Chapter 2.
Search methods in artificial intelligence

Giới thiệu các chiến lược thiết kế thuật toán

Các chiến lược tìm kiếm lời giải chính xác

Thuật toán đệ qui

Thuật toán chia để trị

Thuật toán quay lui

Thuật toán nhánh và cận

Thuật toán quy hoạch động

Ưu, nhược điểm của các chiến lược tìm kiếm lời giải chính xác

Ưu điểm:

Nhược điểm:

Các chiến lược tìm kiếm lời giải gần đúng

Heurisric

Metaheuristic

Greedy algorithms

Thuật toán tham lam

- Tham lam (greedy) là một trong số các phương pháp giải bài toán tối ưu;
- Các thuật toán tham lam dựa vào sự đánh giá tối ưu cục bộ địa phương (local optimum) để đưa ra quyết định tức thì tại mỗi bước lựa chọn, với hy vọng cuối cùng sẽ tìm ra được phương án tối ưu tổng thể (global optimum).
- Thuật toán tham lam không chắc tìm được phương án tối ưu.

- Trong trường hợp thuật toán tham lam tìm ra lời giải đúng thì trong tài liệu này chưa tập trung vào việc chứng minh tính đúng đắn của thuật toán đó.
- Trong trường hợp thuật toán tham lam không tìm ra đúng phương án tối ưu, chúng ta thường thu được một phương án khả dĩ chấp nhận được.
- Với một bài toán có nhiều thuật toán để giải quyết, thông thường thuật toán tham lam có thời gian tính tốt hơn hẳn so với các thuật toán gần đúng khác; chẳng hạn như thuật toán metaheuristic.

Lược đồ thuật toán tham lam

```
void Greedy(A,S)//A là tập các ứng cử viên, S là tập nghiệm
{
        S=ϕ;
while (A ≠ ϕ)
        {
        x=select(A); //chọn phần tử tốt nhất trong A
        A=A - {x};
        if (S ∪ {x} chấp nhận được)
                S= S ∪ {x};
        }
}
```

VD1.Bài toán người bán hàng (Traveling Salesman Problem-TSP)

- Có n thành phố (được đánh số từ 1 đến n), một người bán hàng xuất phát từ một thành phố, muốn đi qua tất cả các thành phố khác, mỗi thành phố một lần rồi quay về thành phố xuất phát. Giả thiết biết được chi phí đi từ thành phố i đến thành phố j là c_{ij}.
- Hãy tìm một hành trình cho người bán hàng sao cho tổng chi phí theo hành trình này là thấp nhất.
- TSP là bài toán thuộc lớp bài toán NP-khó.

Traveling/ Travelling đều đúng

Bài toán người bán hàng có thể được mô hình hoá như một đồ thị vô hướng có trọng số, trong đó mỗi thành phố là một đỉnh của đồ thị còn đường đi giữa các thành phố là mỗi cạnh. Khoảng cách giữa hai thành phố là độ dài cạnh. Đây là vấn đề cực tiểu hoá với điểm đầu và điểm cuối là cùng một đỉnh sau khi thăm hết các đỉnh còn lại đúng một lần. Mô hình này thường là một đồ thị đầy đủ (giữa mỗi cặp đỉnh đều có cạnh). Nếu không có đường giữa hai thành phố thì có thể thêm một cạnh với độ dài đủ lớn vào đồ thị mà không ảnh hưởng đến kết quả tối ưu sau cùng.

Đối xứng và bất đối xứng

Trong bài toán TSP đối xứng, khoảng cách giữa hai thành phố là không đổi dù đi theo chiều nào. Như vậy đồ thị trong bài toán này là đồ thị vô hướng. Việc đối xứng này làm giảm đi một nửa số lời giải có thể. Trong khi đó, với bài toán TSP bất đối xứng thì đường đi giữa hai thành phố có thể chỉ một chiều hoặc có độ dài khác nhau giữa mỗi chiều, tạo nên đồ thị có hướng. Các tai nạn giao thông, đường một chiều hay phí hàng không giữa các thành phố với phí điểm xuất phát và điểm đến khác nhau là những ví dụ về sự bất đối xứng.

Thuật toán GTS1 (Greedy Traveling Saleman)

Input: n là số thành phố, u là đỉnh xuất phát u và c ,là ma trận chi phí.

Output: tour (thứ tự các thành phố đi qua; tính cả thanh phố xuất phát ở cuối của hành trình).

cost – chí phí ứng với tour tìm được

```
v=u;
tour={u};
cost=0;
for i=1 to n
{
        đặt w là thành phố kề sau thành phố v.
        tour=tour + {w};
        cost=cost+c[v,w]
        v=w;
}
tour=tour + {u};
cost=cost+c[v,u]
```

Ví dụ

Cho đồ thị có ma trận chi phí như sau:

∞	20	42	31	6	24
10	∞	17	6	35	18
25	5	∞	27	14	9
12	9	24	∞	30	12
14	7	21	15	∞	38
40	15	16	5	20	∞

Sử dụng thuật toán GTS1 để tìm hành trình bắt đầu tại các đỉnh $v_1=1$; $v_2=3$; $v_3=4$; $v_4=5$

Hướng dẫn giải:

GTS1(v_1): $1 \rightarrow 5 \rightarrow 2 \rightarrow 4 \rightarrow 6 \rightarrow 3 \rightarrow 1$

$cost(v_1) = 6 + 7 + 6 + 12 + 16 + 25 = 72.$

GTS1(v_2):$3 \rightarrow 2 \rightarrow 4 \rightarrow 1 \rightarrow 5 \rightarrow 6 \rightarrow 3$

$cost\ (v_2) = 5 + 6 + 12 + 6 + 38 + 16 = 83.$

GTS1(v_3): $4 \rightarrow 2 \rightarrow 1 \rightarrow 5 \rightarrow 3 \rightarrow 6 \rightarrow 4$

$cost\ (v_3) = 9 + 10 + 6 + 21 + 9 + 5 = 60.$

GTS1(v_4): $5 \rightarrow 2 \rightarrow 4 \rightarrow 1 \rightarrow 6 \rightarrow 3 \rightarrow 5$

$cost\ (v_4) = 7 + 6 + 12 + 24 + 16 + 14 = 79.$

Thuật toán GTS2 (Greedy Traveling Saleman)

Input: n, c, p,v_i (i = 1..p) // v_i là các thành phố cho trước hoặc cũng có thể được chọn ngẫu nhiên trong tập 1..p

Output: besttour, bestcost

```
bestcost=0;
besttour={}
for i=1 to p
{
GTS1(vk); // suy ra được tour(vk) và  cost(vk)
If cost(vk)<bestcost
        {
                bestcost=cost(vk)
                besttour=tour(vk)
        }
}
```

- Sử dụng thuật toán GTS2 để tìm hành trình tốt nhất với p=4 (v1=2; v2=3; v3=5; v4=6; bài toán dạng này có thể không cho cụ thể các v_i, lúc đó vi có thể được chọn ngẫu nhiên).

Hướng dẫn giải:

Áp dụng thuật toán GTS1 như trên để tính

GTS1(v_1) : $2 \to 4 \to 1 \to 5 \to 3 \to 6 \to 2$

cost(v_1)=.6+12+6+21+9+15=69

GTS1(v_2): $3 \to 2 \to 4 \to 1 \to 5 \to 6 \to 3$

cost (v_2) =5 + 6 + 12 + 6 +38 + 16 = 83.

GTS1(v3):$5 \to 2 \to 4 \to 1 \to 6 \to 3 \to 5$

cost (v_3)=7 + 6 + 12 + 24 +16 + 14 = 79.

GTS1(v4):$6 \to 4 \to 2 \to 1 \to 5 \to 3 \to 6$

cost (v_4) =5 + 9 + 10 + 6 +21 + 9 = 60.

Kết luận: Hành trình tốt nhất có chi phí là 60 với chi tiết tour như sau:

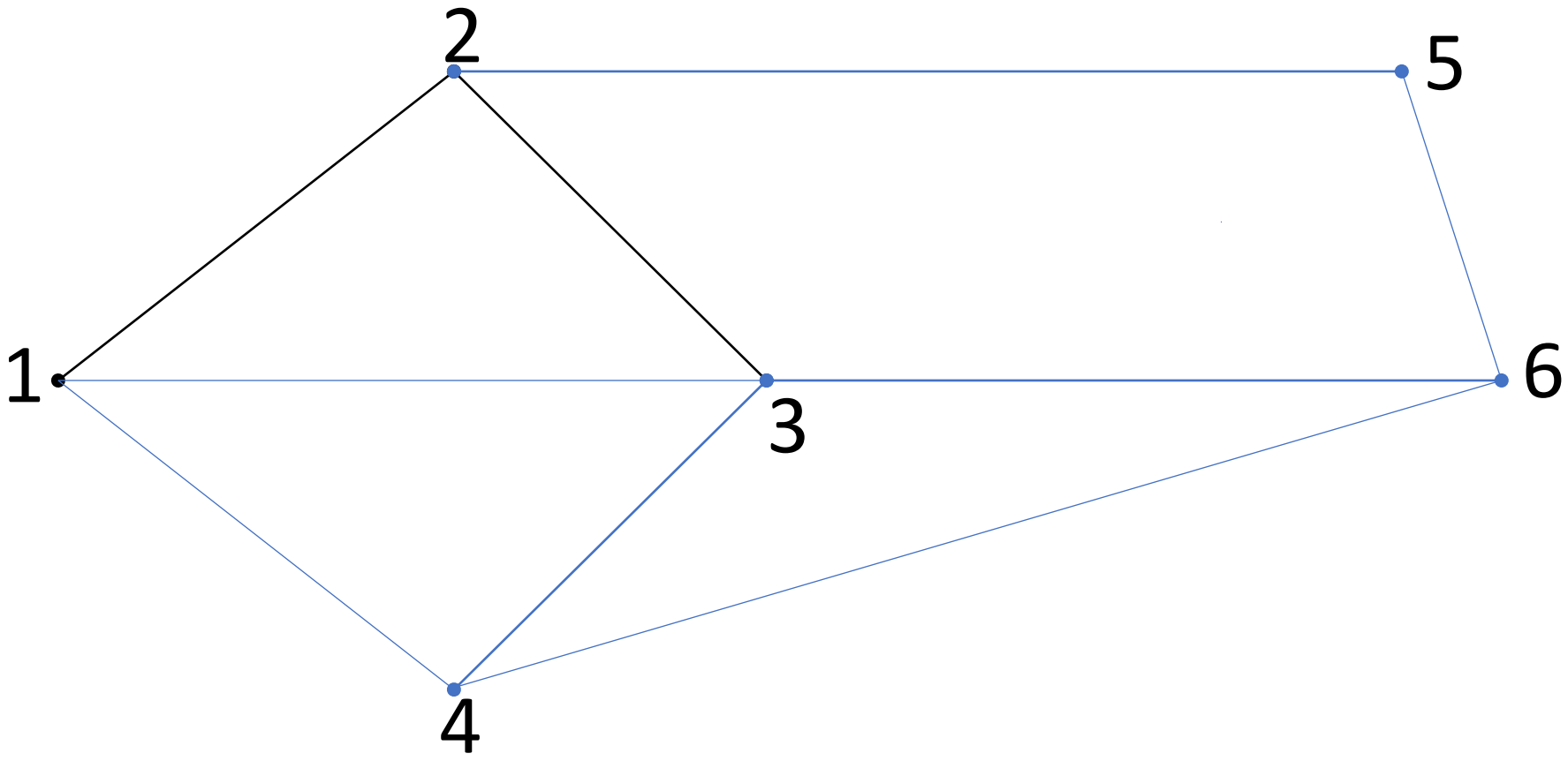
$6 \to 4 \to 2 \to 1 \to 5 \to 3 \to 6$

VD2. Coloring problem(Bài toán tô màu)

- Cho *n* thành phố, hãy tô màu các thành phố này sao cho không có bất kỳ hai thành phố nào kề nhau được tô cùng một màu và số màu được tô là ít nhất có thể.
- Dữ liệu vào được lưu trên một trận vuông c_{ij}.
- Nếu c_{ij} = 1 thì hai thành phố i,j là kề nhau, c_{ij}=0 thì hai thành phố i,j không kề nhau.
- Bài toán tô màu là bài toán thuộc lớp bài toán NP-Đầy đủ.
- Hãy đề xuất thuật toán tham lam giải bài toán trên.

- **Tô màu đỉnh** (tiếng Anh: *vertex coloring*) là gán cho mỗi đỉnh của đồ thị một màu nào đó sao cho không có hai đỉnh nào liền kề lại trùng màu nhau;
- **Tô màu cạnh** (tiếng Anh: *edge coloring*) là gán cho mỗi cạnh của đồ thị một màu nào đó sao cho sao cho không có 2 cạnh nào trùng màu;
- **Tô màu miền** (tiếng Anh: *face coloring*) là gán cho mỗi miền của đồ thị phẳng một màu sao cho không có 2 miền có chung đường biên lại cùng màu.
- **Sắc số** (tiếng Anh: *chromatic number*) của một đồ thị là số màu ít nhất để tô các đỉnh. Sắc số của đồ thị G được ký hiệu là $\chi(G)$.

Hãy tô màu đồ thị sau:



Thuật toán tô màu (D. J. A. Welsh, M. B. Powell)

- Bước 1:Tìm bậc của các đỉnh.
- Bước 2:Sắp xếp các đỉnh theo bậc giảm dần.
- Bước 3:Dùng màu thứ nhất tô cho đỉnh có bậc cao nhất và các đỉnh khác có thể tô còn lại; dùng màu thứ hai tô cho đỉnh có bậc cao thứ nhất (còn lại) và các đỉnh khác có thể tô còn lại; và cứ như thế... cho đến khi tất cả các đỉnh của đồ thị đều được tô màu.

Welsh - Powell graph coloring Algorithm

1. Find the degree of each vertex
2. List the vertices in order of descending degrees.
3. Colour the first vertex with color 1.
4. Move down the list and color all the vertices not connected to the coloured vertex, with the same color.
5. Repeat step 4 on all uncolored vertices with a new color, in descending order of degrees until all the vertices are coloured.

Tham khảo một thuật toán khác giải bài toán tô màu (thuật toán DSATUR - Degree of Saturation)

- *Tính bậc của tất cả các đỉnh*

  while (còn đỉnh có bậc lớn hơn 0)

  {

  - *Tìm đỉnh(chưa được tô) có bậc lớn nhất. Chẳng hạn đó là đỉnh i_0.*
  - *Tìm màu để tô đỉnh i_0 là màu nhỏ nhất trong danh sách các màu còn lại có thể tô cho đỉnh i_0. Chẳng hạn đó là màu j.*
  - *Ngăn cấm việc tô màu j cho các đỉnh kề đỉnh i_0.*
  - *Tô màu đỉnh i_0 là j.*
  - *Gán bậc của đỉnh được tô bằng 0, các đỉnh kề với đỉnh được tô có bậc giảm đi 1 đơn vị.*

}

Sau khi kết thúc vòng lập trên có thể còn đỉnh chưa được tô nhưng tất cả các đỉnh lúc này đều đã có bậc bằng 0 – nghĩa là không thể hạ bậc được nữa. Khi đó màu của các đỉnh chưa được tô chính là màu nhỏ nhất hợp lệ trong danh sách màu của đỉnh đó.

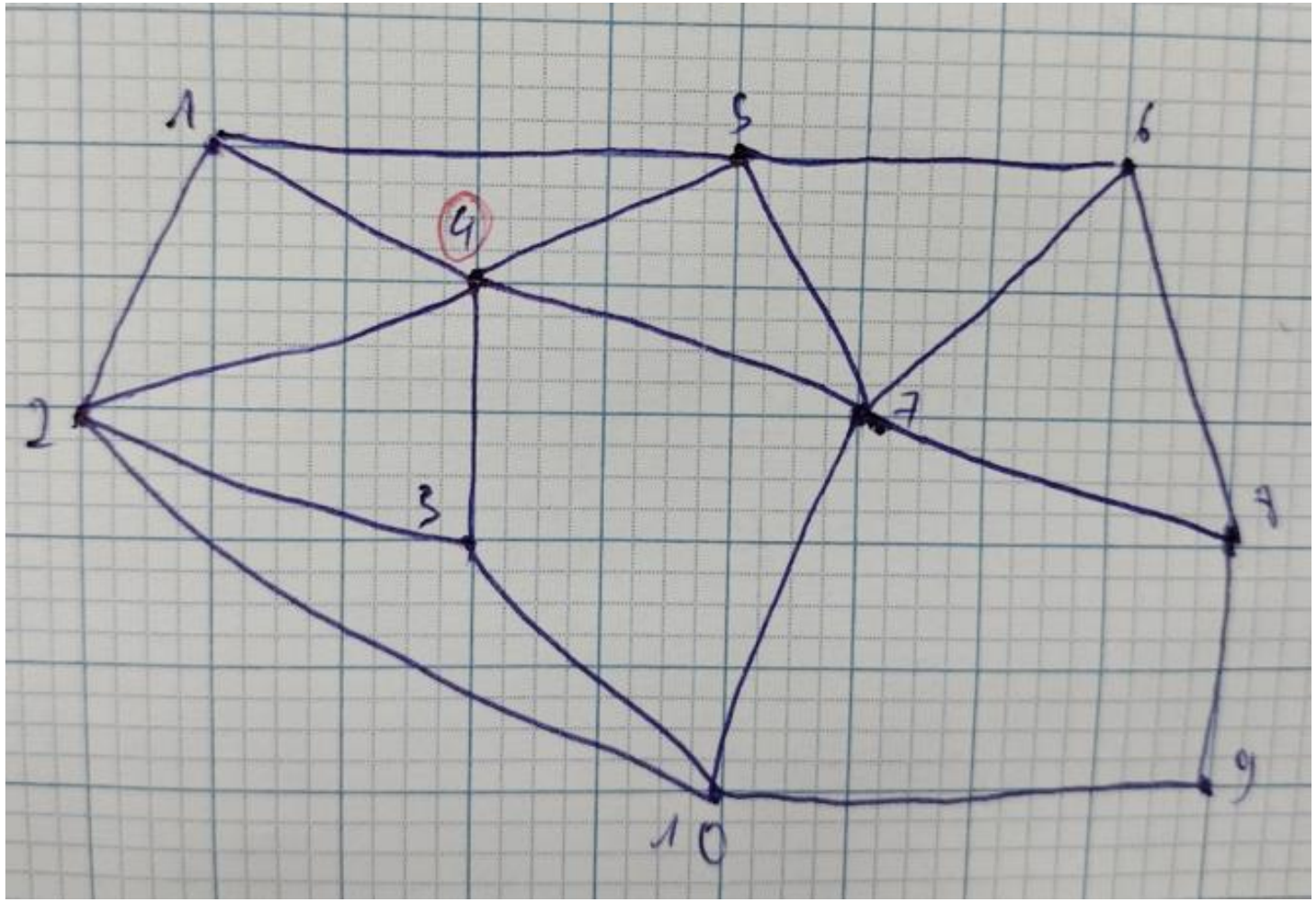
DSatur Algorithm for Graph Coloring - GeeksforGeeks

DSatur algorithm

- Let **G** be a graph with **n** vertices and **m** edges. In addition, assume that we will use the colour labels **0, 1, 2, …, n-1**. (More than **n** colours are never required in a solution). The **DSatur algorithm** operates as follows

1. Let **v** be the uncoloured vertex in **G** with the largest saturation degree. In cases of ties, choose the vertex among these with the largest degree in the subgraph induced by the uncoloured vertices. Further ties can be broken arbitrarily.
2. Assign **v** to colour **i**, where **i** is the smallest integer from the set **{0, 1, 2, …, n}** that is not currently assigned to any neighbour of **v**.
3. If there remain uncoloured vertices, repeat all steps again, otherwise, end at this step.

Hai thuật toán tô màu trên là không tối ưu!
Sau đây là một phản ví dụ



VD3a. Bài toán phân việc (Assignment problem)

- Có n công việc và n thợ; mỗi thợ đều có khả năng thực hiện tất cả các công việc, w_{ij} là hiệu quả phân công thợ i làm công việc j, ($i, j = 1, 2,..., n$).
- Tìm cách phân công thợ thực hiện các công việc sao cho mỗi thợ chỉ thực hiện một việc và mỗi việc chỉ do một thợ thực hiện, đồng thời tổng hiệu quả thực hiện các công việc là lớn nhất.
- Mỗi thợ không nhất thiết phải có khả năng thực hiện tất cả các công việc và số thợ và số công việc không nhất thiết phải bằng nhau.

Ví dụ. Sử dụng thuật toán tham lam giải bài toán phân việc khi n=6.

6

4 9 5 1 4 9

3 1 3 7 8 6

5 7 3 4 1 9

8 8 3 7 6 6

5 4 3 8 9 7

6 3 1 4 2 5

VD3b. Bài toán phân việc với các máy đồng nhất

- Giả sử có m máy NHƯ NHAU được ký hiệu từ $P_1,\ldots,P_m$. Có n công việc $J_1,\ldots,J_n$ cần được thực hiện. Các công việc có thể được thực hiện đồng thời và bất kỳ công việc nào cũng có thể chạy trên một máy nào đó. Mỗi lần máy được cho thực hiện một công việc nó sẽ làm cho tới khi hoàn chỉnh. Công việc J_i có thời gian thực hiện là T_i .
- Hãy tìm cách phân công sao cho tất cả các công việc được hoàn thành trong thời gian sớm nhất.

Thuật toán 1

- Giả sử L là một thứ tự các công việc cần được thực hiện.
- Lặp lại các công việc sau cho đến khi tất cả các công việc đều được phân công:
  - Nếu có máy nào rảnh thì nạp **công** việc kế tiếp trong danh sách *L* vào
  - Nếu có 2 hay nhiều máy cùng rảnh tại một thời điểm thì máy với chỉ số thấp sẽ được phân cho công việc.

Ví dụ:

Giả sử có 3 máy P_1,P_2,P_3 và 6 công việc J_1,J_2,J_3,J_4,J_5 J_6, với

$T_i=(2,5,8,1,5,1)$

$L= (J_2,J_5,J_1,J_4,J_6,J_3)$

Giải:

P1: $J_2(5)$,

P2: $J_5(5)$,

P3: $J_1(2)$, $J_4(1)$,$J_6(1)$,$J_3(8)$

Vậy thời gian hoàn thành tất cả các công việc là 12.

Thuật toán 2

- Gọi L^* là phương án mà các công việc được sắp theo thứ tự thời gian giảm dần, tức $L^*=\{J_3,J_2,J_5,J_1,J_4,J_6\}$
- Áp dụng như thuật toán 1 thì thời gian hoàn thành tất cả các công việc là 8.

Câu hỏi: Thuật toán 2 có tối ưu không ?

Trả lời: Thuật toán 2 trên không tối ưu!

Xem phản ví dụ sau:

Cho hai máy P_1,P_2 và 5 công việc J_1,J_2,J_3,J_4,J_5. Thời gian thực hiện các công việc đã được sắp giảm là 3,3,2,2,2.

Khi đó cách phân công công việc là:

P_1: 3 2 2

P_2: 3 2

Thời gian hoàn thành là 7 (không tối ưu) do thời gian hoàn thành tối ưu là 6:

3 3

2 2 2

VD3c. Sequencing Problem N jobs on 2 Machines (Bài toán gia công trên 2 máy)

- Mỗi một chi tiết $D_1, D_2, ... D_n$ cần phải được lần lượt gia công trên 2 máy A,B. Thời gian gia công chi tiết D_i trên máy A là a_i trên máy B là b_i (i=1,2..n).
- Hãy tìm lịch (trình tự gia công) các chi tiết trên hai máy sao cho việc hoàn thành gia công tất cả các chi tiết là sớm nhất.

Thuật toán Johnson (Bài toán gia công trên 2 máy)

Bước 1: Chia các chi tiết thành 2 nhóm:

Nhóm N_1 gồm các chi tiết D_i thoả mãn $a_i <= b_i$

Nhóm N_2 gồm các chi tiết D_i thoả mãn $a_i > b_i$

Bước 2:

Sắp xếp các chi tiết trong nhóm N_1 theo chiều tăng của các a_i và sắp xếp các chi tiết trong nhóm N_2 theo chiều giảm của các b_i

Bước 3:

Nối các chi tiết trong N_2 vào cuối các chi tiết trong nhóm N_1 thì dãy thu được (đọc từ trái sang phải) sẽ là lịch gia công của các chi tiết trên các máy A,B.

Ví dụ về thuật Johnson

- Có 10 công việc được đánh số thứ tự là: 1,2,3,4,5,6,7,8,9,10
- Thời gian hoàn thành các công việc trên máy A và máy B lần lượt là A_i (i=1..10), B_i (i=1..10):

Công việc	1	2	3	4	5	6	7	8	9	10
A_i	6	4	8	3	4	6	9	5	1	7
B_i	2	10	9	1	8	5	3	6	7	8

Ví dụ khác: Tính thời gian 2 máy hoàn thành tất cả các công việc:

6 5 3 9 4

7 2 4 3 6

VD3d.

Một dịch vụ in ấn luận văn tốt nghiệp, có 3 nhân viên đánh máy và một quản lý. Dịch vụ nhận được yêu cầu đánh máy luận văn của sinh viên làm luận văn tốt nghiệp như sau:

Luận văn	L1	L2	L3	L4	L5	L6	L7	L8	L9	L10	L11	L12
Số trang	240	180	100	140	100	160	60	140	140	200	80	100

Giả sử trong một giờ thì một nhân viên đánh máy được 10 trang (giả thiết các nhân viên có công suất đánh máy bằng nhau).

a. Phân chia các luận văn cho 03 nhân viên đánh máy sao cho thời gian hoàn thành việc đánh máy các luận văn trên là sớm nhất.
b. Trong trường hợp người quản lý cũng tham gia đánh máy, nhưng công suất của người quản lý chỉ bằng $^{2}/_{3}$ công suất của nhân viên. Tìm cách chia các luận văn cho 3 nhân viên và người quản lý, sao cho thời gian hoàn thành việc đánh máy các luận văn là sớm nhất.

Hướng dẫn giải

a.Theo bài ra, thời gian để các nhân viên đánh máy xong các luận văn tốt nghiệp là:

Luận án	L1	L2	L3	L4	L5	L6	L7	L8	L9	L10	L11	L12
Thời gian	24	18	10	14	10	16	6	14	14	20	8	10

Sắp xếp các luận văn theo thời gian đánh máy giảm

Luận án	L1	L10	L2	L6	L4	L8	L9	L3	L5	L12	L11	L7
Thời gian	24	20	18	16	14	14	14	10	10	10	8	6

Lịch đánh máy các luận văn của nhân viên:

Nhân viên 1:L1(24), L8(14), L5(10),L11(8)

Nhân viên 2:L10(20), L4(14), L9(14), L7(6)

Nhân viên 3:L2(18), L6(16), L3(10), L12(10)

Theo lịch trên, thời gian để các nhân viên 1,2,3 hoàn thành công việc của mình lần lượt là:

56,54,54.

Vậy thời gian hoàn thành việc đánh máy cho các luận văn là 56.

b.Khi người quản lý tham gia, và có công suất đánh máy bằng 2/3 nhân viên. Ta có lịch sau;

Nhân viên 1:L1(24), L9(14), L11(8)

Nhân viên 2:L10(20), L8(14), L12(10)

Nhân viên 3:L2(18), L4(14),L5(10)

Quản lý:L6(24), L3(15), L7(9)

Thời gian để các nhân viên và người quản lý hoàn thành công việc của mình lần lượt là 46,44,42,48

Vậy thời gian hoàn thành việc đánh máy cho các luận văn khi có người quản lý tham gia là 48.

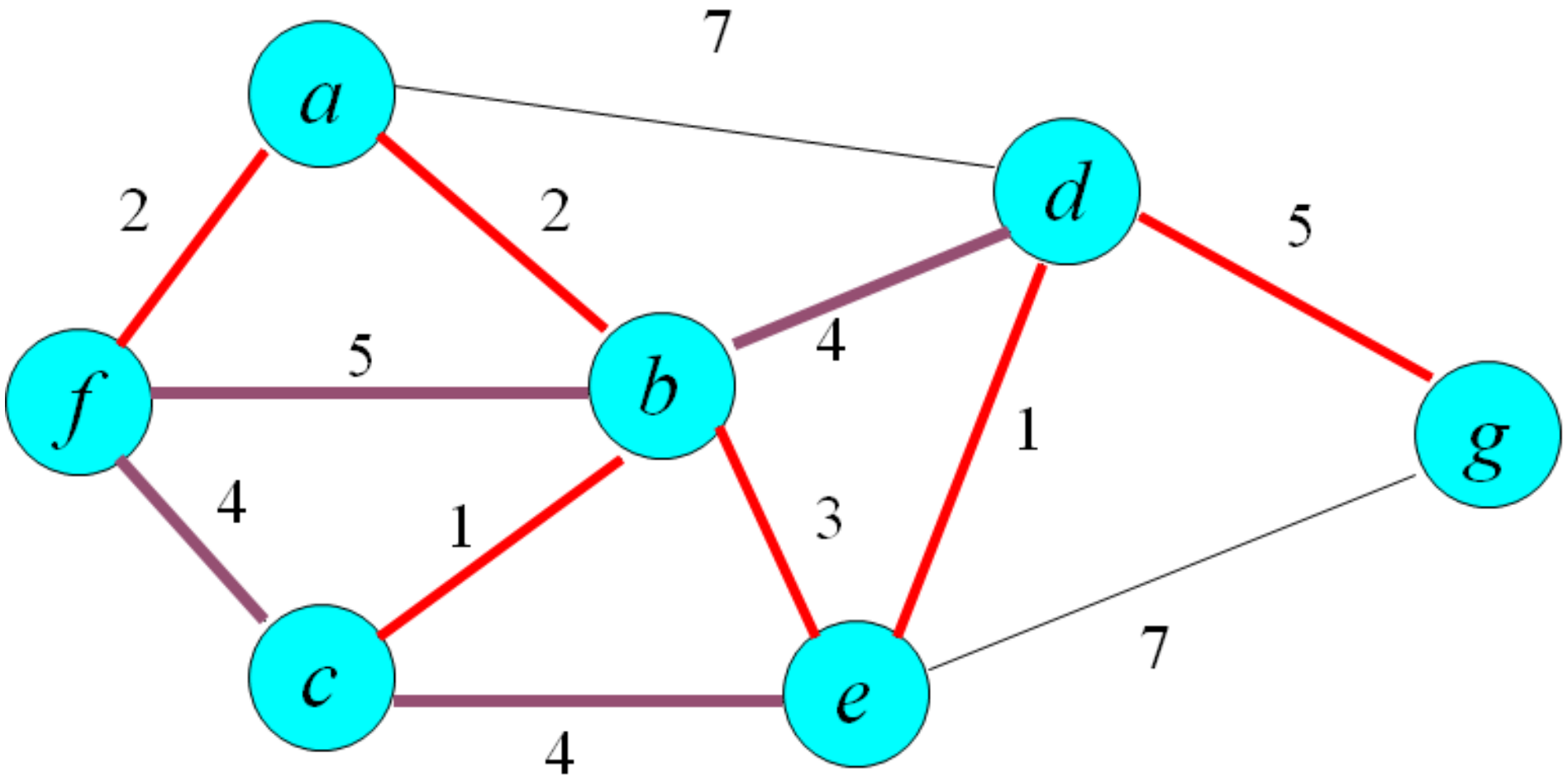
VD4. Minimum spanning tree problem-MST

Phát biểu Bài toán cây khung nhỏ nhất

Thuật toán tìm cây khung nhỏ nhất

- Thuật toán Kruskal
- Thuật toán Prim

Ví dụ



Độ dài của CKNN: 14

VD 5.(The problem of finding non-intersecting line segments) Bài toán tìm các đoạn thẳng không giao nhau

Bài toán: Tìm tập các đoạn thẳng không giao nhau

- **Đầu vào:** Cho họ n đoạn thẳng $C = \{(a_1, b_1), (a_2, b_2), ..., (a_n, b_n)\}$; các điểm đầu mút của đoạn thẳng nằm trên trục hoành.
- **Đầu ra:** Tập các đoạn thẳng không giao nhau và có số đoạn được chọn là nhiều nhất.

Greedy 1: Bắt đầu sớm thì chọn trước

- Bước 1: Sắp xếp các đoạn thẳng theo thứ tự tăng dần của đầu mút trái;
- Bước 2: Bắt đầu từ tập *S* bằng rỗng, ta lần lượt bổ sung các đoạn thẳng theo thứ tự đã được sắp vào *S* nếu nó không có điểm chung với bất cứ đoạn thẳng nào trong *S*.

Greedy 2: Đoạn thẳng ngắn nhất thì được lựa chọn trước

- Bước 1: Sắp xếp các đoạn thẳng theo thứ tự tăng dần của độ dài;
- Bước 2: Bắt đầu từ tập *S* bằng rỗng, ta lần lượt bổ sung các đoạn thẳng theo thứ tự đã được sắp vào *S* nếu nó không có điểm chung với bất cứ đoạn thẳng nào trong *S*.

Greedy 3. Kết thúc sớm thì lựa chọn trước

- Bước 1: Sắp xếp các đoạn thẳng theo thứ tự tăng dần của đầu mút phải;
- Bước 2: Bắt đầu từ tập *S* bằng rỗng, ta lần lượt bổ sung các đoạn thẳng theo thứ tự đã được sắp vào *S* nếu nó không có điểm chung với bất cứ đoạn thẳng nào trong *S*.

❖Hãy xác định độ phức tạp của mỗi thuật toán trên ?
❖Các thuật toán tham lam trên có tìm được lời giải tối ưu không ?

Giải bài toán sau bằng 3 thuật toán tham lam trên

12
3 7;
3 4;
5 7;
4 5;
-5 -1;
-2 0;
1 5;
-9 -6;
-8 -7;
2 6;
1 7;
-9 0;

VD6.

- Có n công việc được đánh số từ 1 đến n và có một máy để thực hiện. Biết p_i là **thời gian** cần thiết để hoàn thành công việc i, d_i là **thời điểm** cần hoàn thành công việc i.
- Máy bắt đầu hoạt động từ thời điểm 0. Mỗi công việc cần được thực hiện liên tục từ lúc bắt đầu cho đến khi kết thúc, không được phép ngắt quãng. Giả sử c_i là thời điểm hoàn thành công việc i. Khi đó, nếu $c_i > d_i$ ta nói công việc i hoàn thành trễ hạn, còn nếu $c_i <= d_i$ thì ta nói công việc i hoàn thành đúng hạn.
- Hãy tìm trình tự các công việc sao cho số công việc hoàn thành trễ hạn là ít nhất (hay số công việc hoàn thành đúng hạn là nhiều nhất).

Gợi ý thuật toán:

-Sắp xếp các công việc theo thứ tự p_i tăng dần.

-Tính thời gian thực tế hoàn thành mỗi công việc

-Từ đó biết được các công việc đúng hạn hoặc trễ hạn.

Ví dụ: n=12

(p_i,d_i)

(7; 35),

(9; 53);

(3; 10),

(8; 41),

(2; 8),

(3; 4),

(1; 1),

(5; 9),

(4; 15),

(2; 7),

(6; 27),

(10; 24)

CÁC THUẬT TOÁN DUYỆT ĐỒ THỊ (Graph Searching, Graph Traversal)

Các thuật toán duyệt đồ thị

- Duyệt đồ thị: Graph Searching hoặc Graph Traversal
  - Duyệt qua mỗi đỉnh và mỗi cạnh của đồ thị
- Ứng dụng:
  - Cần để khảo sát các tính chất của đồ thị
  - Là thành phần cơ bản của nhiều thuật toán trên đồ thị
- Hai thuật toán duyệt cơ bản:
  - Tìm kiếm theo chiều rộng (Breadth First Search – BFS)
  - Tìm kiếm theo chiều sâu (Depth First Search – DFS)

Tìm kiếm theo chiều rộng

Breadth-first Search (BFS)

Ý tưởng

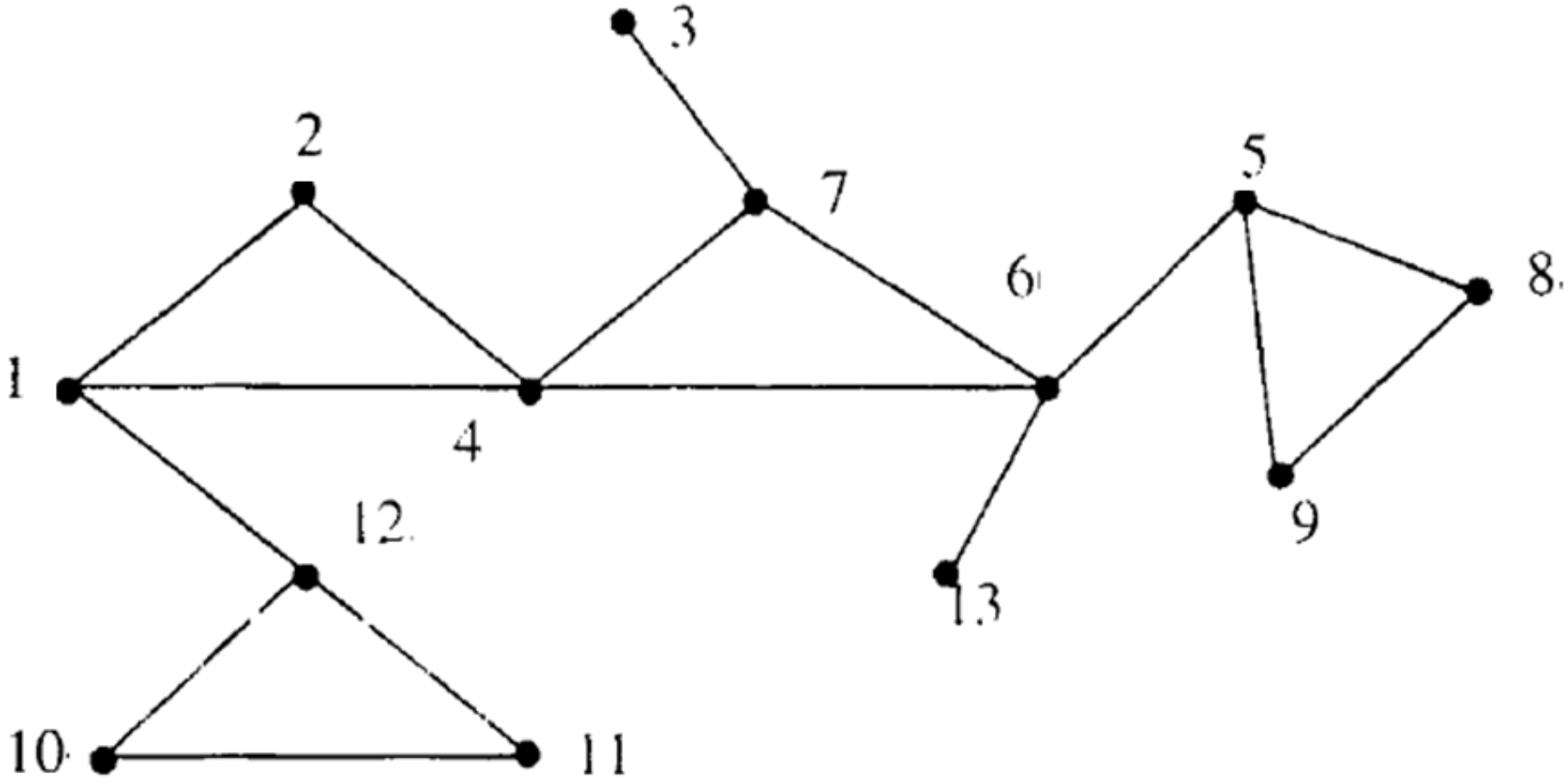
- Ta sẽ bắt đầu tìm kiếm từ một đỉnh v_o nào đó của đồ thị; sau đó duyệt qua tất cả các đỉnh kề với đỉnh v_o; chọn u là đỉnh đầu tiên kề với $v_o$và lặp lại quá trình tìm kiếm.
- Ở bước tổng quát, giả sử ta đang xét đỉnh v. thì ta duyệt hết tất cả các đỉnh kề với đỉnh v.
- Còn nếu như không còn đỉnh nào kề với v là chưa xét thì ta nói đỉnh này đã duyệt xong và quay trở lại tiếp tục tìm kiếm với đỉnh được duyệt sau khi duyệt đến đỉnh v.

```
procedure BFS(v);
(*  Tìm kiếm theo chiều rộng bắt đầu từ đỉnh v;
    Các biến Chuaxet, Ke  là biến toàn cục   *)
begin
     QUEUE:=∅;
     QUEUE ⇐ v; (* Kết nạp v vào QUEUE *)
    Chuaxet[v]:=false;
    while  QUEUE ≠ ∅  do
   begin
      p ⇐ QUEUE; (* Lấy p từ QUEUE *)
     Thăm_đỉnh(p);
    for  u ∈ Ke(p)  do
       if   Chuaxet[u]  then
      begin
         QUEUE ⇐ u;  Chuaxet[u]:=false;
     end;
   end;
```

Khi đó, Tìm kiếm theo chiều rộng trên đồ thị được thực hiện nhờ thuật toán sau:

```
BEGIN
    (*  Initialization *)
    for v ∈V do Chuaxet[v]:=true;
    for v ∈V do
        if Chuaxet[v] then BFS(v);
END.
```

Ví dụ (BFS)



Phân tích BFS

- Việc khởi tạo đòi hỏi $O(|V|)$.
- Vòng lặp duyệt
  - Mỗi đỉnh được nạp vào và loại ra khỏi hàng đợi một lần, mỗi thao tác đòi hỏi thời gian $O(1)$. Như vậy tổng thời gian làm việc với hàng đợi là $O(V)$.
  - Danh sách kề của mỗi đỉnh được duyệt qua đúng một lần. Tổng độ dài của tất cả các danh sách kề là $\Theta(|E|)$.
- Tổng cộng ta có thời gian tính của BFS(s) là $O(|V|+|E|)$,là tuyến tính theo kích thước của danh sách kề biểu diễn đồ thị.

CÁC ỨNG DỤNG CỦA BFS

- Sử dụng BFS để kiểm tra tính liên thông của đồ thị vô hướng:
  - Mỗi lần gọi đến BFS ở trong chương trình chính sẽ sinh ra một thành phần liên thông
- Xét sự tồn tại đường đi từ đỉnh *s* đến đỉnh *t*:
  - Thực hiện BFS(s).
  - Nếu π[t] = NIL thì không có đường đi, trái lại ta có đường đi
  - $t \leftarrow \pi[t] \leftarrow \pi[\pi[t]] \leftarrow \ldots \leftarrow s$

*Chú ý: BFS tìm được đường đi ngắn nhất theo số cạnh.

Tìm kiếm theo chiều sâu

Depth-first Search (DFS)

Ý tưởng của tìm kiếm theo chiều sâu

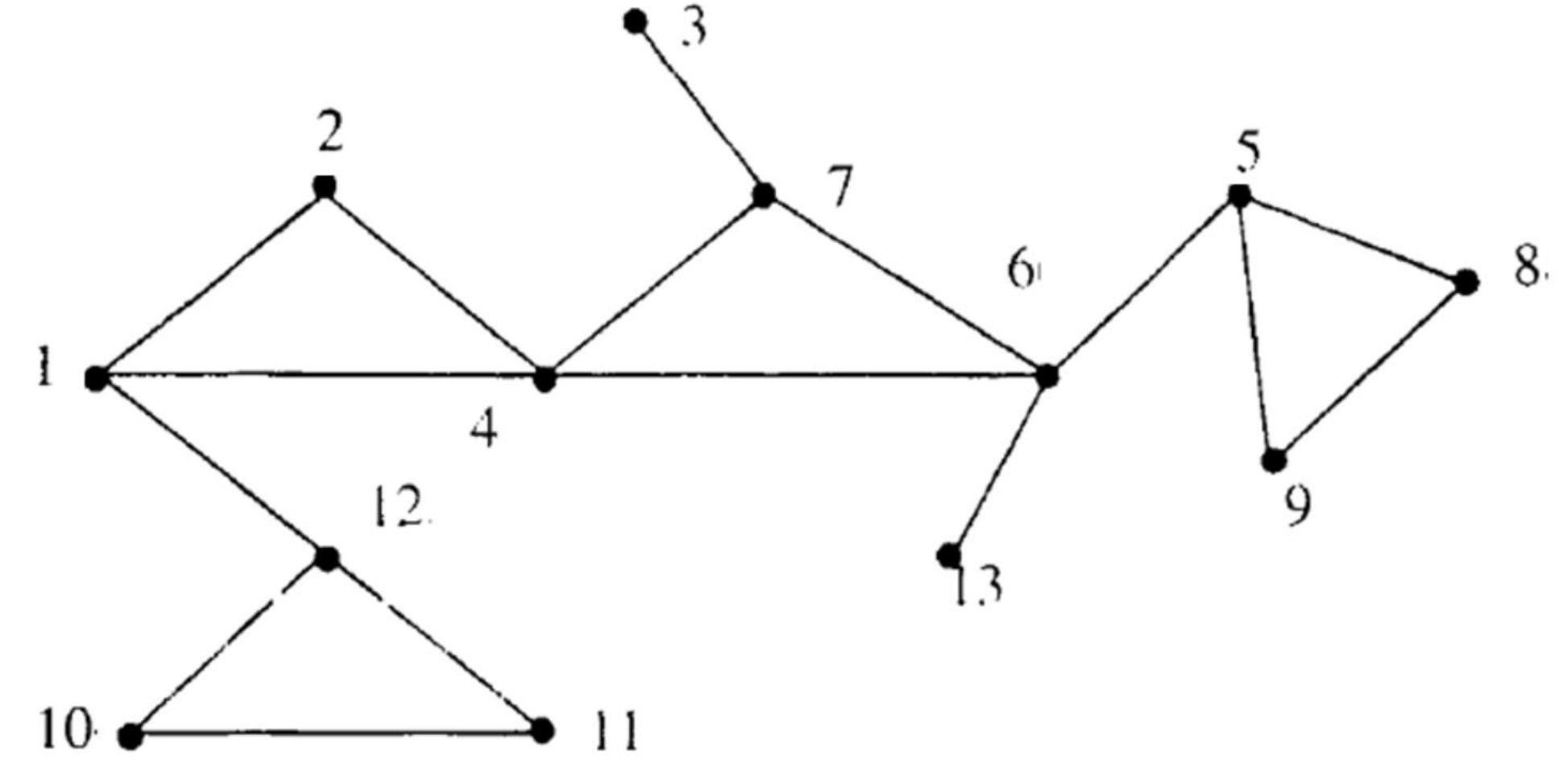
- Ta sẽ bắt đầu tìm kiếm từ một đỉnh v_o nào đó của đồ thị; sau đó chọn u là một đỉnh nào đó kề với v_o và lặp lại quá trình đối với u.
- Ở bước tổng quát, giả sử ta đang xét đỉnh v. Nếu như trong số các đỉnh kề với v ta tìm được đỉnh w là chưa được xét thì ta sẽ xét đỉnh này (w trở thành đỉnh đã xét) và bắt đầu từ nó ta sẽ tiếp tục quá trình tìm kiếm.
- Còn nếu như không còn đỉnh nào kề với v là chưa xét thì ta nói đỉnh này đã duyệt xong và quay trở lại tiếp tục tìm kiếm từ đỉnh mà trước đó ta đến được đỉnh v (nếu $v=v_o$ thì kết thúc quá trình tìm kiếm).

```
procedure DFS(v);
(*  Tìm kiếm theo chiều sâu bắt đầu từ đỉnh v;
    Các biến Chuaxet, Ke là biến toàn cục  *)
begin
      Thăm_đỉnh(v);
      Chuaxet[v]:=false;
      for  u ∈ Ke(v)  do
                if  Chuaxet[u]  then  DFS(u);
end;  (* đỉnh v là đã duyệt xong *)
```

Khi đó, Tìm kiếm theo chiều sâu trên đồ thị được thực hiện nhờ thuật toán sau:

```
BEGIN
   (*   Initialization  *)
   for  v ∈ V  do Chuaxet[v]:=true;
   for  v ∈ V  do
          if  Chuaxet[v] then  DFS(v);
END.
```

Ví dụ (DFS)



Phân tích thuật toán DFS

- Mỗi đỉnh được thăm đúng 1 lần, việc thăm mỗi đỉnh đòi hỏi chi phí thời gian $O(1)$, suy ra thao tác thăm đỉnh đòi hỏi thời gian $O(|V|)$.
- Vòng lặp trong DFS(u) thực hiện việc duyệt cạnh của đồ thị
  - Mỗi cạnh được duyệt qua đúng một lần nếu đồ thị là có hướng và 2 lần nếu đồ thị là vô hướng
  - Như vậy tổng số lần lặp là $O(|E|)$.
- Vậy, thuật toán có thời gian $O(|V|+|E|)$
- Đối với đồ thị, thuật toán có đánh giá như vậy gọi là *thuật toán thời gian tuyến tính*

DFS: Các loại cạnh

- DFS tạo ra một cách phân loại các cạnh của đồ thị đã cho:
  - *Cạnh của cây* (*Tree edge*): là cạnh mà theo đó từ một đỉnh ta đến thăm một đỉnh mới (cạnh đi vào đỉnh trắng)
  - *Cạnh ngược* (*Back edge*): đi từ con cháu (descendent) đến tổ tiên (ancestor)
  - *Cạnh tới* (*Forward edge*): đi từ tổ tiên đến con cháu
  - *Cạnh vòng* (*Cross edge*): cạnh nối hai đỉnh không có quan hệ họ hàng

Phân tích DFS

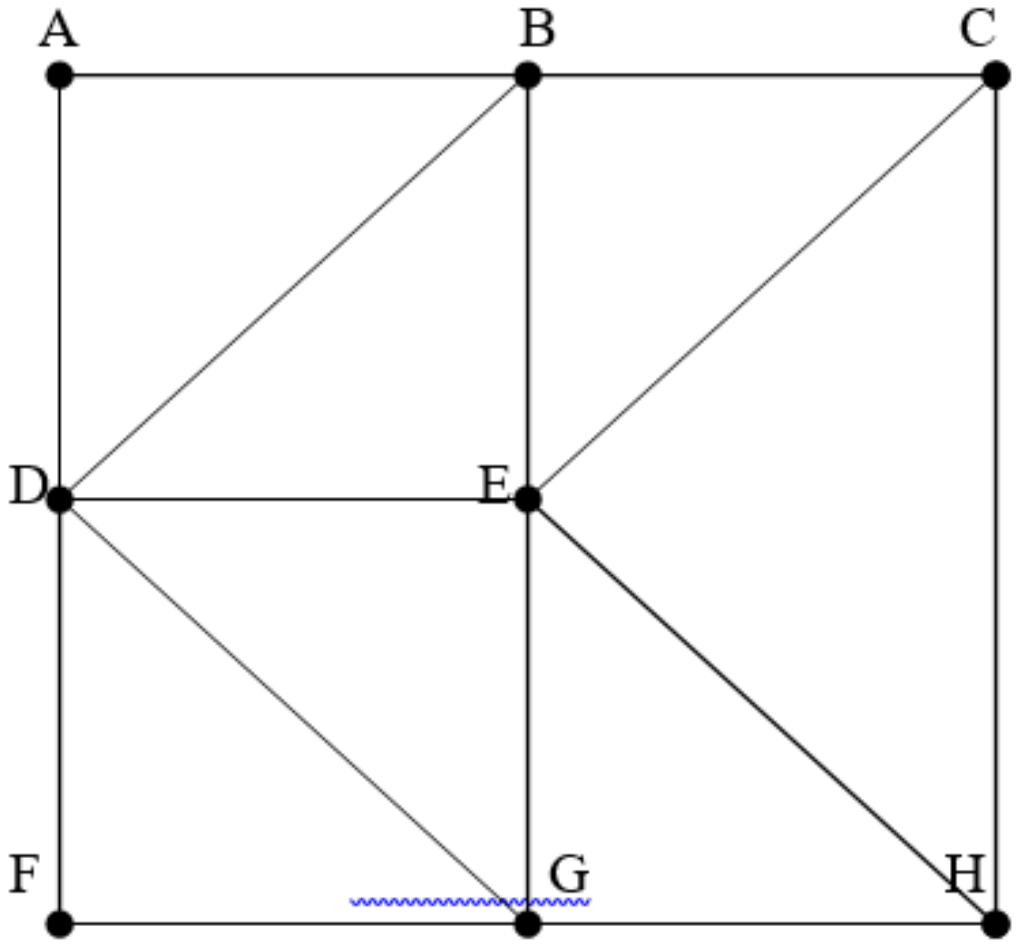
- Do đó thời gian của DFS là $\Theta(|V|+|E|)$.
- Thuật toán trên đồ thị có đánh giá thời gian như trên gọi là thuật toán thời gian tuyến tính

CÁC ỨNG DỤNG CỦA DFS

- Kiểm tra tính liên thông của đồ thị
- Tìm đường đi từ *s* đến *t*
- Tìm cây DFS
- Phát hiện chu trình

VÍ DỤ

Hãy liệt kê các đỉnh của đồ thị được duyệt theo phương pháp tìm kiếm theo chiều sâu, tìm kiếm theo chiều rộng. Tìm đường đi từ đỉnh A đến đỉnh H.



BEST-FIRST SEARCH ALGORITHM

(Tìm kiếm tốt nhất đầu tiên)

BEST-FIRST SEARCH ALGORITHM
(Tìm kiếm tốt nhất đầu tiên)

Tìm kiếm tốt nhất đầu tiên (best first search) là sự kết hợp của tìm kiếm theo chiều rộng với hàm đánh giá. Khác với phương pháp tìm kiếm theo chiều rộng, các nút không được phát triển lần lượt mà được lựa chọn dựa trên hàm đánh giá; đỉnh này có thể ở mức hiện tại hoặc ở các mức trên.

Bước 1: Khởi động

- Mọi đỉnh *n* là hàm *f, g, h* đều ẩn;
- Mở đỉnh đầu tiên S_o. Gán $g(S_o)=0$;
- Sử dụng tri thức bổ sung ước tính $h(S_o)$;
- Tính $f(S_o) = g(S_o) + h(S_o)$;

Bước 2: Lượng giá

- Chọn 1 đỉnh mở ứng với hàm *f* là min và gọi là đỉnh *N;;*

- Nếu *N* là đích → dừng (đường đi từ đỉnh ban đầu đến đỉnh *N* là ngắn nhất và bằng *g*(*N*));

- Nếu không tồn tại *N* thì cây biểu diễn vấn đề không có đường đi tới mục tiêu → dừng (bài toán không lời giải);

- Nếu tồn tại nhiều hơn 1 đỉnh *N* có cùng hàm f_{min} thì phải kiểm tra xem trong số đó có đỉnh nào là đích không;

    + Nếu có → dừng;

    + Nếu không → chọn ngẫu nhiên 1 trong các đỉnh đó và gọi đó là đỉnh N;

Bước 3: Phát triển

- Đóng đỉnh N và mở mọi đỉnh sau N;
- Mọi đỉnh S sau N, tính;

$$g(S) = g(N) + g(N - S);$$

- Dùng tri thức bổ sung để ước tính hàm $h(S)$;
- Tính $f(S) = g(S) + h(S)$;

Bước 4: Quay lui

- Quay lại bước 2;

Các dạng của thuật toán tìm kiếm tốt nhất đầu tiên

Một số dạng mở rộng của thuật toán tìm kiếm tốt nhất đầu tiên thường dùng như:

- Tìm kiếm tham lam (đã trình bày ở phần trước)
- Tìm kiếm trên cây A^{KT},
- Tìm kiếm A^*,
- Tìm kiếm leo đồi (sẽ được đề cập trong chương 5),...

Thuật toán tìm kiếm A^{KT}

- Mục này trình bày một phiên bản của thuật toán tìm kiếm trên cây đó là thuật toán A^{KT} (Algorithm knowledgeable For Tree);
- Ứng dụng thuật toán A^{KT} giải một số bài toán:
  - Bài toán TACI (bài toán 8-puzzle hay còn được gọi là bài toán đẩy số)
  - Tháp Hà Nội.

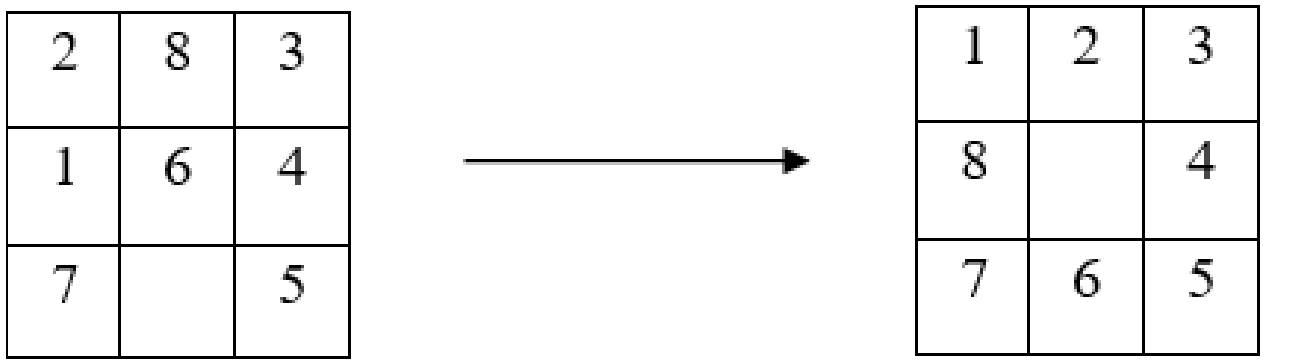
VD7.Bài toán TACI

- Có n^2-1 số mang các giá trị từ 1 tới n^2-1 được sắp xếp vào một lưới các ô vuông kích thước n x n. Mỗi số đó được gọi là một quân cờ và lưới ô đó được gọi là bàn cờ. Có một vị trí của bàn cờ bỏ trống. Mỗi lần di chuyển quân, người chơi được phép chuyển một quân ở vị trí ô tiếp giáp cạnh với ô trống vào ô trống.
- *Yêu cầu*:

+Từ một trạng thái ban đầu (a) (sự sắp xếp ban đầu của các quân trên bàn cờ), hãy thực hiện các nước đi hợp lệ để thu được trạng thái kết thúc (b) (trạng thái đích cần đạt được).

+SV nghiên cứu điều kiện của trạng thái ban đầu, trạng thái đích để bài toán taci có lời giải

Dùng thuật toán A^{KT} giải bài toán TACI với n =3 như sau:



Trạng thái ban đầu (a) Trạng thái kết thúc (b)

Với độ ước lượng $H = \sum_{i=1}^{n^2-1} \delta(a_i, b_i)$ Trong đó $\delta(a_i, b_i) = 0$ nếu $a_i = b_i$ và $\delta(a_i, b_i) = 1$ nếu a_i khác b_i

Hướng dẫn

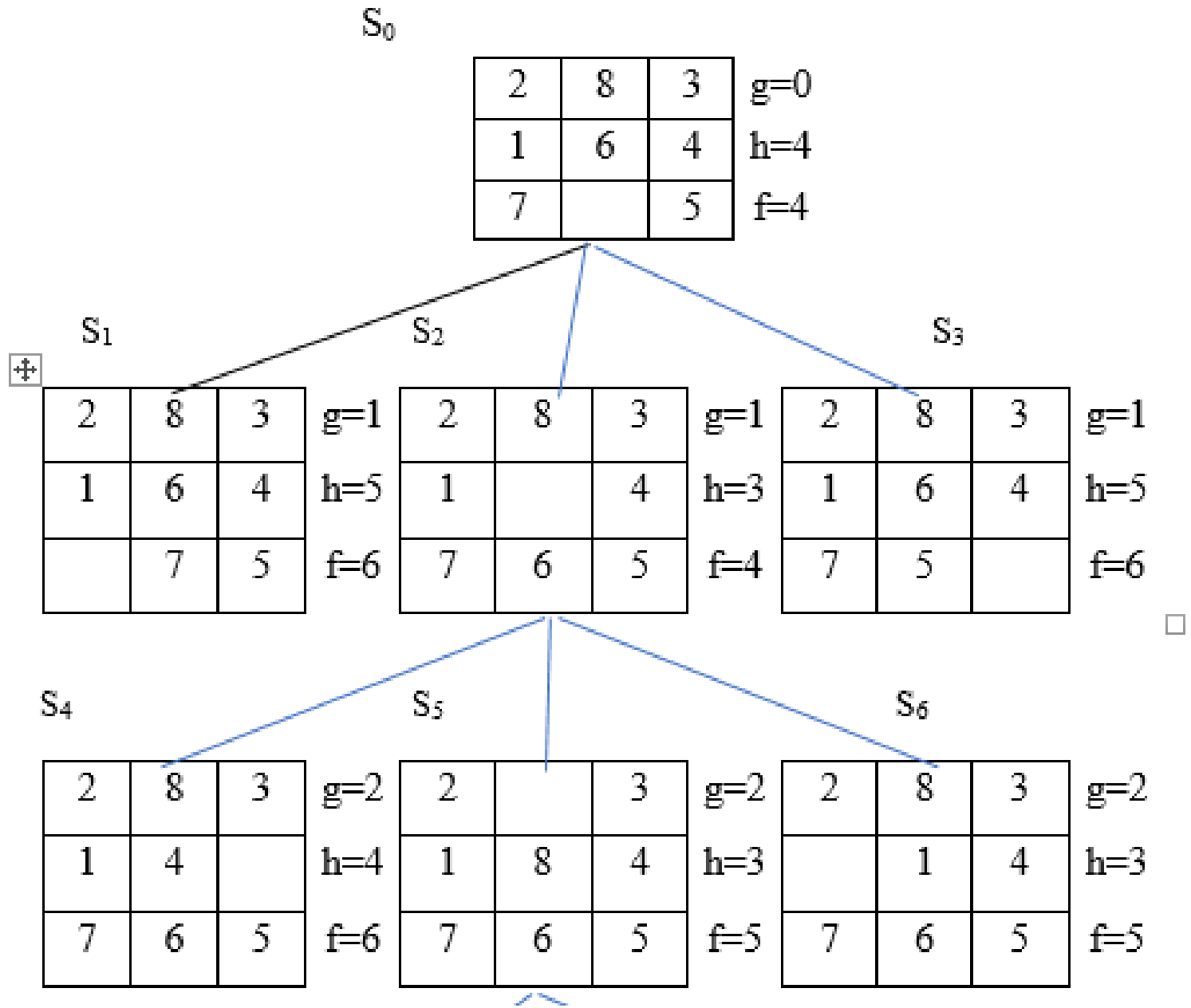
+ Gọi trạng thái bắt đầu của bài toán là đỉnh S_0

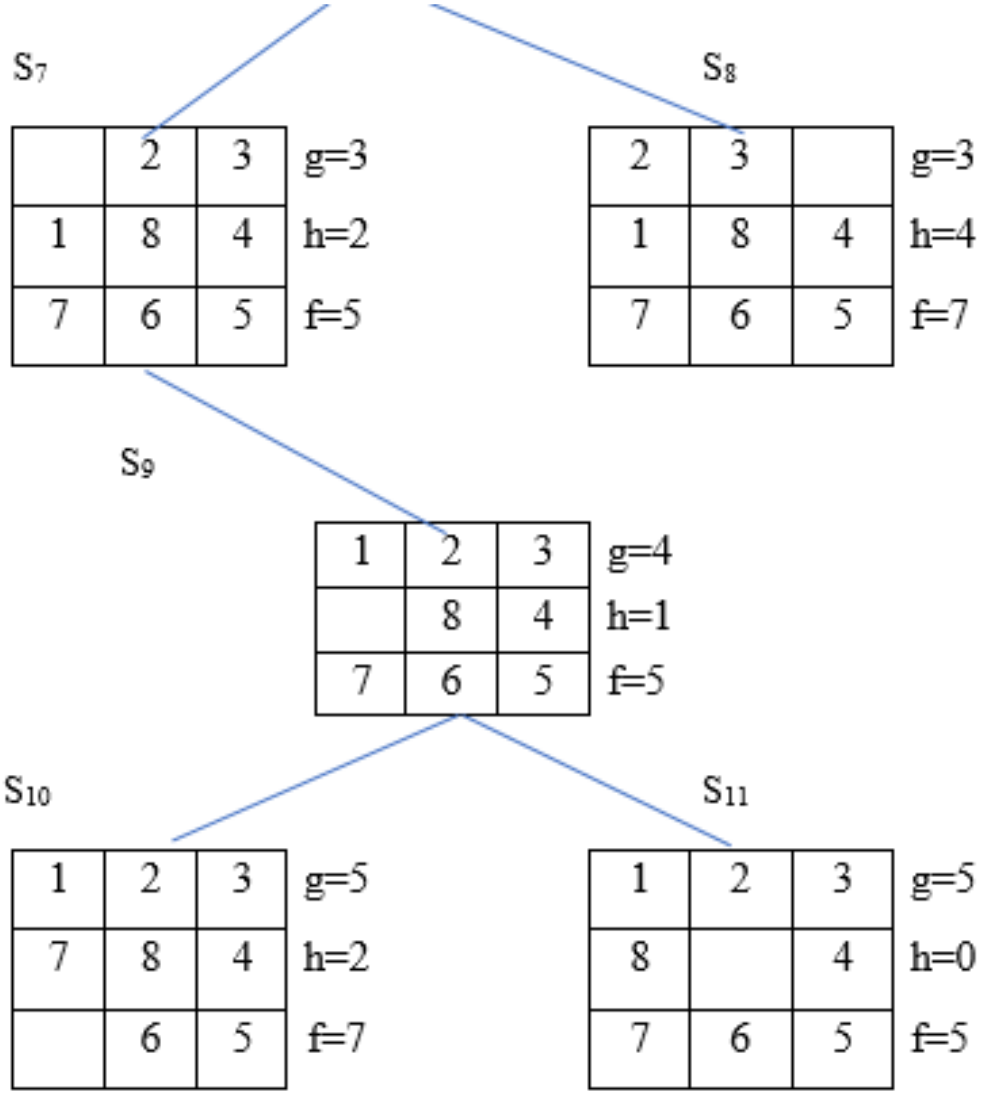
+ g (S) là cạnh (số bước chuyển) của đường đi ngắn nhất từ đỉnh S_0 đến đỉnh S.

+ h (S) : Giá trị ước lượng được tính dựa vào tri thức bổ sung của bài toán.

+ f (S) = g (S) + h (S)

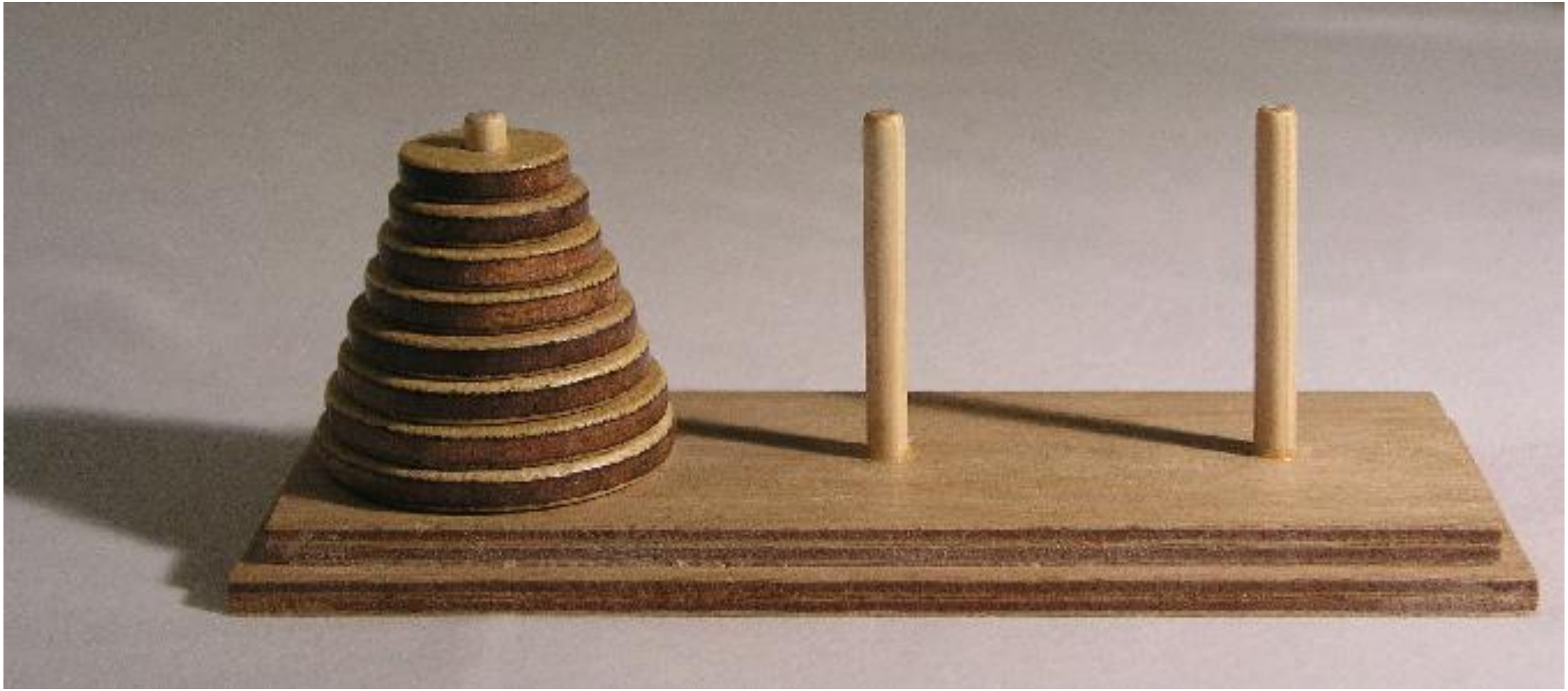
Theo bài toán, ta có cây biểu diễn lời giải như sau:


S0
2 8 3
1 6 4
7 5
g=0
h=4
f=4
S1
2 8 3
1 6 4
7 5
g=1
h=5
f=6
S2
2 8 3
1 4
7 6 5
g=1
h=3
f=4
S3
2 8 3
1 6 4
7 5
g=1
h=5
f=6
S4
2 8 3
1 4
7 6 5
g=2
h=4
f=6
S5
2 3
1 8 4
7 6 5
g=2
h=3
f=5
S6
2 8 3
1 4
7 6 5
g=2
h=3
f=5



Vậy có $g = 5$ bước chuyển đổi đi từ trạng thái ban đầu về trạng thái đích:

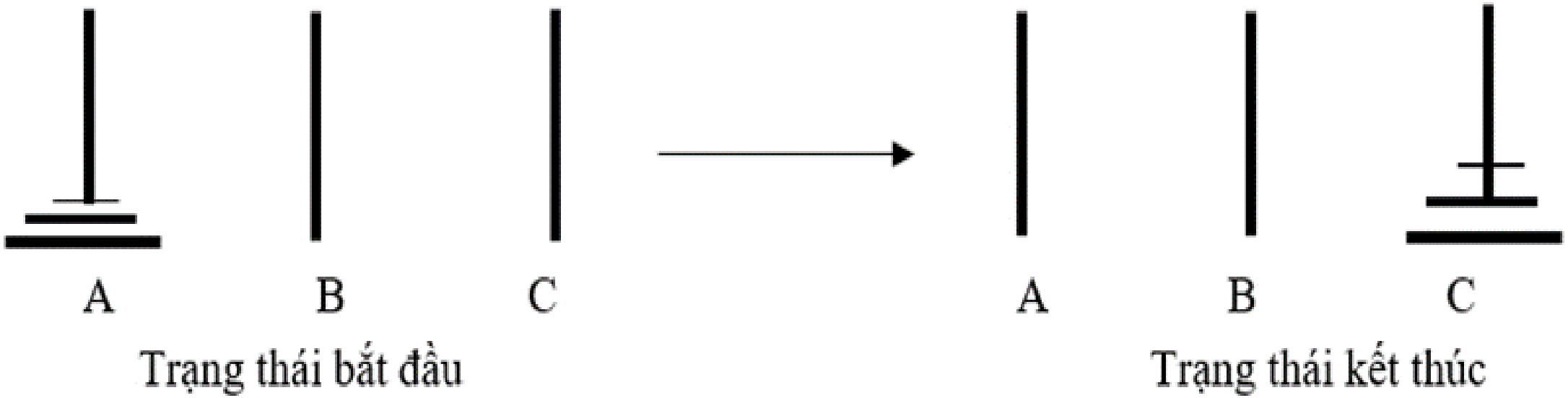
$S_0 \rightarrow S_2 \rightarrow S_5 \rightarrow S_7 \rightarrow S_9 \rightarrow S_{11}$



https://www.youtube.com/watch?v=CrBTg0nms2Y

https://www.buaxua.vn/tro-choi/tro-choi-thap-ha-noi

VD8.Bài toán THÁP HÀ NỘI



Trạng thái bắt đầu

Trạng thái kết thúc

Hướng dẫn

Gọi trạng thái bất đầu của bài toán là S_o

$g(S)$ là số bước dịch chuyển ngắn nhất từ đỉnh So đến đỉnh trạng thái S.

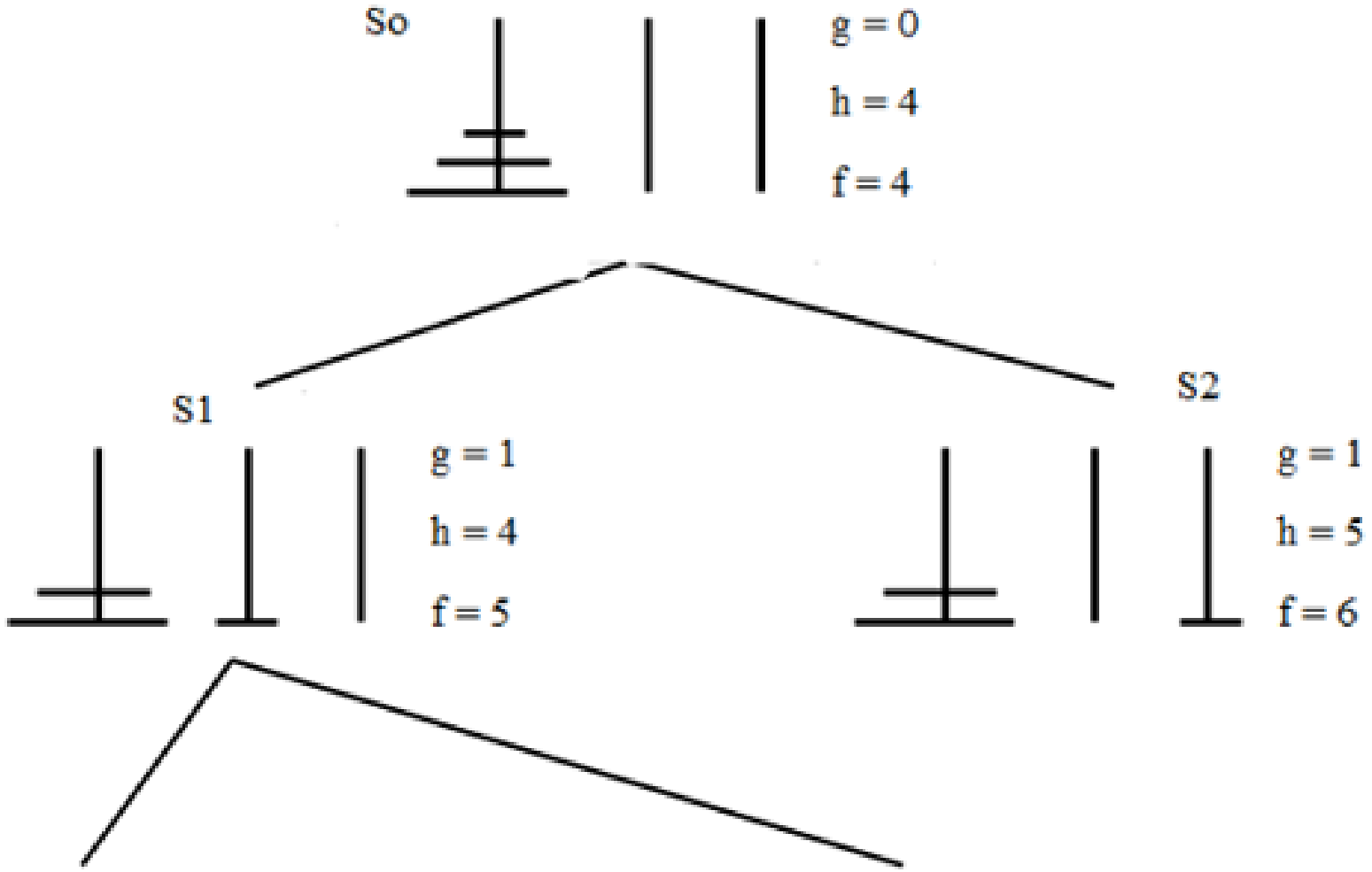
$h(S)$:giá trị ước lượng được tính dựa vào tri thức bổ sung của bài toán; trong bài này $h(S)$ là số lần ít nhất phải di chuyển đĩa từ trạng thái S về trạng thái đích.

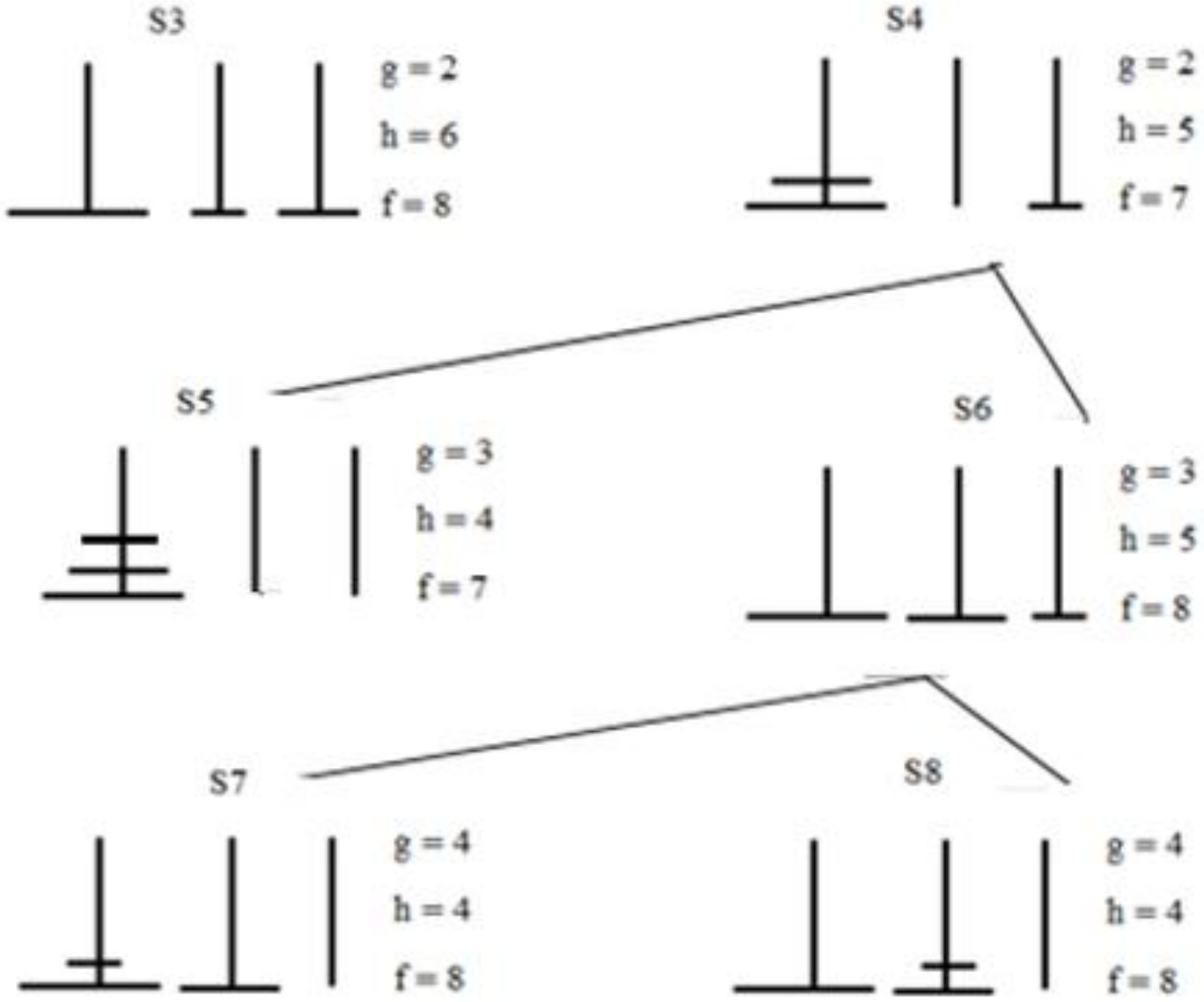
$f(s) = g(s) + h(s)$.

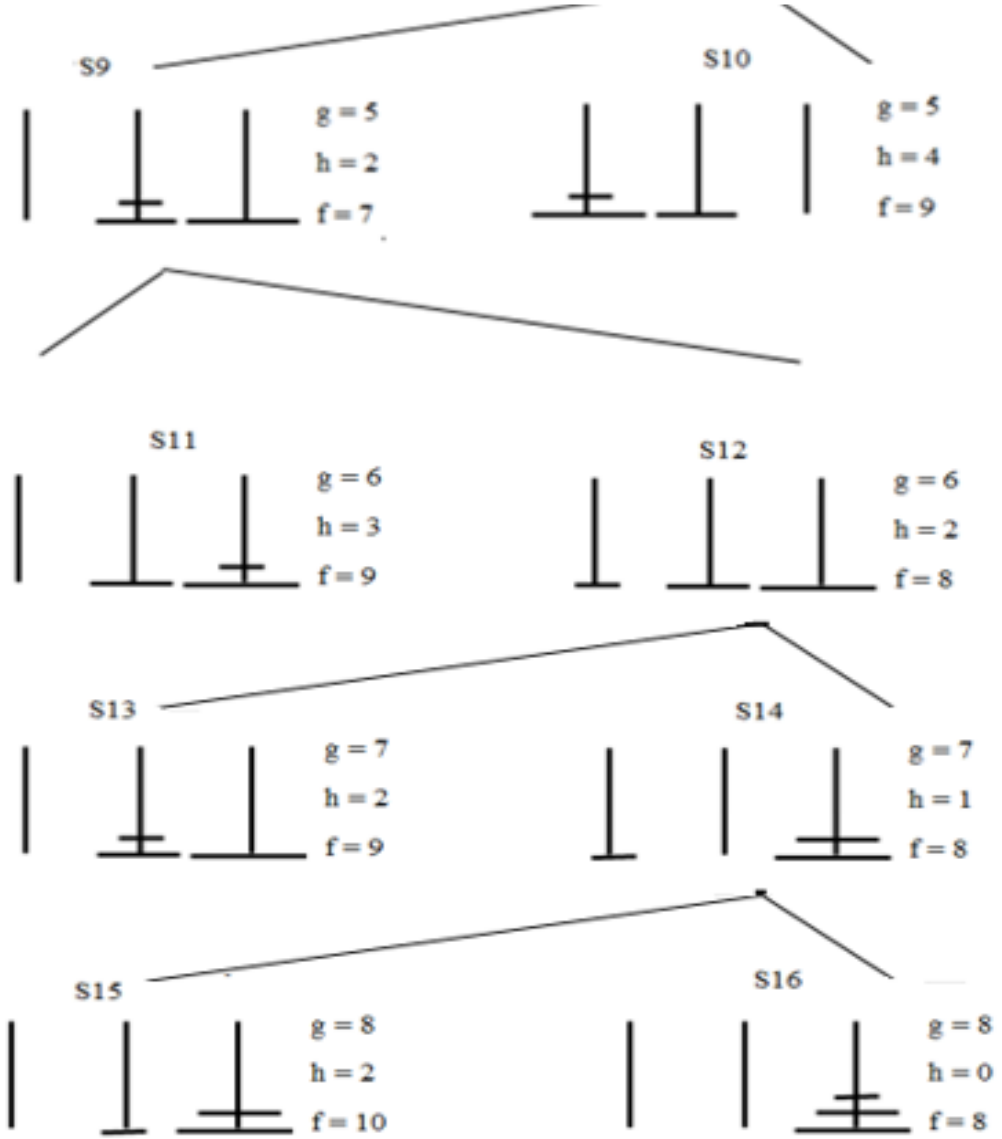
Các trường hợp có thể ở cột C và giá trị h tương ứng; trong đó 1 ứng với đĩa nhỏ nhất, 2 ứng với đĩa vừa và 3 ứng với đĩa lớn nhất.

1							
2	2		1				1
3	3	3	3		1	2	2
h=0	h=1	h=2	h=3	h=4	h=5	h=6	h=7

Theo bài toán, ta có cây biểu diễn lời giải như sau:




S3
g = 2
h = 6
f = 8
S4
g = 2
h = 5
f = 7
S5
g = 3
h = 4
f = 7
S6
g = 3
h = 5
f = 8
S7
g = 4
h = 4
f = 8
S8
g = 4
h = 4
f = 8



Vậy có $g=8$ bước chuyển để đi từ trạng thái ban đầu về trạng thái đích:

$$S_0 \rightarrow S_1 \rightarrow S_4 \rightarrow S_6 \rightarrow S_8 \rightarrow S_9 \rightarrow S_{12} \rightarrow S_{14} \rightarrow S_{16}$$

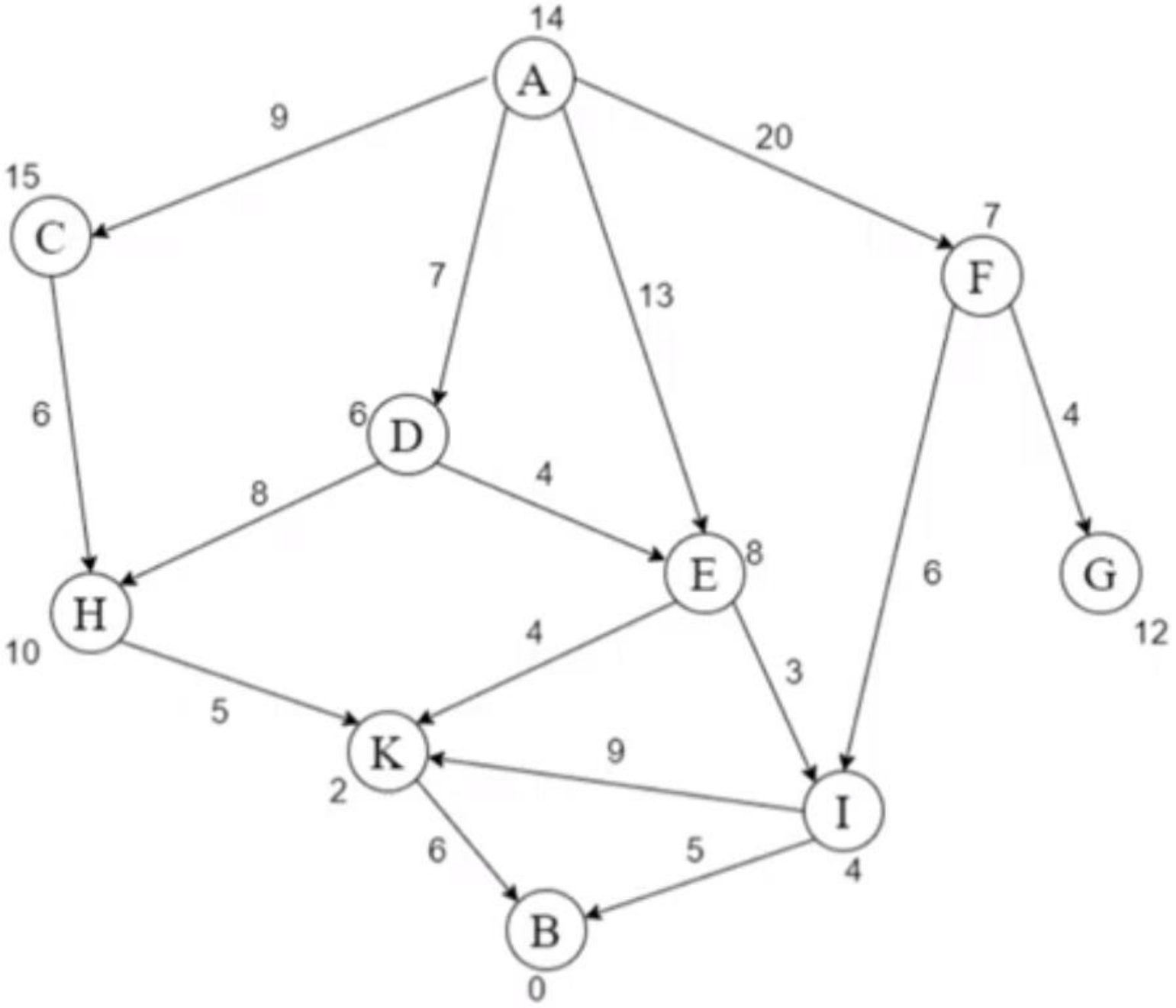
Thuật toán tìm kiếm A*

Thuật giải A*

1. `Open` : tập các trạng thái đã được sinh ra nhưng chưa được xét đến.
2. `Close` : tập các trạng thái đã được xét đến.
3. `Cost(p, q)` : là khoảng cách giữa `p` , `q` .
4. `g(p)` : khoảng cách từ trạng thái đầu đến trạng thái hiện tại `p` .
5. `h(p)` : giá trị được lượng giá từ trạng thái hiện tại đến trạng thái đích.
6. `f(p) = g(p) + h(p)`
    - Bước 1:
        - `Open: = {s}`
        - `Close: =` {}
    - Bước 2: `while (Open !={})`
        - Chọn trạng thái (đỉnh) tốt nhất `p` trong `Open` (xóa `p` khỏi `Open`).
        - Nếu `p` là trạng thái kết thúc thì thoát.
        - Chuyển `p` qua `Close` và tạo ra các trạng thái kế tiếp `q` sau `p` .
            - Nếu `q` đã có trong `Open`
                - Nếu `g(q) > g(p) + Cost(p, q)`
                    - `g(q) = g(p) + Cost(p, q)`
                    - `f(q) = g(q) + h(q)`
                    - `prev(q) = p` (đỉnh cha của `q` là `p`)
            - Nếu `q` chưa có trong `Open`
                - `g(q) = g(p) + cost(p, q)`
                - `f(q) = g(q) + h(q)`
                - `prev(q) = p`
                - Thêm `q` vào `Open`
            - Nếu `q` có trong `Close`
                - Nếu `g(q) > g(p) + Cost(p, q)`
                    - Bỏ `q` khỏi `Close`
                    - Thêm `q` vào `Open`
    - Bước 3: Không tìm được.

Xét ví dụ 9a.

Tìm đường đi ngắn nhất từ A đến B theo thuật toán A*

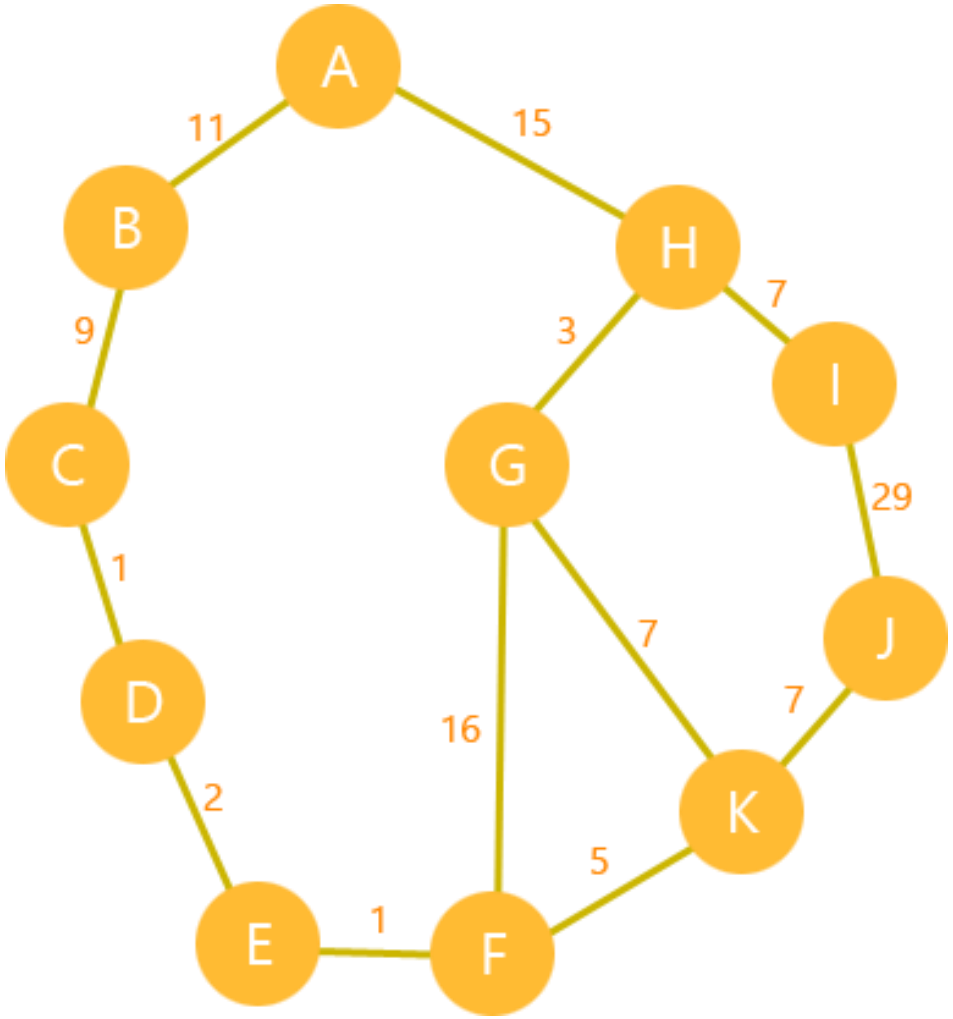


Xét ví dụ 9b

- Ước lượng khoảng cách từ đỉnh hiện tại cho đến đỉnh kết thúc f(x)=g(x)+h(x) trong đó g là khoảng cách ngắn nhất từ đỉnh hiện tại đến đích.
- Open: tập các trạng thái đã được sinh ra nhưng chưa được xét đến.
- Close: tập các trạng thái đã được xét đến.
- Cost(p, q): là khoảng cách giữa p, q.
- g(p): khoảng cách từ trạng thái đầu đến trạng thái hiện tại p.
- h(p): giá trị được lượng giá từ trạng thái hiện tại đến trạng thái đích.
- f(p) = g(p) + h(p)

h(A) = 60 / h(B) = 53 / h(C) = 36 / h(D) = 35 / h(E) = 31 / h(F) = 19 / h(G) = 16 / h(H) = 38 / h(I) = 23 / h(J) = 7 / h(K) = 0

Đỉnh bắt đầu A, Đỉnh kết thúc K.



Xét ví dụ 9c: h(G)=47; h(I)=44;

Xét ví dụ 9d: h(G)=47; h(I)=44; (F,K)=25

TÌM KIẾM CỤC BỘ VÀ BÀI TOÁN TỐI ƯU

- Tìm kiếm cục bộ (local search)
- Tìm kiếm leo đồi (hill-climbing search)
- Thuật giải di truyền (genetic algorithm)
- Thuật giải bees