Chương 6
Giải thuật quay lui

Giải thuật quay lui

Giải thuật nhánh-và-cận

Giải thuật quay lui

Một phương pháp tổng quát để giải quyết vấn đề: thiết kế giải thuật tìm lời giải cho bài tóan không phải là bám theo một tập qui luật tính tóan được xác định mà là bằng cách *thử và sửa sai* (*trial and error*).

Khuôn mẫu thông thường là phân rã quá trình thử và sửa sai thành những công tác bộ phận. Thường thì những công tác bộ phận này được diễn tả theo lối đệ quy một cách thuận tiện và bao gồm việc *thăm dò một số hữu hạn những công tác con*.

Ta có thể coi toàn bộ quá trình này như là một *quá trình tìm kiếm* (search process) mà dần dần cấu tạo và duyệt qua một cây các công tác con.

Bài toán đường đi của con hiệp sĩ (The Knight's Tour Problem)

Cho một bàn cờ $n \times n$ với n^2 ô. Một con hiệp sĩ – được di chuyển tuân theo luật chơi cờ vua – được đặt trên bàn cở tại ô đầu tiên có tọa độ x_0, y_0.

Vấn đề là tìm một lộ trình gồm $n^2 - 1$ bước sao cho phủ toàn bộ bàn cờ (mỗi ô được viếng đúng một lần).

Cách rõ ràng để thu giảm bài toán phủ n^2 ô là xét bài toán, hoặc là

- thực hiện bước đi kế tiếp, hay

- phát hiện rằng không kiếm được bước đi hợp lệ nào.

```
procedure try next move;
begin  initialize selection of moves;
   repeat
      select next candidate from list of next moves;
       if acceptable then
  begin
          record move;
           if board not full then
                begin
                      try next move;                         (6.3.1)
                 if not successful then erase previous recording
                end
        end
   until (move was successful) ∨ (no more  candidates)
end
```

Cách biểu diễn dữ liệu

Chúng ta diễn tả bàn cờ bằng một ma trận *h*.

type index = 1..n **;**
var h: **array**[index, index] **of** integer;
h[x, y] = 0: ô <x,y> chưa hề được viếng
h[x, y] = i: ô <x,y> đã được viếng tại bước chuyển thứ *i* ($1 \le i \le n^2$)

Điều kiện "board not full" có thể được diễn tả bằng "$i < n^2$".

u, v: tọa độ của ô đến.

Điều kiện "acceptable" có thể được diễn tả bằng $(1 \le u \le n) \wedge (1 \le v \le n) \wedge (h[u,v]=0)$

```
procedure try(i: integer; x,y : index; var q: boolean);
var u, v: integer; q1 : boolean;
begin  initialize selection for moves;
   repeat  let u, v be the coordinates of the next move ;
     if (1≤u≤n) ∧ (1≤v≤n) ∧ (h[u,v]=0) then
     begin  h[u,v]:=i;
        if i < sqr(n) then                                    (6.3.2)
        begin
           try(i + 1, u, v, q1);  if ¬ q1 then h[u,v]:=0
        end
        else q1:= true
      end
    until q1 ∨ (no more candidates);
   q:=q1
end
```

Cho tọa độ của ô hiện hành <x, y>, có 8 khả năng để chọn ô kế tiếp <u, v> để đi tới. Chúng được đánh số từ 1 đến 8 như sau:

	3		**2**	
4				**1**
		⊕		
5				**8**
	6		**7**	

Sự tinh chế sau cùng

Cách đơn giản nhất để đạt được tọa độ *u, v* từ *x, y* là bằng cách cọng độ sai biệt toạ độ tại hai mảng *a* và *b.*

Và *k* được dùng để đánh số ứng viên (candidate) kế tiếp.

```
program knightstour (output);
const n = 5; nsq = 25;
type index = 1..n
var  i,j: index;  q: boolean;
s: set of index;
a,b: array [1..8] of integer;
h: array [index, index] of integer;
```

```
procedure try (i: integer; x, y: index; var q:boolean);
var k,u,v :  integer; q1: boolean;
begin    k:=0;
   repeat
      k:=k+1; q1:=false; u:=x+a[k]; v:=y+b[k];
      if (u in s) ∧ (v in s) then
        if h[u,v]=0 then
        begin
             h[u,v]:=i;
              if i < nsq then
              begin
                  try(i+1, u,v,q1);
                   if ¬ q1 then h[u,v]:=0
              end
              else q1:=true
       end
   until q1 ∨ (k =8);
   q:=q1
end {try};
```

```
begin
      s:=[1,2,3,4,5];
a[1]:=  2;  b[1]:=  1;
a[2]:=  1; b[2]:= 2;
a[3]:=  –1; b[3]:= 2;
a[4]:=  –2;   b[4]:=1;
a[5]:=  –2;   b[5]:= –1;
a[6]:=  –1;  b[6]:= –2;
a[7]:=  1;     b[7]:= –2;
a[8]:=  2;     b[8]:= –1;
  for i:=1 to n do
for j:=1 to n do h[i,j]:=0;
h[1,1]:=1; try (2,1,1,q);
 if q then
     for i:=1 to n do
     begin
          for j:=1 to n do
               write(h[i,j]:5);
          writeln
      end
  else writeln ('NO
SOLUTION')
end.
```

Thủ tục đệ quy được khởi động bằng lệnh gọi với tọa độ khởi đầu x_0, y_0 , từ đó chuyến đi bắt đầu.

$H[x_0,y_0]:= 1; try(2, x_0, y_0, q)$

Hình 6.3.1 trình bày một lời giải đạt được với vị trí <1,1> với n = 5.

1	**6**	**15**	**10**	**21**
14	**9**	**20**	**5**	**16**
19	**2**	**7**	**22**	**11**
8	**13**	**24**	**17**	**4**
25	**18**	**3**	**12**	**23**

Từ thí dụ trên ta đi đến với một kiểu “giải quyết vấn đề” mới:

Đặc điểm chính là

“bước hướng về lời giải đầy đủ và ghi lại thông tin về bước này mà sau đó nó có thể bị tháo gỡ và xóa đi khi phát hiện rằng bước này đã không dẫn đến lời giải đầy đủ, tức là một bước đi dẫn đến “tình thế bế tắc”(*dead-end*). (Hành vi này được gọi là *quay lui - bactracking*.)

Khuôn mẫu tổng quát của giải thuật quay lui

```
procedure try;
begin intialize selection of candidates;
repeat
   select next;
        if acceptable then
        begin
           record it;
           if solution incomplete then
           begin
              try next step;                        (6.3.3)
              if not successful then cancel recording
           end
        end
until successful ∨ no more candidates
end
```

Nếu tại mỗi bước, số ứng viên phải thử là cố định thì kiểu mẫu trên có thể biến đổi như :

⇒

Thủ tục được gọi bằng lệnh gọi

try(1).

```
procedure try (i: integer);
var k : integer;
begin k:=0;
    repeat
        k:=k+1; select k-th candidate;
        if acceptable then
        begin
            record it;
            if i<n then
            begin
                try (i+1);                          (6.3.4)
                if not successful then
                            cancel recording
            end
        end
     until successful ∨ (k=m)
end
```

Bài toán 8 con hậu

Bài toán này đã được C.F. Gauss khảo sát năm 1850, nhưng ông ta không hoàn toàn giải quyết được.

"Tám con hậu được đặt vào bàn cờ sao cho không có con hậu nào có thể tấn công con hậu nào".

Dùng khuôn mẫu ở hình 6.3.1, ta sẽ có được một thủ tục sau cho bài toán 8 con hậu:

```
procedure try (i: integer);
begin
    initialize selection of positions for i-th queen;
    repeat
        make next selection;
        if safe then
        begin
            setqueen;
            if i < 8 then
            begin
                try (i + 1);
                if not successful then remove queen
            end
         end
     until successful ∨ no more positions
end
```

Luật cờ: *Một con hậu có thể tấn công các con hậu khác nằm trên cùng một hàng, cùng một cột hay là cùng đường chéo trên bàn cờ.*

Cách biểu diễn dữ liệu

Làm cách nào để diễn tả 8 con hậu trên bàn cờ?

var x: **array**[1..8] of integer;
a: **array**[1..8] of Boolean;
b: **array**[b1..b2] of Boolean;
c: **array**[c1..c2] of Boolean;

với

x[i] chỉ vị trí của con hậu trên cột thứ *i*;

a[j] cho biết không có con hậu trên hàng thứ *j*;

b[k] cho biết không có con hậu trên đường chéo ⇙ thứ *k*;

c[k] cho biết không có con hậu trên đường chéo ⇘ thứ *k*.

Việc chọn trị cho các mốc *b1*, *b2*, *c1*, *c2* được xác định bởi cách mà các chỉ số của các mảng *b* và *c* được tính. Hãy chú ý rằng trên cùng một đường chéo chiều ⇙ tất cả các ô sẽ có cùng giá trị của tổng hai tọa độ *i +j*, và trên cùng một đường chép chiều ⇘ diagonal, tất cả các ô sẽ có cùng giá trị của hiệu hai tọa độ (*i – j*).

Như vậy, phát biểu *setqueen* được tinh chế như sau:

x[i]:=j; a[j]:=false; b[i+j]:=false;c[i-j]:=false;

Phát biểu *removequeen* được chi tiết hóa như sau:

a[j] = true; b[i+j] = true ; c[i-j] := true

Điều kiện *safe* được diễn tả như sau:

a[j] $\wedge$ b[i+j] $\wedge$ c[i-j]

```
program eightqueeen1(output);
{find one solution to eight queens problem}
var     i : integer; q: boolean;
a : array [1..8]  of boolean;
b : array [2..16] of boolean;
c : array [–7..7] of boolean;
x : array [1..8]  of integer;
procedure try(i: integer; var q: boolean);
var j: integer;
begin
  j:=0;
  repeat
    j:=j+1; q:=false;
    if a[j] ∧ b[i+j] ∧ c[i-j] then
    begin
      x[i]:=j;
      a[j]:=false; b[i+j]:=false;
      c[i-j]:=false;
      if i<8 then
      begin
        try (i+1, q);
        if ¬ q then
        begin
          a[j]:=true; b[i+j]:=true;
          c[i-j]:=true
        end
      end
      else q:=true
    end
  until q ∨ (j=8)
end {try};
```

```
begin
   for i:= 1 to 8 do a[i]:=true;
   for i:= 2 to 16 do b[i]:=true;
   for i:= –7 to 7 do c[i]:=true;
   try (1,q);
   if q then
     for i:=1 to 8 do
                     write (x[i]:4);
   writeln
end
```

Một lời giải của bài toán 8 con hậu được cho ở hình vẽ sau:

1	**H**							
2							**H**	
3					**H**			
4								**H**
5		**H**						
6				**H**				
7						**H**		
8			**H**					

Sự mở rộng: Tìm tất cả các lời giải

Sự mở rộng là tìm không chỉ một lời giải mà tất cả những lời giải của bài toán đã cho.

Phương pháp: ***Một khi một lời giải được tìm thấy và ghi lại, ta tiếp tục xét ứng viên kế trong quá trình chọn ứng viên một cách có hệ thống*.**

Khuôn mẫu tổng quát được dẫn xuất từ (6.3.4) và được trình bày như sau:

```
procedure try(i: integer);
var k: integer;
begin
    for k:=1 to m do
    begin
        select k-th candidate;
        if acceptable then
        begin
            record it;
            if i<n then try (i+1) else print solution;
            cancel recording
        end
    end
end
```

Trong giải thuật mở rộng, để đơn giản hóa điều kiện dừng của quá trình chọn, phát biểu *repeat* được thay thế bằng phát biểu *for*

```
program eightqueens(output);
var i: integer;
a: array [1.. 8]   of boolean;
b: array [2.. 16] of boolean;
c: array [–7.. 7] of boolean;
x: array [1.. 8]   of integer;
procedure print;
var k : integer;
begin
   for k : 1 to 8 do write(x[k]:4);
   writeln
end {print};
```

```
procedure try (i:integer);
var j: integer;
begin
   for j:=1 to 8 do
      if a[j] ∧ b[i+j] ∧ c[i-j] then
      begin
         x[i]:=j;
         a[j]:=false; b[i+j]:= false;
         c[i-j]:=false;
         if i < 8 then try(i+1) else print;
         a[j]:=true; b[i+j]:= true;
         c[i-j]:= true;
      end
end {try};
```

```
begin
   for i:= 1 to 8 do a[i]:=true;
   for i:= 2 to 16 do b[i]:=true;
   for i:= –7 to 7 do c[i]:=true;
   try(1);
end.
```

Giải thuật mở rộng có thể sản sinh tất cả 92 lời giải cho bài toán 8 con hậu.

Nhưng thật ra chỉ có 12 lời giải thật sự khác biệt nhau.

Mười hai lời giải đó được liệt kê trong bảng sau:

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	N
1	5	8	6	3	7	2	4	876
1	6	8	3	7	4	2	5	264
1	7	4	6	8	2	5	3	200
1	7	5	8	2	4	6	3	136
2	4	6	8	3	1	7	5	504
2	5	7	1	3	8	6	4	400
2	5	7	4	1	8	6	3	72
2	6	1	7	4	8	3	5	280
2	6	8	3	1	4	7	5	240
2	7	3	6	8	5	1	4	264
2	7	5	8	1	4	6	3	160
2	8	6	1	3	5	7	4	336

Những giá trị ở cột N chỉ số lần thử để tìm một ô an toàn. Trung bình cần 161 phép thử trong 92 lời giải này.

Cây không gian trạng thái

- Để tiện diễn tả giải thuật quay lui, ta xây dựng cấu trúc cây ghi những lựa chọn đã được thực hiện. Cấu trúc cây này được gọi là *cây không gian trạng thái* (state space tree) hay *cây tìm kiếm* (search tree).
- Nút rễ của cây diễn tả *trạng thái đầu tiên* trước khi quá trình tìm kiếm lời giải bắt đầu.
- Các nút ở mức đầu tiên trong cây diễn tả những lựa chọn được làm ứng với thành phần đầu tiên của lời giải.
- Các nút ở mức thứ haì trong cây diễn tả những lựa chọn được làm ứng với thành phần thứ hai của lời giải và các mức kế tiếp tương tự như thế.

Một nút trên cây KGTT được gọi là *triển vọng* nếu nó tương ứng với lời giải bộ phận mà sẽ có thể dẫn đến lời giải đầy đủ; trái lại, nó được gọi là một lời giải *không triển vọng*.

Các nút lá diễn tả những trường hợp *bế tắc* (dead end) hay những *lời giải đầy đủ*.

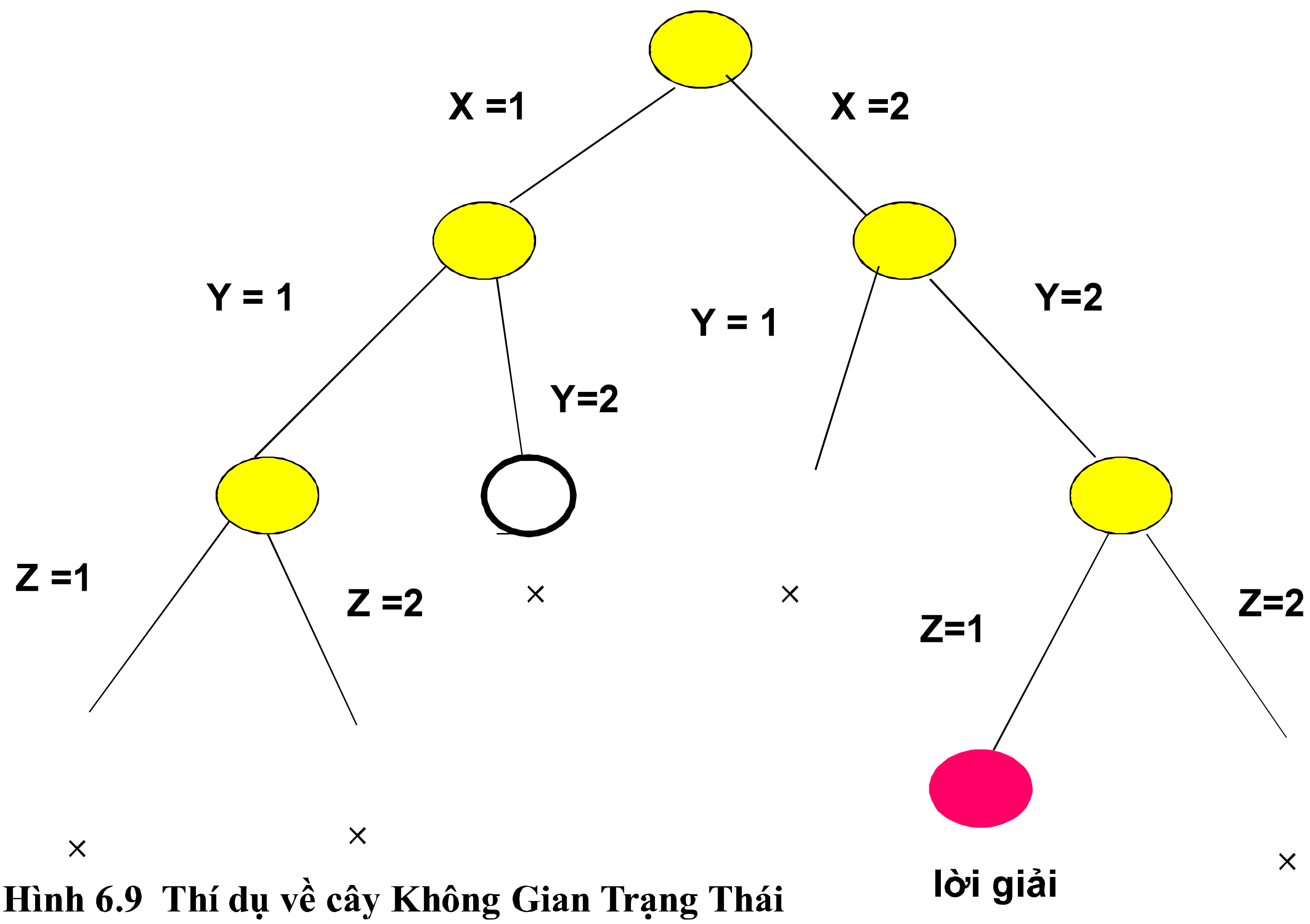
Thí dụ: Cho một bài toán như sau:

Tập biến: X, Y, Z.

Gán trị từ tập {1,2} vào các biến sao cho thỏa mãn các ràng buộc: $X = Y$, $X \neq Z$, $Y > Z$.

Hãy giải bài toán bằng một giải thuật quay lui.

Cây không gian trạng thái của bài toán này được cho ở hình vẽ sau:



Hình 6.9 Thí dụ về cây Không Gian Trạng Thái

Độ phức tạp của giải thuật quay lui

Thời gian tính toán của các giải thuật quay lui thường là hàm mũ (*exponential*).

Nếu mỗi nút trên cây không gian trạng thái có trung bình α nút con, và chiều dài của lối đi lời giải là N, thì số nút trên cây sẽ tỉ lệ với α^N.

Thời gian tính toán của giải thuật đệ quy tương ứng với số nút trên cây không gian trạng thái nên có độ phức tạp hàm mũ.

Giải thuật nhánh và cận (branch-and-bound)

- **Bài toán người thương gia du hành (TSP): cho một tập các thành phố và khoảng cách giữa mỗi cặp thành phố, tìm một lộ trình đi qua tất cả mọi thành phố sao cho tổng khoảng cách của lộ trình nhỏ hơn M.**

Điều này dẫn đến một bài toán khác: cho một đồ thị vô hướng, có cách nào để nối tất cả các nút bằng một chu trình đơn hay không. Đây chính là bài toán Chu trình Hamilton (HCP).

Để giải bài toán (HCP), ta có thể cải biên giải thuật tìm kiếm theo chiều sâu trước (DFS) để giải thuật này có thể sinh ra mọi lối đi đơn mà đi qua mọi đỉnh trong đồ thị.

Tìm kiếm vét cạn: Giải thuật DFS cải biên sinh ra mọi lối đi đơn

Điều này có thể thực hiện được bằng cách sửa lại thủ tục *visit* như sau:

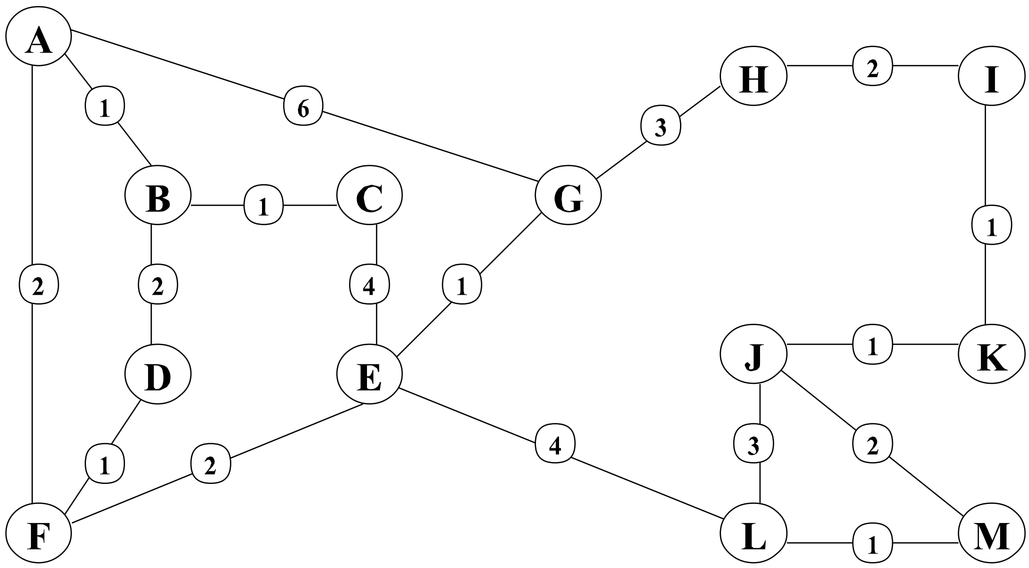
```
procedure visit( k: integer);
var t: integer;
begin
  id := id +1; val[k]:= id;
   for t:= 1 to V do
      if a[k, t] then
         if val[k]= 0 then visit(t);
  id := id –1; val[k] := 0
end;
```

Thủ tục đệ quy này có thể sinh ra mọi lối đi đơn từ một đỉnh khởi đầu nào đó.

Ví dụ:

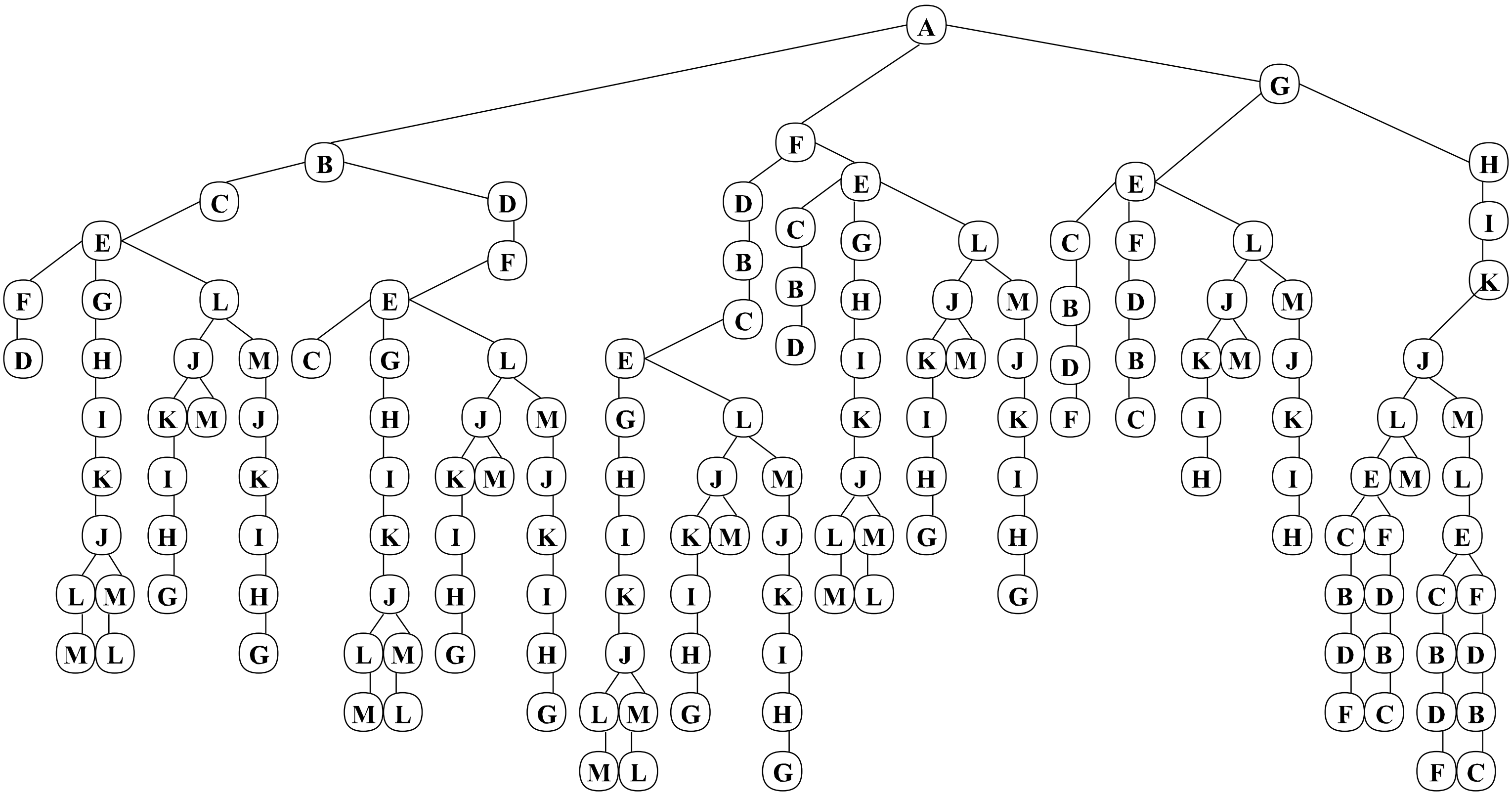
```
id :=0;
for k:= 1 to V do val[k]:=0;
  visit(1);
```

Một thí dụ về bài toán TSP



Hình 5.10

Tìm kiếm vét cạn các lối đi đơn



Hình 5.11

Từ giải thuật sinh tất cả các lối đi đơn đến giải thuật giải bài toán TSP

- **Ta có thể cải biên thủ tục *visit* ở trên để có thể nhận diện chu trình Hamilton bằng cách cho nó kiểm tra xem có tồn tại một cạnh nối từ đỉnh *k* về đỉnh 1 xuất phát khi *val[k]=V* hay không.**
- **Trong thí dụ trên, xem hình vẽ, ta tìm thấy 2 chu trình Hamilton là**
  - A F D B C E L M J K I H G
  - A G H I K J M L E C B D F và hai chu trình này chỉ là một.

- **Chương trình nhận diện chu trình Halmiton có thể được sửa đổi để có thể giải bài toán TSP bằng cách** theo dõi chiều dài của lối đi hiện hành trong mảng *val*, và theo dõi lối đi có chiều dài nhỏ nhất trong số các chu trình Hamilton tìm thấy.

Ý tưởng nhánh và cận

Khi áp dụng giải thuật DFS cải biên để sinh ra mọi lối đi đơn, trong quá trình tìm kiếm một lối đi tốt nhất (tổng trọng số nhỏ nhất) cho bài toán TSP, có một kỹ thuật *tỉa nhánh* quan trọng là kết thúc sự tìm kiếm ngay khi thấy rằng nó không thể nào thành công được.

Giả sử một lối đi đơn có chi phí *x* đã được tìm thấy. Thì thật vô ích để duyệt tiếp trên lối đi chưa-đầy-đủ nào mà chi phí cho đến hiện giờ đã *lớn hơn* *x*. Điều này có thể được thực hiện bằng cách *không* gọi đệ quy thủ tục *visit* nếu lối đi chưa-đầy-đủ hiện hành đã lớn hơn chi phí của *lối đi đầy đủ tốt nhất* cho đến bây giờ.

Ý tưởng nhánh và cận (tt.)

Rõ ràng ta sẽ không *bỏ sót* lối đi chi phí nhỏ nhất nào nếu ta bám sát một chiến lược như vậy.

Kỹ thuật tính cận (bound) của các lời giải chưa-đầy-đủ để hạn chế số lời giải phải dò tìm được gọi là *giải thuật nhánh và cận*.

Giải thuật này có thể áp dụng khi có chi phí được gắn vào các lối đi.