình 2.1, $h_1(u) = 4$, vì các các số khôngđúng vị trí là 3, 8, 6 và 1.
    - $h_2(u)$: Gọi khoảng cách của một số làsố bước dịch chuyển ít nhất theo hàng và/hoặc cột để chuyểnsốđó tới vị trí củanó trong trạng thái đích.Khi đó $h_2(u)$ là tổng các khoảng cách của các số trong trạng thái u.Ví dụ hình2.1,$h_2(u) = 2 + 3 + 1 + 3 = 9$, vì khoảng cách của số 3 là 2, số 8 là 3, số 6 là 1 và số 1 là 3.



Hình 2.1 Đánh giá trạng thái u.

- ✓ f(u) = g(u) + h(u).

- **Các thuật toán tìm kiếm có thông tin**
  - ✓ Tìm kiếm tốt nhất đầu tiên: Tìm kiếm theo bề rộng và sử dụnghàm đánh giá.
  - ✓ Tìm kiếm leo đồi: Tìm kiếm theo độ sâu và sử dụng hàm đánh giá.
  - ✓ Thuật toán A*: Cải tiến thuật toán tìm kiếm tốt nhất đầu tiên.

2.2.1Thuật toán tìm kiếm tốt nhất đầu tiên (best-first search)

- Tìm kiếm tốt nhất - đầu tiên là tìm kiếm theo bề rộng được hướng dẫn bởi hàm đánh giá.

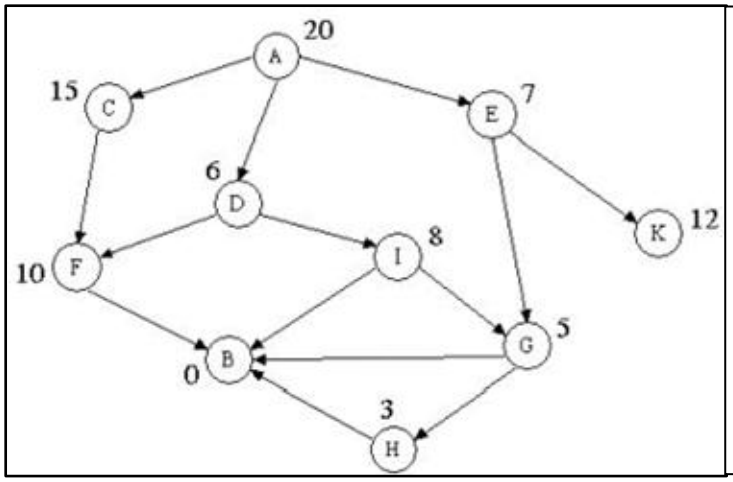
- Open được sắp xếp tăng theo giá trị hàm đánh giá. Đỉnh được chọn để mở là đỉnh ở đầu open

- Trong tìm kiếm theo bề rộng ta lần lượt phát triển tất cả cácđỉnh ở mức hiện tại để sinh ra các đỉnh ở mức tiếp theo. Trong tìm kiếm tốt nhất - đầu tiên ta chọn đỉnh để phát triển là đỉnh được đánh giá tốt nhất trong số các đỉnh đang chờ phát triển, đỉnh này có thể ở mức hiện tại hoặc ở các mức trên.

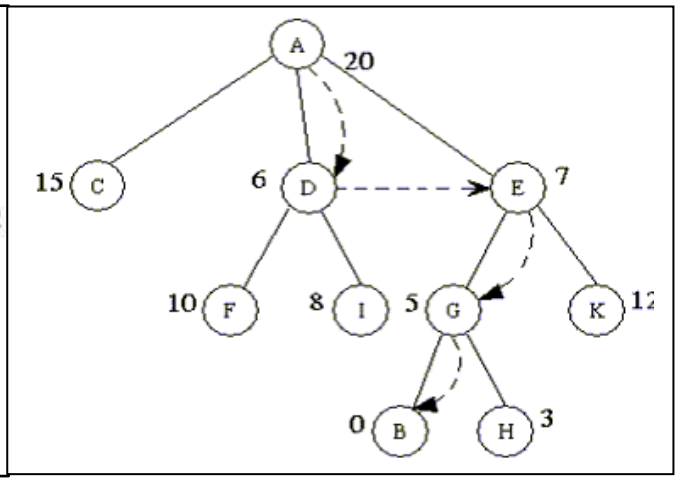
✓ **Ví dụ:**

Xét không gian trạng thái được biểu diễn bởi đồ thị trong hình 2.6.Trạng thái ban đầu là A, trạng thái kết thúc là B. Số ghi kế bên mỗi đỉnh là giá trị của hàm đánh giá.Quá trình tìm kiếm tốt nhất - đầu tiêndiễn 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 2.7.



Hình 2.6: Đồ thị không gian trạng thái



Hình 2.7: cây tìm kiếm tốt nhất đầu tiên

Bước lặp	Open	Close
Khởi tạo	(A,20,null) // A có giá trị f=20, và có cha là null	Rỗng
1	(D,6,A),(E,7,A),(C,15,A)	A
2	(E,7,A),(I,8,D),(F,10,D),(C,15,A)	A,D
3	(G,5,E),(I,8,D),(F,10,D),(K,12,E),(C,15,A)	A,D,E
4	(B,0,G),(H,3,G),(I,8,D),(F,10,D),(K,12,E),(C,15,A)	A,D,E,G

Mở được đỉnh B => ngừng, kết quả: A->E->G->B

✓ **Thuật toán**

bool Best_First_Search {

if (trạng thái ban đầu là trạng thái kết thúc) return true; //tìm kiếm thành công

khởi tạo danh sách open chỉ chứa đỉnh bắt đầu;

khởi tạo danh sách close rỗng;

while(true){

if (open rỗng) return false;

chọn và loại đỉnh u ở đầu danh sách open;

thêm u vào danh sách close;

for (mỗi đỉnh v kề u) {

if (v không có trong close và v không có trong open) {

father(v) =u;

if (v là trạng thái kết thúc) return true;

chèn v vào open sao cho open sắp tăng theo giá trị hàm đánh giá;

}

```
            }
        }
}
```

2.2.2 Thuật toán tìm kiếm leo đồi (hill-climbing search)

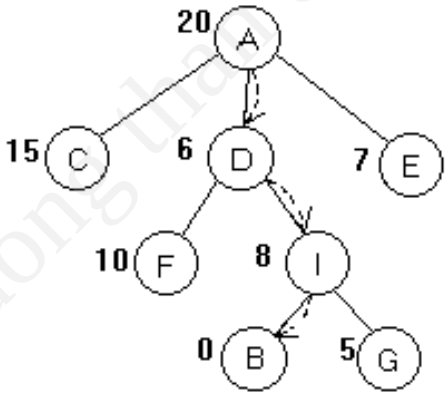
- Tìm kiếm leo đồi chính là tìm kiếm theo độ sâu đượchướng dẫn bởi hàm đánh giá.

- Khi chọn một đỉnh để phát triển, ta chọnđỉnh được đánh giá tốtnhất ở mức hiện tại.

✓ **Ví dụ:**

Xét đồ thị không gian trạng thái trong hình 2.6. 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 (vìf(D)= 6 nhỏ nhất),và sinh ra F, I. Trong hai đỉnh này, ta chọn I để phát triển, và sinh ra các đỉnh con B, G. Do tìm được B nên thuật toán kết thúc. Cây tìm kiếm leo đồi được cho trong hình 2.8.



Hình 2.8: cây tìm kiếm leo đồi

✓ **Thuật toán**

```
bool  Hill_Climbing_Search {
        if (trạng thái ban đầu là trạng thái kết thúc)  return true; //tìm kiếm thành công
        khởi tạo danh sách open chỉ chứa đỉnh bắt đầu;
        khởi tạo danh sách close rỗng;
        while(true){
                if (open rỗng)  return  false;
                chọn và loại đỉnh u ở đầu danh sách  open;
                thêm u vào danh sách close;
                khởi tạo danh sách L rỗng;
```

```
            for (mỗi đỉnh v kề u) {
                if (v không có trong close và v không có trong open) {
                    father(v) =u;
                    if (v là đỉnh kết thúc) return true;
                    chèn v vào L sao cho L được sắp tăng theo giá trị hàm đánh giá;
                }
            }
            Gắn danh sách Lvào đầu danh sách open;
        }
}
```

2.2.3Thuật toán A*

- Thuật toán A* sử dụng thuật toán tìm kiếm tốt nhất đầutiên (Best_First_Search)vớihàm đánh giá f(u)=g(u)+h(u), với g(u) là chi phí nhỏ nhất để đi từ đỉnh ban đầu tới đỉnh u, h(u) là chi phí dự kiến để biến đổi đỉnh u tới đỉnh đích.

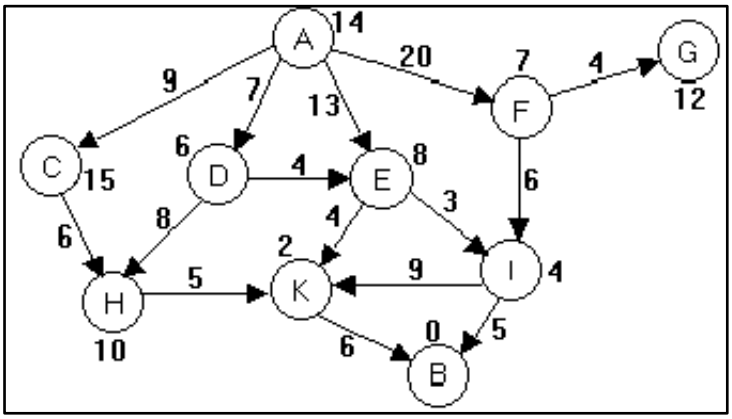
- Khi phát triển một đỉnh được đỉnh con u có trạng thái giống với trạng thái của một đỉnh v đã có trong open thì tính giá trị đỉnh con u. Nếu f(u)<f(v) thì loại bỏ v khỏi open và thêm u vào open.

- Nếu hàm h(u) <=chi phí thực tế để biến đổi u về đích (đặc biệt, h(u) = 0 với mọi u) thì thuật toán A* là thuật toán tối ưu, nghĩa là tìm được trạng thái kết thúc với chi phí nhỏ nhất.

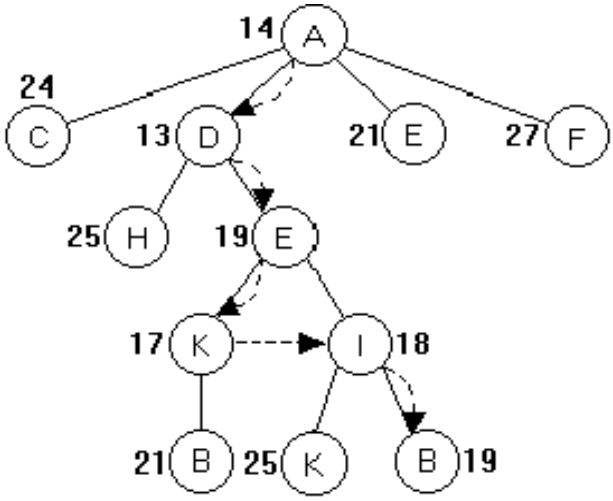
✓ **Ví dụ:**

Cho không gian trang thái như hình 2.9. Các số ghi cạnh các cung là độ dài cung đó, các số ghi cạnh các đỉnh là giá trị của hàm h. Tìm dãy các phép biến đổi để biến đổi A thành B.

Ta có cây tìm kiếm A* như hình 2.10. Quá trình thuật toán thực hiện như hình 2.11



Hình 2.9: không gian trạng thái với hàm đánh giá.



Hình 2.10: Cây tìm kiếm A*

Bước lặp	Open	Close
Khởi tạo	(A,14,null) // A có giá trị f=14, và có cha là null	Rỗng
1	(D,13,A),(E,21,A),(C,24,A),(F,27,A)	A
2	(E,19,D),(C,24,A),(H,25,D),(F,27,A)	A,D
3	(K,17,E),(I,18,E),(C,24,A),(H,25,D),(F,27,A)	A,D,E
4	(I,18,E), (B,21,K), (C,24,A),(H,25,D),(F,27,A)	A,D,E,K
5	(B,19,I), (C,24,A), (K,25,I), (H,25,D),(F,27,A)	A,D,E,K,I
6	Lấy được đỉnh B trong open => ngừng	A,D,E,K,I,B

Hình 2.11: các bước A* thực hiện với chi phí tìm kiếm = 19, chuỗi biến đổi : A->D->E->I->B

Giải thích:

Đầu tiên, phát triển A sinh ra C, D, E, F. Ta có: g(C)=9, h(C)=15, f(C)=g(C) + h(C)=9+15=24; tính tương tự f(D)=13, f(E)=21, f(F)=27.Như vậy đỉnh tốt nhất là D vì f(D) nhỏ nhất.

Phát triển D, ta được H và E mới. Ta có: g(H)=g(D)+Độ dài cung (D, H) = 7 + 8 = 15, f(H) = 15 + 10 = 25. g(E) = g(D) + Độ dài cung (D, E) = 7 + 4 = 11. Vậy đỉnh E mới có f(E) = g(E) + h(E) = 11 + 8 = 19< f(E cũ) => bỏ đỉnh E cũ thay bằng E mới (nếu E mới có giá trị >= giá trị E cũ thì giữ E cũ, bỏ E mới). Như vậy đỉnh tốt nhất là E.

Phát triển E được K và I. Tiếp tục cho tới khi đỉnh được chọn để phát triển là đỉnh kết thúc B thì ngừng và độ dài đường đi ngắn nhất tới B là g(B) = 19.

✓ **Thuật toán**

```
bool  A_Star_Search {
    if (trạng thái ban đầu là trạng thái kết thúc)  return true; //tìm kiếm thành công
    khởi tạo danh sách open chỉ chứa đỉnh bắt đầucó trạng thái ban đầu u0 và g(u0)=0;
    khởi tạo danh sách close rỗng;
    while (true) {
        if (open rỗng) return  false;
        chọn và loại đỉnh u ở đầu danh sách  open;
        if (u là trạng thái kết thúc) return true;
        thêm u vào danh sách close;
        for (mỗi đỉnh v kề u) {
            father(v) =u;
            tính g(v)= g(u) + a(u,v); //a(u,v) là trọng số cung (u,v)
            tính f(v)= g(v) + h(v);
            if (v không có trong close và v không có trong open) {
                chèn v vào open sao cho open được sắp tăng theo giá trị f;
            }else{
                if (v có trong open và f(v)<f (v cũ)) {//v cũ là v trong open
                    hủy v cũ;
                    chèn v vào open sao cho open được sắp tăng theo giá trị f;
                }
            }
        }
    }
}
```

2.3 Tóm tắt

- **Tìm kiếm không có thông tin**:
  - ✓ Breadth First Search,
  - ✓ Depth First Search,
  - ✓ Depth Deepening Search
- **Tìm kiếm có thông tin:**
  - ✓ best-first search
  - ✓ hill-climbing search
  - ✓ A* search**.**

2.4Bài tập

Sử dụng các thuật toán tìm kiếm, cài đặt các bài toán sau:

- ✓ tìm đường đi trong mê cung.
- ✓ trò chơi 8 số.
- ✓ triệu phú và kẻ cướp.

- Hết -