NHẬP MÔN TRÍ TUỆ NHÂN TẠO

CHƯƠNG 2

GIẢI QUYẾT VẤN ĐỀ BẰNG TÌM KIẾM

Nội dung

- Biểu diễn bài toán trong KGTT
- Tìm kiếm mù (uninformed search)
- Tìm kiếm heuristic (informed search)
- Cây trò chơi, cắt tỉa alpha –beta
- Bài toán thoả mãn ràng buộc

TÌM KIẾM DỰA TRÊN KINH NGHIỆM (INFORMED/ HEURISTIC SEARCH)

Tìm kiếm có thông tin

- Phép đo Heuristic
- Tìm kiếm tốt nhất đầu tiên (Best-First-Search - BFS)
- Tìm kiếm háu ăn (Greedy Best-First Search)
- **Tìm kiếm A***
- **Tìm kiếm leo đồi**

Giải thuật A*

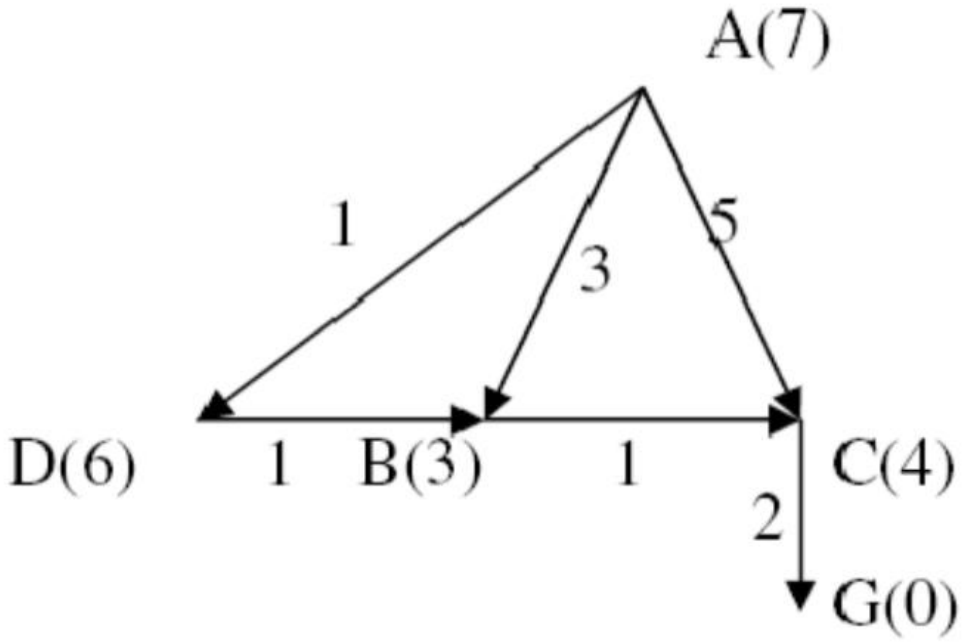
- Giải thuật A* là một trường hợp đặc biệt Best first search (việc cập nhật lại đường đi dựa trên giá trị g(n) thay vì dựa trên giá trị f(n) tổng quát)
  - $f(n) = g(n) + h(n)$
  - ***h(n) phụ thuộc vào trạng thái n*** *nên f(n) chỉ thay đổi khi* ***g(n)*** *thay đổi hay nói cách khác khi ta tìm được một đường đi mới đến n tốt hơn đường đi cũ => cập nhật lại g khi đường đi mới tốt hơn)*
- Mỗi trạng thái n tùy ý sẽ gồm 4 yếu tố (g(n), h(n), f(n), cha(n)) Cha(n) là nút cha của nút đang xét n.

Mã giả giải thuật A*

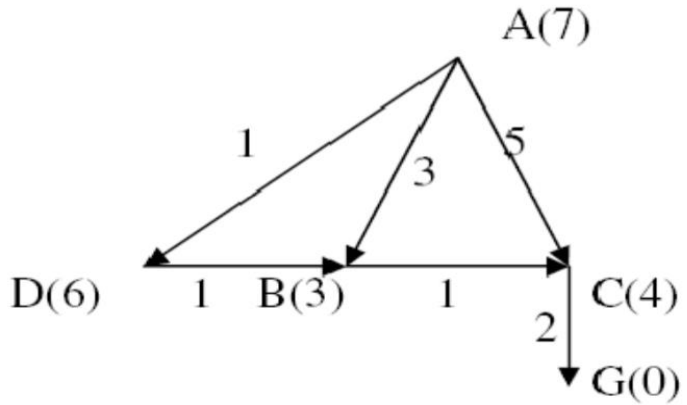
```
g(n_o)=0; f(n_o)=h(n_o);
open:=[n_o]; closed:=[];
while open<>[] do
    loại n bên trái của open và dưa n vào closed;
    if (n là một đích) then thành công, thoát
    else
        Sinh các con m của n;
        For m thuộc con(n) do
            g(m)=g(n)+c[n,m];
            If m không thuộc open hay closed then
                f(m)=g(m)+h(m); cha(m)=n;  Bỏ m vào open;
            If m thuộc open (tồn tại m' thuộc open, sao cho m=m')  then
                If g(m)<g(m') then   g(m')=g(m); f(m')=g(m')+h(m');  Cha(m')=n;
            If m thuộc closed (tồn tại m' thuộc closed, sao cho m=m') then
                If g(m)<g(m') then  f(m)=g(m)+h(m); cha(m)=n;
                                    Đưa m vào open; loại m' khỏi closed;
        Sắp xếp open để t.thái tốt nhất nằm bên trái;
```

Giải thuật A*

Sử dụng giải thuật A* để tìm đường đi từ A đến G với giá trị g(n) được ghi trên cạnh của đồ thị và h(n) ghi ngay đỉnh của đồ thị

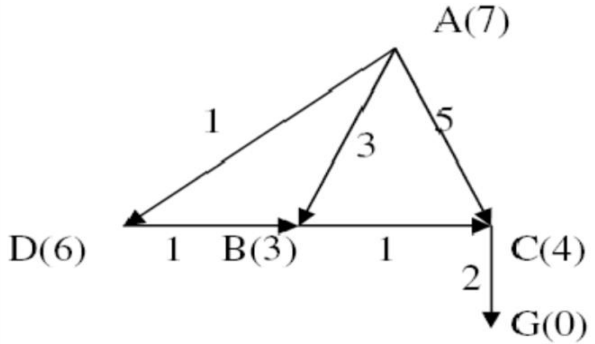


- Mỗi trạng thái n tùy ý sẽ gồm bốn yếu tố (g(n), h(n), f(n), cha(n))
- **Bước 1**:
  - *Open={A(0,7,7,-)};* *Close={}*
- **Bước 2:**
  - *Các con của A: D,B,C*
  - ***Xét D***
    - **g(D) = g(A) + c[A,D] = 0 + 1= 1 (g(m) = g(n) + c[m,n])**
    - **D không thuộc Open; Close**
      - ***Tính giá trị f(D) = g(D) + h(D) = 1 + 6 = 7***
      - ***Cập nhật cha của D: A***
      - ***Đưa D vào open: D(1, 6, 7, A)***
  - ***Xét B***



- **Bước 2:**
  - *Các con của A: D,B,C*
  - ….
  - ***Xét B***
    - **g(B)** = g(A) + c[A,B] = 0+ 3 = 3 (g(m) = g(n) + c[m,n])
    - **B** không thuộc **Open; Close**
      - *Tính giá trị **f(B)** = g(B) + h(B) = 3 + 3 = 6*
      - *Cập nhật cha của **B**: A*
      - *Đưa **B** vào **open**: B(3, 3, 6, A)*
  - ***Xét C***
    - **g(C)** = g(A) + c[A,C] = 0 + 5 = 5 (g(m) = g(n) + c[m,n])
    - **C** không thuộc **Open; Close**
      - *Tính giá trị **f(C)** = g(C) + h(C) = 5+ 4 = 9*
      - *Cập nhật cha của **C**: A*
      - *Đưa **C** vào **open**: C(5, 4, 9, A)*
  - *Sắp xếp các phần tử trong open để trạng thái tốt nhất bên trái*

*Open{**B(3, 3, 6, A)},** D(1, 6,7, A), C(5, 4, 9, A)} ; close={A(0, 7, 7, -)}*

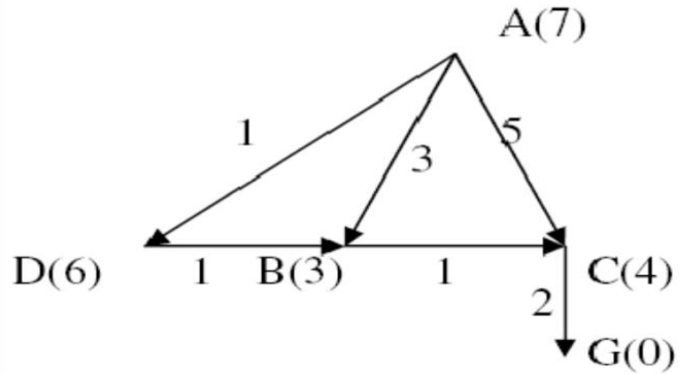


- **Bước 3:**
  - *Quay lại đầu vòng lặp while*
  - *Lấy phần tử **B** ra khỏi Open đưa vào Close*

    *Open = {**D(1, 6, 7, A)**, C(5, 4, 9, A)}*

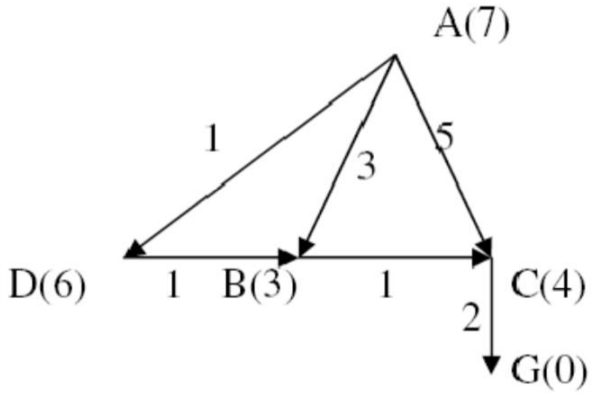
    *Close = {A(0, 7, 7, -), **B(3, 3, 6, A)**}*
  - *Các con của B: C*
  - ***Xét C***
    - **g(C)** = g(B) + c[B,C] = 3 + 1 = 4 (g(m) = g(n) + c[m,n])
    - **C thuộc Open;**
      - *So sánh giá trị $g(C_n)$ hiện tại và $g(C_o)$ đã tồn tại trong Open*
      - *$G(C_o)$ =5 > $g(C_n)$ =4 => cập nhật lại C*
      - *Tính giá trị **f(C)** = g(C) + h(C) = 4 + 4 =8*
      - *Cập nhật cha của **C**: B*
      - *Đưa **C** vào **open**: **C**(4, 4, 8, B)*
  - *Sắp xếp các phần tử trong open để trạng thái tốt nhất bên trái*

*Open{**D(1, 6, 7, A)**, **C**(4, 4, 8, B)}* *; close={A(0, 7, 7, -), **B(3, 3, 6, A)**} }*



- **Bước 4:**
  - *Quay lại đầu vòng lặp while*
  - *Lấy phần tử **D** ra khỏi Open đưa vào Close*

*Open = {**C**(4,4,8,B) }; Close={A(0,7,7,-), B(3,3,6,A), **D(1,6,7,A)**}*



  - *Các con của D: B*
  - ***Xét B***
    - **g(B)** = g(D) + c[D,B] = 1 + 1 = 2 (g(m) = g(n) + c[m,n])
    - **B thuộc Closed;**
      - *So sánh giá trị $g(B_n)$ hiện tại và $g(B_o)$ đã tồn tại trong Open*
      - *$G(B_o)$ =3 > $g(B_n)$ =2 => cập nhật lại B*
      - *Tính giá trị **f(B)** = g(B) + h(B) = 2 + 3 =5*
      - *Cập nhật cha của **B**: D*
      - *Đưa B(2,3,5,D) vào Open*
      - *Loại B(3,3,6,A) ra khỏi Close*
  - *Sắp xếp các phần tử trong Open để trạng thái tốt nhất bên trái*

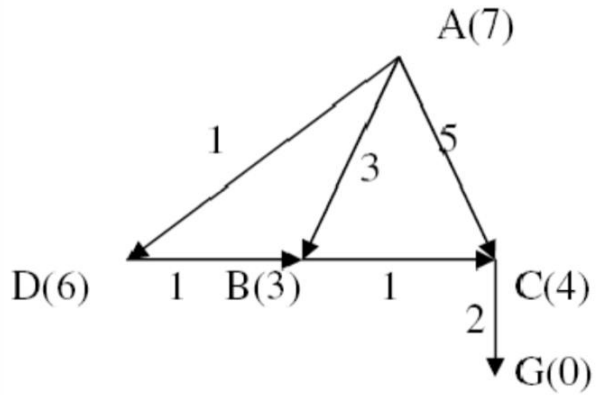
*Open{**B**(2,3,5,D) , **C**(4,4,8,B)} ; Close={A(0,7,7,-), **D(1,6,7,A)**}*

- **Bước 5:**
  - *Quay lại đầu vòng lặp while*
  - *Lấy phần tử **B** ra khỏi Open đưa vào Close*

    Open = {C(4, 4, 8, B)};

    *Close = {A(0, 7, 7, -), D(1, 6, 7, A), **B**(2, 3, 5, D)}*
  - *Các con của B: C*
  - ***Xét C***
    - **g(C)** = g(B) + c[B,C] = 1 + 2 = 3 (g(m) = g(n) + c[m,n])
    - **C thuộc Open;**
      - *So sánh giá trị $g(C_n)$ hiện tại và $g(C_o)$ đã tồn tại trong Open*
      - *$G(C_o)$ =4 > $g(C_n)$ =3 => cập nhật lại C*
      - *Tính giá trị **f(C)** = g(C) + h(C) = 3 + 4 =7*
      - *Cập nhật cha của **C**: B*
      - *Đưa **C** vào **open**: **C**(3, 4, 7, B)*
  - *Sắp xếp các phần tử trong open để trạng thái tốt nhất bên trái*

*Open{**C**(3, 4, 7, B) } ; Close ={A(0, 7, 7, -), D(1, 6, 7, A), **B**(2, 3, 5, D) }*

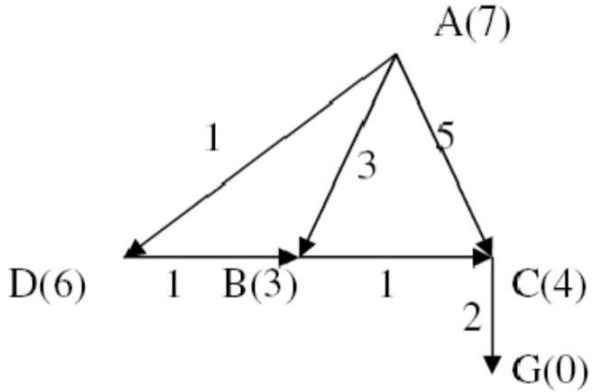


- **Bước 6:**
  - *Quay lại đầu vòng lặp while*
  - *Lấy phần tử **C** ra khỏi Open đưa vào Close*

    Open = {};

    *Close={A(0,7,7,-), D(1,6,7,A), B(2,3,5,D), **C(3,4,7,B)**}*
  - *Các con của C: G*
  - ***Xét G***
    - **g(G)** = g(C) + c[C,G] = 3 + 2 = 5 (g(m) = g(n) + c[m,n])
    - **G** không **thuộc Open; Closed**
      - ***Tính giá trị f(G) = g(G) + h(G) = 5 + 0 = 5***
      - ***Cập nhật cha của G: C***
      - ***Đưa G vào open: G(5,0,5,C)***
  - *Sắp xếp các phần tử trong open để trạng thái tốt nhất bên trái*

*Open{**G(5, 0, 5, C)**}* *; Close={A(0,7,7,-), D(1,6,7,A), B(2,3,5,D), **C(3,4,7,B)**}*



■ Bước 6:

- *Quay lại đầu vòng lặp while*
- *Lấy phần tử **G** ra khỏi Open đưa vào Close*

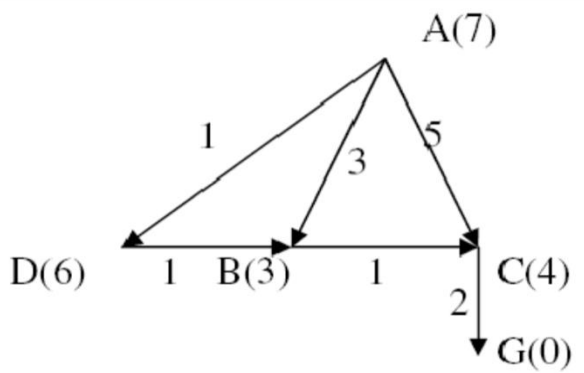
  Open = {};

  *Close = {A(0,7,7,-), D(1,6,7,A), B(2,3,5,D) , C(3,4,7,B), **G(5,0,5,C)}***
- *Xét **G(5,0,5,C) là trạng thái đích => giải thuật dừng lại***

Ta có đường đi

*Close = {A(0,7,7,-), D(1,6,7,A), B(2,3,5,D) , C(3,4,7,B), **G(5,0,5,C)}***

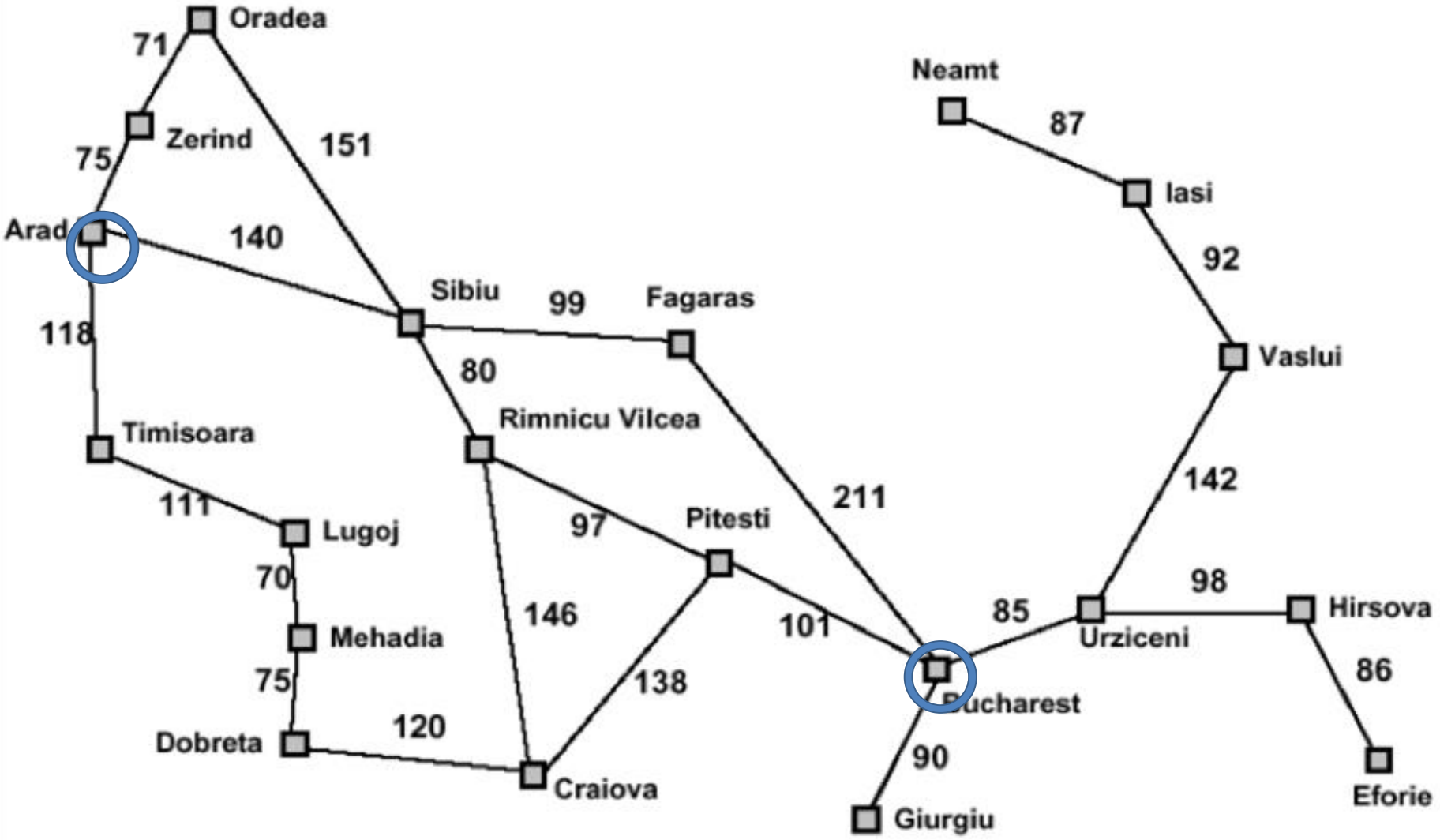
A => D => B => C => G



Giải thuật A*

- Bài toán tìm đường:
  - *Thành phố xuất phát: Arad*
  - *Thành phố đích: Bucharest*
  - *Các cạnh biểu diễn đường nối trực tiếp giữa hai thành phố, các con số ghi trên các cạnh là chi phí đi giữa hai thành phố.*
  - *Cột bên phải là khoảng cách Euclic từ các thành phố đến thành phố đích Bucharest.*
- Sử dụng phương pháp tìm kiếm A* (hàm ước lượng f(n) = g(n) + h(n), với g(n) là chi phí từ thành phố xuất phát đến n và h(n) là khoảng cách Euclic từ n đến đích)

Giải thuật A*



Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

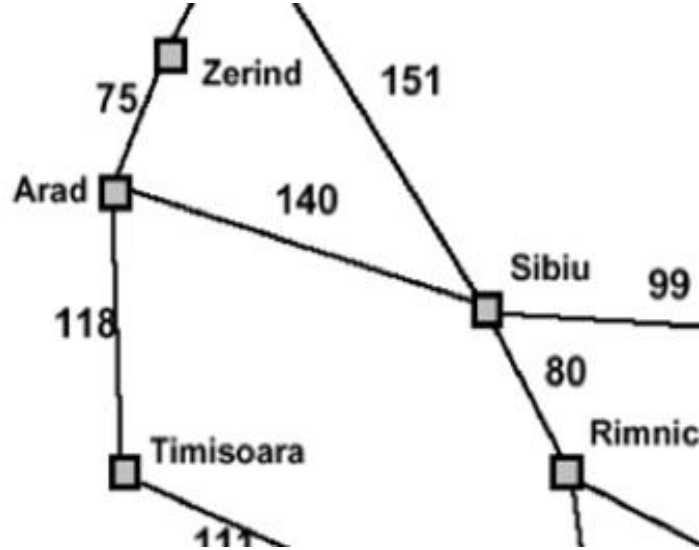
Mã giả giải thuật A*

```
g(n_o)=0; f(n_o)=h(n_o);
open:=[n_o]; closed:=[];
while open<>[] do
    loại n bên trái của open và dưa n vào closed;
    if (n là một đích) then thành công, thoát
    else
        Sinh các con m của n;
        For m thuộc con(n) do
            g(m)=g(n)+c[n,m];
            If m không thuộc open hay closed then
                f(m)=g(m)+h(m); cha(m)=n;  Bỏ m vào open;
            If m thuộc open (tồn tại m' thuộc open, sao cho m=m')  then
                If g(m)<g(m') then   g(m')=g(m); f(m')=g(m')+h(m');  Cha(m')=n;
            If m thuộc closed (tồn tại m' thuộc closed, sao cho m=m') then
                If g(m)<g(m') then  f(m)=g(m)+h(m); cha(m)=n;
                                    Đưa m vào open; loại m' khỏi closed;
        Sắp xếp open để t.thái tốt nhất nằm bên trái;
```

- Mỗi trạng thái n tùy ý sẽ gồm: (g(n), h(n), f(n), cha(n))
- **Bước 1**:

  Open={ Arad (0,366,366,-)};

  Close={}



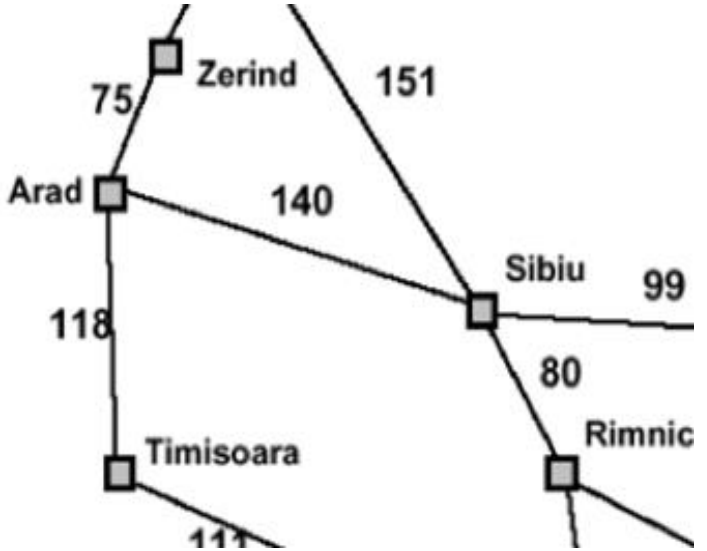
- **Bước 2:**
  - *Các con của Arad: Timisoara, Sibu, Zerind*
  - ***Xét Timisoara***
    - **g(Timisoara) = g(Arad) + c[Arad, Timisoara] = 0+ 118= 118 (do đề bài cung cấp) (g(m) = g(n) + c[m,n])**
    - **Timisoara không thuộc Open; Close**
      - ***Tính giá trị f(Timisoara) = g(Timisoara) + h(Timisoara)***

        = 118 + 329 = 447
      - ***Cập nhật cha của Timisoara : Arad***
      - ***Đưa Timisoara vào open: Timisoara(118,329,447,Arad)***

- **Bước 2:**
  - **…..**
  - ***Xét Sibiu***
    - **g(Sibiu) = g(Arad) + c[Arad, Sibiu]**

      = 0+ 140= 140 (do đề bài cung cấp)

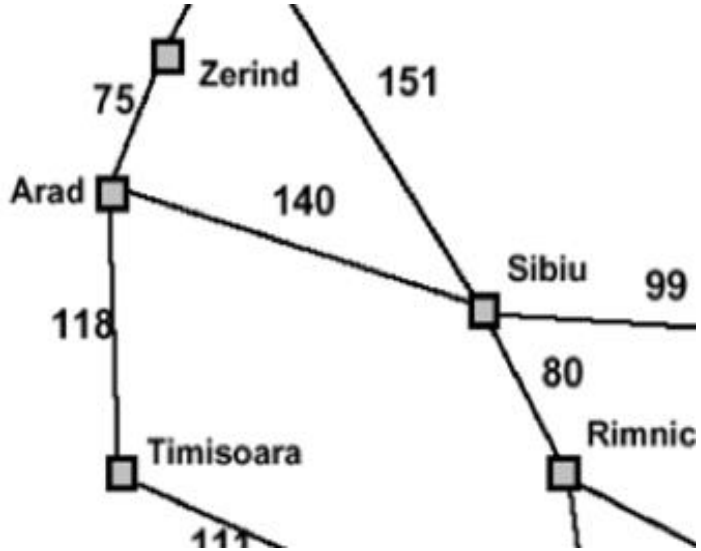
      (g(m) = g(n) + c[m,n])
    - **Sibiu không thuộc Open; Close**
      - ***Tính giá trị f(Sibiu) = g(Sibiu) + h(Sibiu) = 140 + 253 = 393***
      - ***Cập nhật cha của Sibiu : Arad***
      - ***Đưa Sibiu vào open: Sibiu (140,253,393,Arad)***



- **Bước 2:**
  - **…..**
  - ***Xét Zerind***
    - **g(Zerind) = g(Arad) + c[Arad, Zerind]**
      = 0+ 75= 75 (do đề bài cung cấp)
      (g(m) = g(n) + c[m,n])
    - Zerind **không thuộc Open; Close**
      - ***Tính giá trị f(Zerind) = g(Zerind) + h(Zerind) = 75 + 374 = 449***
      - ***Cập nhật cha của Zerind : Arad***
      - ***Đưa Zerind vào open: Zerind(75,374,449,Arad)***
  - *Sắp xếp các phần tử trong open để trạng thái tốt nhất bên trái*

Open = {Sibiu (140,253,393,Arad), Timisoara(118,329,447,Arad), Zerind(75,374,449,Arad)};

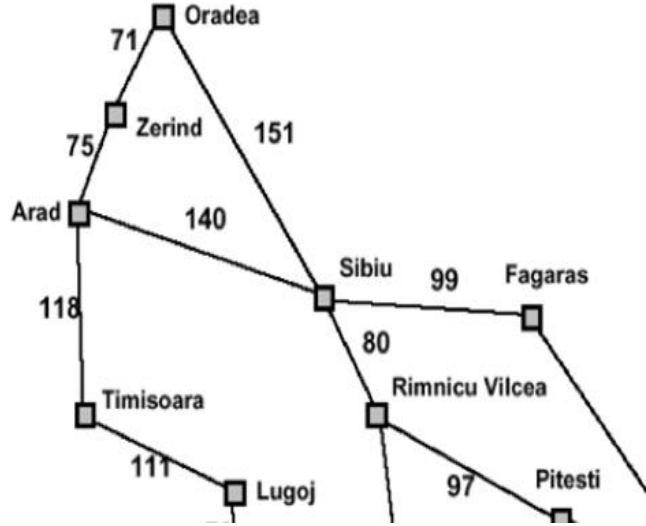
Close = {Arad (0,366,366,-)}



- **Bước 3**
  - *Quay lại đầu vòng lặp while*
  - *Lấy phần tử* ***Sibiu*** *ra khỏi Open đưa vào Closed*

*Open = {**Timisoara(118,329,447,Arad)**, Zerind**(75,374,449,Arad)**};*

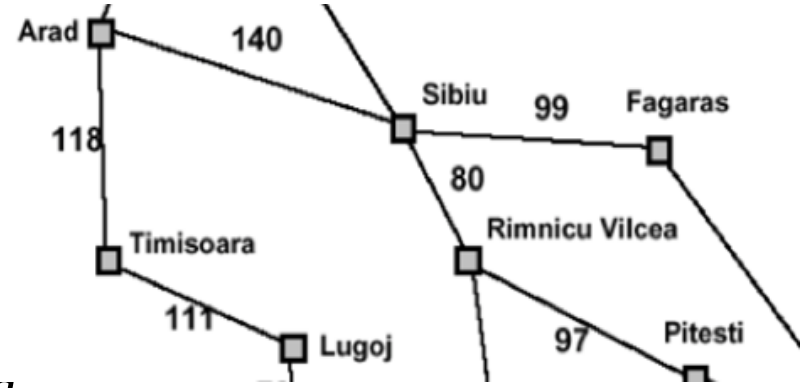
Close = {Arad (0,366,366,-), ***Sibiu (140,253,393,Arad)****}*



  - ***Xét Rimnicu***
    - **g(Rimnicu) = g(Sibiu) + c[Sibiu, Rimnicu] = 140+ 80= 220**

      (g(m) = g(n) + c[m,n])
    - **Rimnicu không thuộc Open; Close**
      - ***Tính giá trị f(Rimnicu) = g(Rimnicu) + h(Rimnicu)***

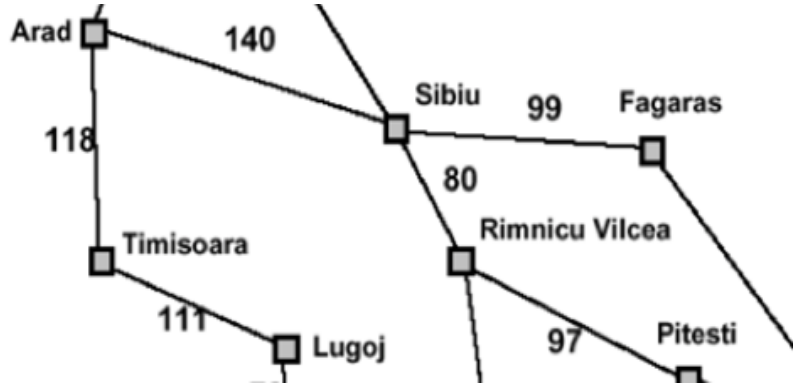
        = 220 + 193 = 413
      - ***Cập nhật cha của Rimnicu : Sibiu***
      - ***Đưa Rimnicu vào open: Rimnicu(220,193,413,Sibiu)***



- **Bước 3**
  - .....
  - *Các con của* ***Sibiu*** *: Rimnicu Vilces, Fagaras, Arad, Oradea*
  - ***Xét Fagaras***
    - **g(Fagaras) = g(Sibiu) + c[Sibiu, Fagaras] = 140+ 99= 239 (g(m) = g(n) + c[m,n])**
    - **Fagaras không thuộc Open; Close**
      - ***Tính giá trị f(Fagaras) = g(Fagaras) + h(Fagaras) = 239 + 178 = 417***
      - ***Cập nhật cha của Fagaras : Sibiu***
      - ***Đưa Fagaras vào open: Fagaras(239,178,417,Sibiu)***
  - *Sắp xếp các phần tử trong open để trạng thái tốt nhất bên trái*

*Open = {****Rimnicu(220,193,413,Sibiu), Fagaras(239,178,417,Sibiu),***
Timisoara(118,329,447,Arad), *Zerind****(75,374,449,Arad)}****;*

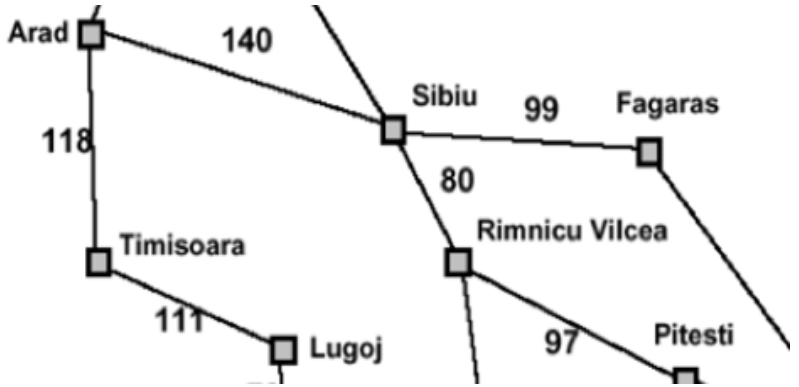
Close = {Arad (0,366,366,-), ***Sibiu (140,253,393,Arad)*** *}*



- **Bước 3**
  - …..
  - *Các con của* ***Sibiu*** *: Rimnicu, Fagaras, Arad, Oradea*
  - ***Xét Oradea***
    - **g(Oradea) = g(Sibiu) + c[Sibiu, Oradea] = 140+ 151= 291 (g(m) = g(n) + c[m,n])**
    - **Oradea không thuộc Open; Close**
      - ***Tính giá trị f(Oradea) = g(Oradea) + h(Oradea) = 291 + 380 = 671***
      - ***Cập nhật cha của Oradea : Sibiu***
      - ***Đưa Oradea vào open: Oradea(291,380,671,Sibiu)***
  - *Sắp xếp các phần tử trong open để trạng thái tốt nhất bên trái*

*Open = {**Rimnicu(220,193,413,Sibiu), Fagaras(239,178,417,Sibiu), Timisoara(118,329,447,Arad), Zerind(75,374,449,Arad), Oradea(291,380,671,Sibiu)}*;

Close = {Arad (0,366,366,-), ***Sibiu (140,253,393,Arad)*** *}*



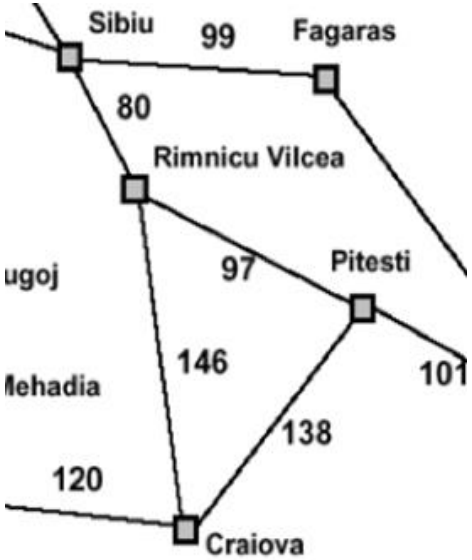
- **Bước 3**
  - .....
  - *Các con của* ***Sibiu*** *: Rimnicu, Fagaras, Arad, Orade*
  - ***Xét*** *Arad*
    - **g(**Arad**) = g(Sibiu) + c[Sibiu,** Arad**] = 140+ 140= 280 (g(m) = g(n) + c[m,n])**
    - Arad **thuộc Close**
      - ***G(Arad)=280> G(Arad') => ko làm gì cả (ko đưa vào open hay closed)***
  - *Sắp xếp các phần tử trong Open để trạng thái tốt nhất bên trái*

*Open = {**Rimnicu(220,193,413,Sibiu), Fagaras(239,178,417,Sibiu),***
***Timisoara(118,329,447,Arad),** Zerind**(75,374,449,Arad), Oradea(291,380,671,Sibiu)}*;

Close = { Arad (0,366,366,-), ***Sibiu (140,253,393,Arad)*** *}*

Bước 4

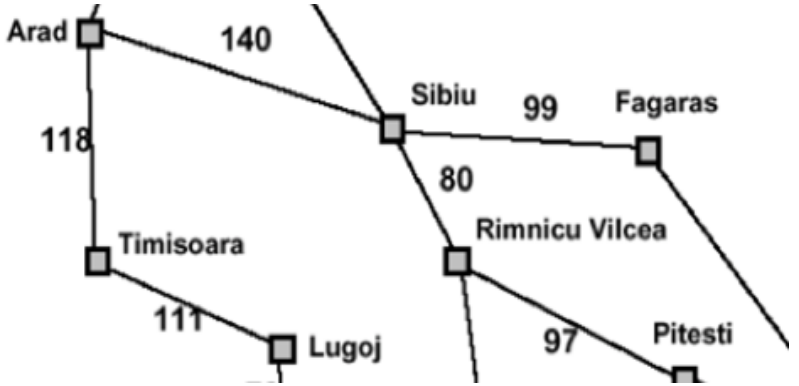
- *Quay lại đầu vòng lặp while*
- *Lấy phần tử* ***Rimnicu*** *ra khỏi Open đưa vào Closed*



*Open = {**Fagaras(239,178,417,Sibiu),Timisoara(118,329,447,Arad),***
*Zerind**(75,374,449,Arad), Oradea(291,380,671,Sibiu)}**;*

Close = { Arad (0,366,366,-), ***Sibiu (140,253,393,Arad), Rimnicu(220,193,413,Sibiu*** *}*

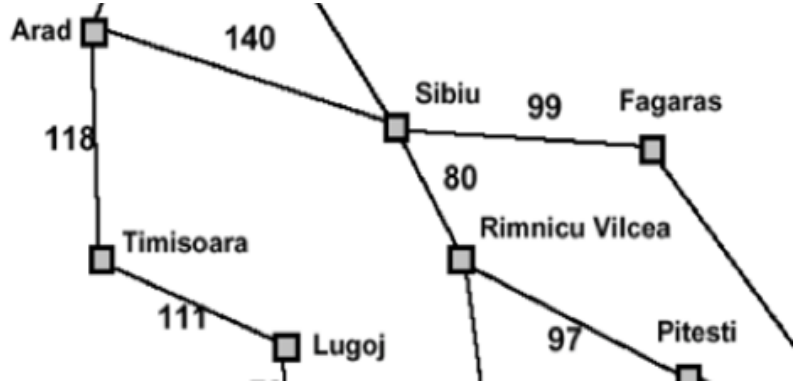
- *Các con của* ***Rimnicu*** *: Craiova, Pitesti,* ***Sibiu***
- ***Xét Craiova***
  - **g(Craiova) = g(Rimnicu) + c[Rimnicu, Craiova] = 220 + 146 = 366**
  - **Craiova không thuộc Open; Close**
    - ***Tính giá trị f(Craiova) = g(Craiova) + h(Craiova) = 366 + 160 = 526***
    - ***Cập nhật cha của Craiova là Rimnicu***
    - ***Đưa Craiova vào Open: Craiova(366,160,526, Rimnicu)***



- **Bước 4**
  - ….
  - *Các con của* ***Rimnicu*** *: Craiova, Pitesti,* ***Sibiu***
  - ***Xét Pitesti***
    - **g(Pitesti) = g(Rimnicu) + c[Rimnicu, Pitesti] = 220 + 97 = 317**
    - **Pitesti không thuộc Open; Close**
      - ***Tính giá trị f(Pitesti) = g(Pitesti) + h(Pitesti) = 317 + 98 = 415***
      - ***Cập nhật cha của Pitesti là Rimnicu***
      - ***Đưa Pitesti vào Open: Pitesti (317,98,415, Rimnicu)***
  - *Sắp xếp các phần tử trong open để trạng thái tốt nhất bên trái*

*Open = {**Pitesti (317,98,415, Rimnicu), Fagaras(239,178,417,Sibiu),Timisoara(118,329,447,Arad),*
Zerind(75,374,449,Arad), Craiova(366,160,526, Rimnicu), Oradea(291,380,671,Sibiu)};

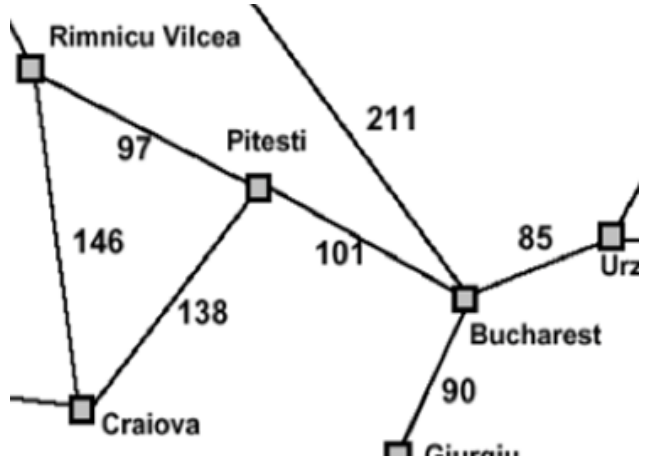
Close = { Arad (0,366,366,-), ***Sibiu (140,253,393,Arad), Rimnicu(220,193,413,Sibiu*** *}*



- **Bước 4**
  - ….
  - *Các con của* ***Rimnicu*** *: Craiova, Pitesti,* ***Sibiu***
  - ***Xét Sibiu***
    - **g(Sibiu) = g(Rimnicu) + c[Rimnicu, Sibiu] = 220 + 80 = 400**
    - **Sibiu thuộc Close**
      - ***Tính giá trị g(Sibiu) =400 > g(Sibiu')=140***

      => không làm gì cả (ko đưa vào Open hay Close)
  - *Sắp xếp các phần tử trong open để trạng thái tốt nhất bên trái*

*Open = {****Pitesti (317,98,415, Rimnicu),***
Fagaras(239,178,417,Sibiu),Timisoara(118,329,447,Arad),
*Zerind****(75,374,449,Arad), Craiova(366,160,526, Rimnicu), Oradea(291,380,671,Sibiu)}****;*
Close = { Arad (0,366,366,-), ***Sibiu (140,253,393,Arad), Rimnicu(220,193,413,Sibiu*** *}*



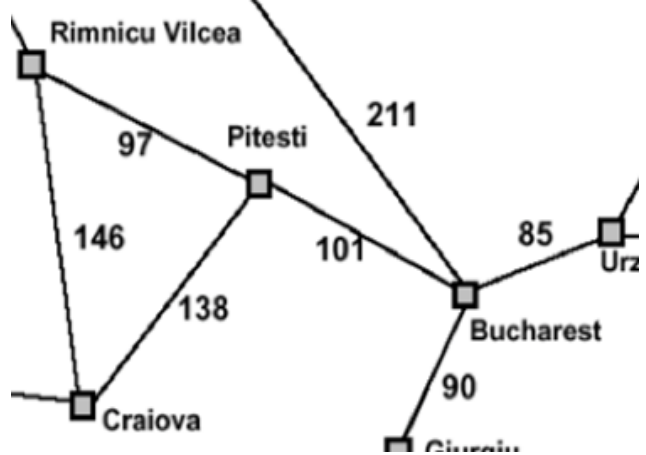
- **Bước 5**
  - *Quay lại đầu vòng lặp while*
  - *Lấy phần tử* ***Pitesti*** *ra khỏi Open đưa vào Closed*

*Open = {****Fagaras(239,178,417,Sibiu),Timisoara(118,329,447,Arad),***

*Zerind****(75,374,449,Arad), Craiova(366,160,526, Rimnicu), Oradea(291,380,671,Sibiu)}****;*

Close = {Arad (0,366,366,-), Sibiu (140,253,393,Arad), Rimnicu(220,193,413,Sibiu), Pitesti (317,98,415, Rimnicu)}

  - *Các con của* ***Pitesti*** *: Craiova, Bucharest, Rimnicu*
  - ***Xét Craiova***
    - **g(**Craiova**) = g(Pitesti) + c[Pitesti ,**Craiova**] = 317 + 138 = 455**
    - **Craiova thuộc Open;**
    - **g(Craiova)=455 > g(Craiova')=** 366 => Không làm gì cả



- **Bước 5**
  - *Quay lại đầu vòng lặp while*
  - *Lấy phần tử* ***Pitesti*** *ra khỏi Open đưa vào Closed*

*Open = {**Fagaras(239,178,417,Sibiu),Timisoara(118,329,447,Arad),***
*Zerind**(75,374,449,Arad), Craiova(366,160,526, Rimnicu), Oradea(291,380,671,Sibiu)};***

Close = { Arad (0,366,366,-), Sibiu (140,253,393,Arad), Rimnicu(220,193,413,Sibiu), Pitesti (317,98,415, Rimnicu)}

Các con của ***Pitesti*** *: Craiova, Bucharest, Rimnicu*

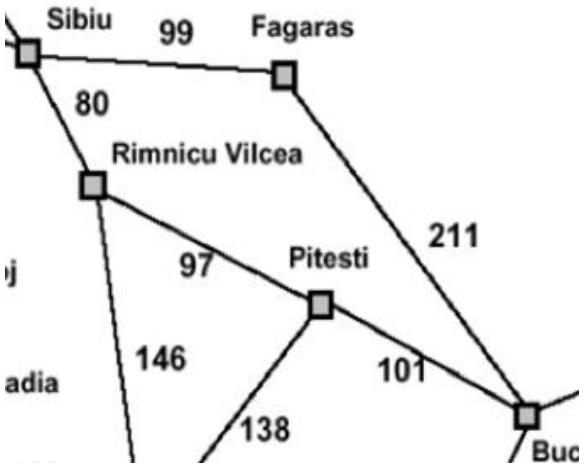
- ***Xét*** *Bucharest*
  - **g(**Bucharest **) = g(Pitesti) + c[Pitesti,** Bucharest **] = 317 + 101 = 418**
  - Bucharest **không thuộc Open; Closed**
    - ***Tính giá trị f(****Bucharest ****) = g(****Bucharest ****) + h(****Bucharest ****) = 418 +0 = 418***
    - ***Cập nhật cha của*** *Bucharest* ***: Pitesti***
    - ***Đưa*** *Bucharest* ***vào open:*** *Bucharest* ***(418,0,418,Pitesti)***

- **Bước 5**
  - …
  - *Các con của* ***Pitesti*** *: Craiova, Bucharest,* ***Rimnicu***
  - ***Xét Rimnicu***
    - **g(Rimnicu) = g(Pitesti) + c[Pitesti , Rimnicu] = 317 + 138 = 455**
    - **Rimnicu thuộc Closed**
    - **Xét g(Rimnicu)=455> g(Rimnicu ‘)=220**

      => Không làm gì cả

Open *= {Fagaras(239,178,417,Sibiu),Bucharest(418,0,418,Pitesti),Timisoara(118,329,447,Arad), Zerind(75,374,449,Arad), Craiova(368,160,528, Rimnicu), Oradea(291,380,671,Sibiu)};*

Close *= { Arad (0,366,366,-), Sibiu (140,253,393,Arad), Rimnicu(220,193,413,Sibiu), Pitesti (317,98,415, Rimnicu) }*

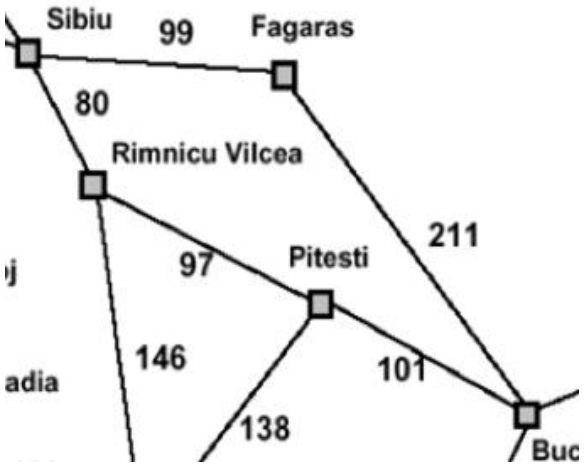


- **Bước 6**
  - *Quay lại đầu vòng lặp while*
  - *Lấy phần tử* ***Fagaras*** *ra khỏi Open đưa vào Closed*

Open *= {Bucharest(418,0,418,Pitesti),Timisoara(118,329,447,Arad), Zerind(75,374,449,Arad), Craiova(368,160,528, Rimnicu),Oradea(291,380,671,Sibiu)};*

Close *= { Arad (0,366,366,-), Sibiu (140,253,393,Arad), Rimnicu(220,193,413,Sibiu), Pitesti (317,98,415, Rimnicu),* ***Fagaras(239,178,417,Sibiu)) }***

  - *Các con của* ***Fagaras*** *: Bucharest, Sibiu*
  - ***Xét Sibiu***
    - **g(Sibiu) = g(Fagaras) + c[Fagaras , Sibiu] = 239 + 99 = 338**
    - **Sibiu thuộc Closed;**
    - **g(Sibiu)=338 > g(Sibiu ')=** 140 => Không làm gì cả



- **Bước 6**
  - *Quay lại đầu vòng lặp while*
  - *Lấy phần tử **Fagaras** ra khỏi Open đưa vào Closed*

***Open** = {Bucharest(418,0,418,Pitesti),Timisoara(118,329,447,Arad), Zerind(75,374,449,Arad), Craiova(368,160,528, Rimnicu),Oradea(291,380,671,Sibiu)};*

Close** = { Arad (0,366,366,-), Sibiu (140,253,393,Arad), Rimnicu(220,193,413,Sibiu), Pitesti (317,98,415, Rimnicu), **Fagaras(239,178,417,Sibiu)) }

  - *Các con của **Fagaras** : Bucharest, Sibiu*
  - ***Xét Bucharest***
    - **g(Bucharest) = g(Fagaras) + c[Fagaras , Bucharest] = 239 + 211 = 450**
    - **Bucharest thuộc Open;**
    - **g(Bucharest)=450> g(Bucharest ')= 418** => Không làm gì cả

- **Bước 7**
  - *Quay lại đầu vòng lặp while*
  - *Lấy phần tử* ***Bucharest*** *ra khỏi Open đưa vào Closed*

Open *= {Timisoara(118,329,447,Arad),*

Zerind(75,374,449,Arad), Craiova(368,160,528, Rimnicu),Oradea(291,380,671,Sibiu)};

Close *= { Arad (0,366,366,-), Sibiu (140,253,393,Arad), Rimnicu(220,193,413,Sibiu), Pitesti (317,98,415, Rimnicu), Fagaras(239,178,417,Sibiu)****, Bucharest(418,0,418,Pitesti)}***

- ***Xét Bucharest là trạng thái đích, giải thuật dừng lại.***

=> Đường đi được tìm bằng cách truy ngược cha của nút gốc và các nút kế tiếp:

Bucharest(418,0,418,Pitesti) *=> Pitesti (317,98,415, Rimnicu) => Rimnicu(220,193,413,Sibiu) => Sibiu (140,253,393,Arad) => Arad (0,366,366,-)*

Lưu ý: *Các trạng thái trong closed là trạng thái đã xét qua, tuy nhiên* ***không phải tất cả*** *các trạng thái trong closed đều góp phần trong đường đi lời giải (Ví dụ đỉnh Fagaras trong bài toán này không thuộc đường đi lời giải)*

Giải thuật leo đồi

Ý tưởng của giải thuật leo đồi:

- Chọn một trạng thái tốt hơn trạng thái hiện hành để mở rộng. Nếu không, thuật toán phải dừng lại.
- Để biết trạng thái nào tốt hơn, chúng ta sẽ sử dụng hàm đánh giá Heuristic f(n).
- Giải thuật leo đồi sẽ không lưu trữ tất cả các trạng thái con mà chỉ lưu đúng một trạng thái được chọn nếu có.

Các loại giải thuật leo đồi

1. **Simple Hill climbing (Leo đồi đơn giản)**
2. **Steepest-Ascent Hill climbing (Leo đồi dốc đứng)**
3. **Stochastic hill climbing (Leo đồi ngẫu nhiên)**

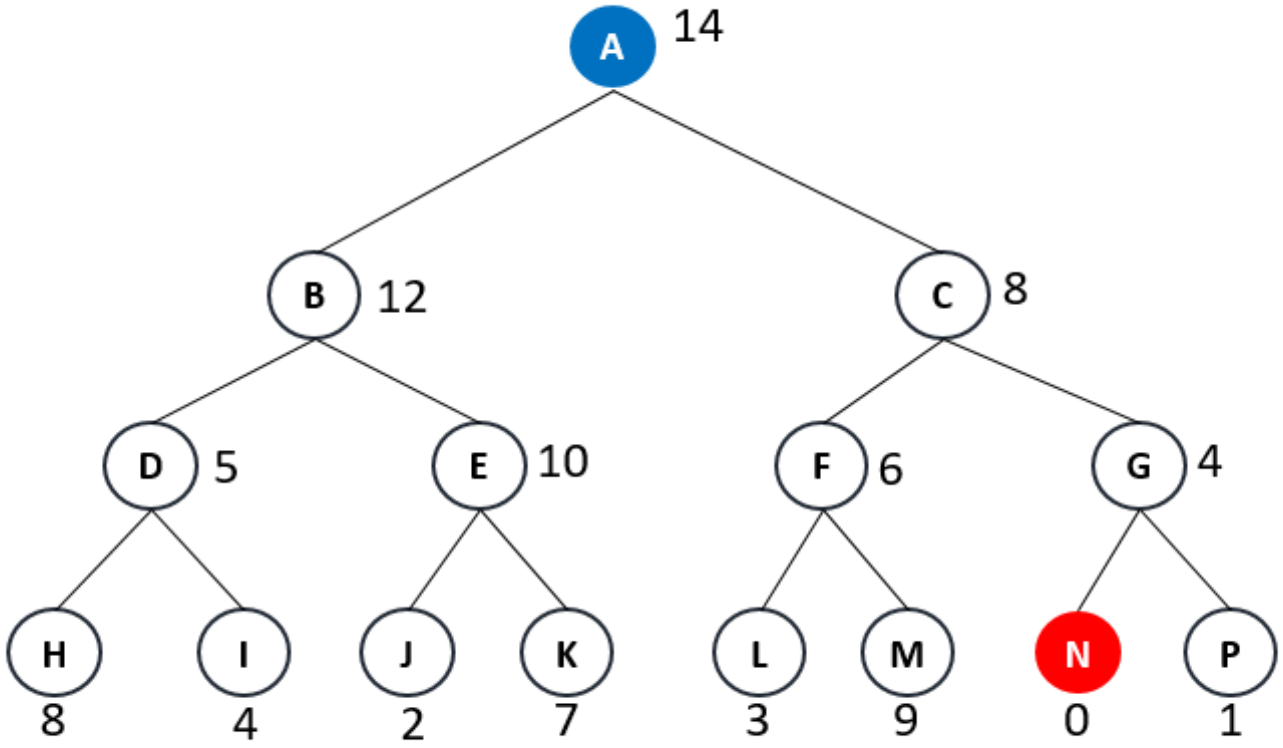
Simple Hill climbing (Leo đồi đơn giản)

1. Xét trạng thái bắt đầu:
- ❑ Nếu trạng thái đích, giải thuật dừng
- ❑ Ngược lại, thiết lập trạng thái bắt đầu như trạng thái hiện tại.

2. Lặp đến khi: gặp trạng thái đích hoặc không còn toán tử (Operators) nào có thể áp dụng được vào trạng thái hiện tại.
- ❑ Chọn toán tử phù hợp để sinh ra trạng thái mới từ trạng thái hiện tại.
- ❑ Xét trạng thái mới sinh ra:
  - Nếu là trạng thái đích → giải thuật dừng.
  - Nếu không là trạng thái đích và tốt hơn trạng thái hiện tại, chọn nó là trạng thái hiện tại.
  - Nếu không là trạng thái đích và không tốt hơn trạng thái hiện tại, thực hiện lặp lại bước 2.

Simple Hill climbing (Leo đồi đơn giản)

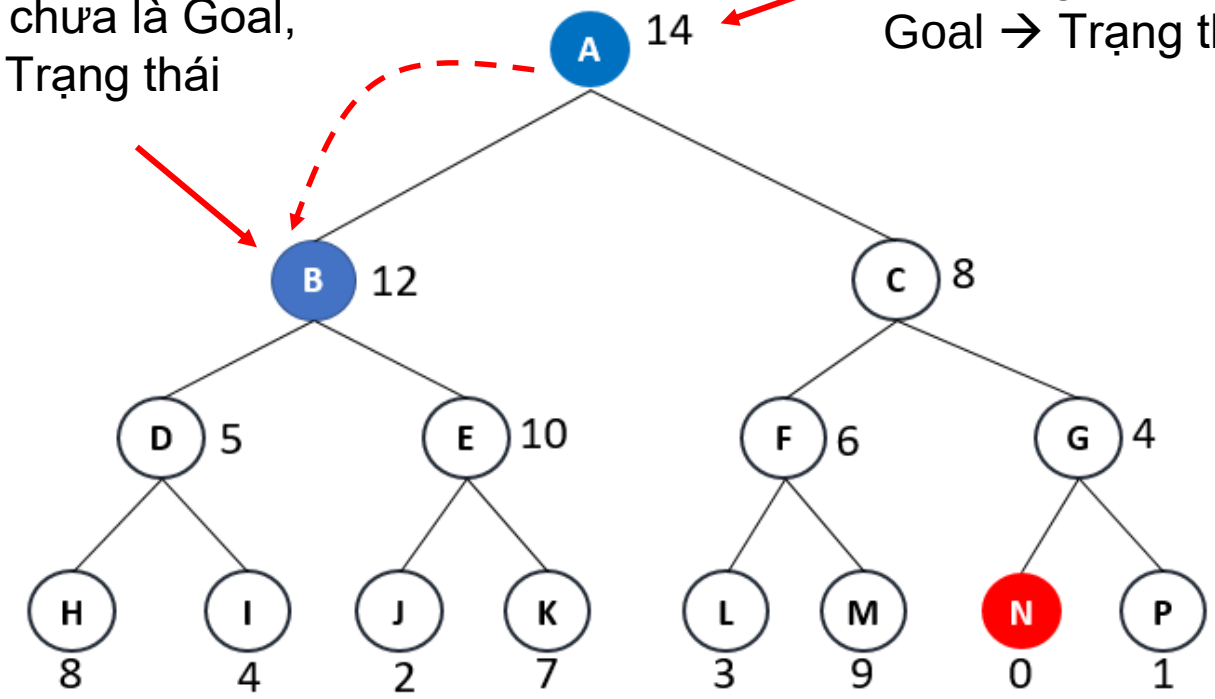
Cho đồ thị như bên dưới, nếu áp dụng giải thuật tìm kiếm leo đồi đơn giản và giá trị f(n) được tính sẵn để bên cạnh các đỉnh. Cho biết thứ tự duyệt các nút, biết trạng thái bắt đầu là A và kết thúc là N.



Simple Hill climbing (Leo đồi đơn giản)

Sinh ra trạng thái B:
Trạng thái B chưa là Goal, tốt hơn A → Trạng thái hiện tại là B

Xét trạng thái A → A chưa là Goal → Trạng thái hiện tại

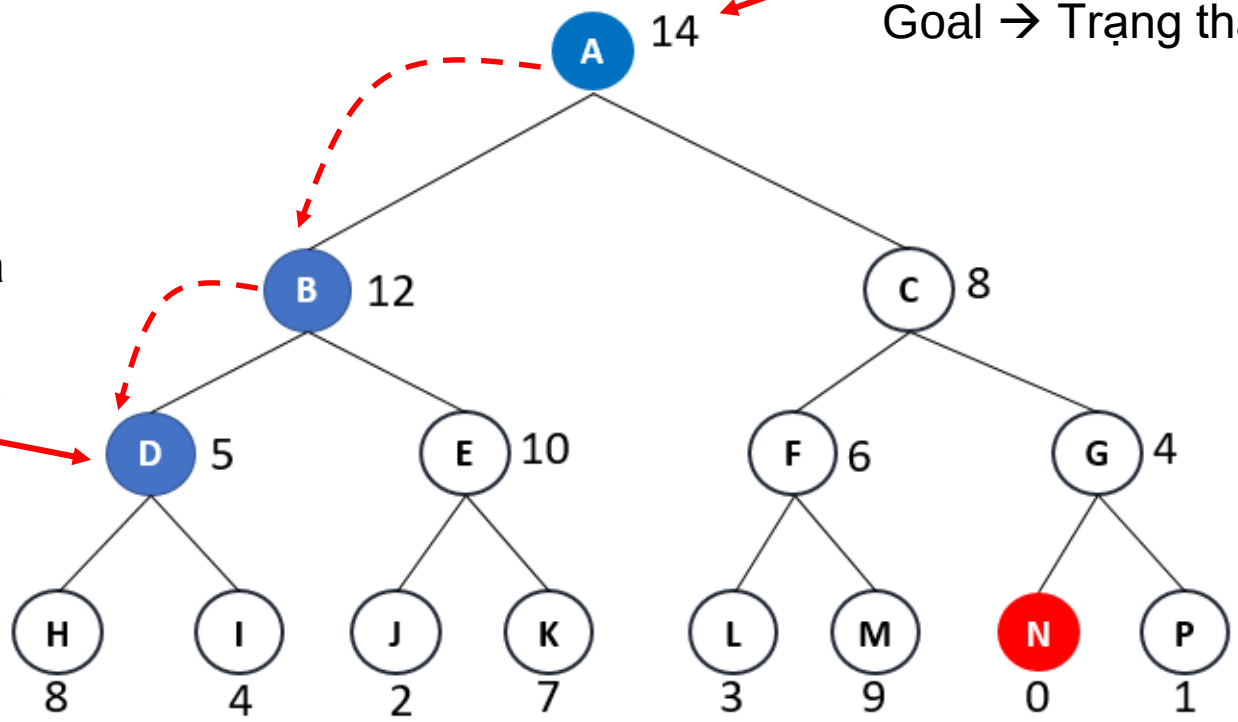


Thứ tự duyệt: A, B

Simple Hill climbing (Leo đồi đơn giản)

Xét trạng thái A → A chưa là Goal → Trạng thái hiện tại

Sinh ra trạng thái D: Trạng thái D chưa là Goal, tốt hơn B → Trạng thái hiện tại là D

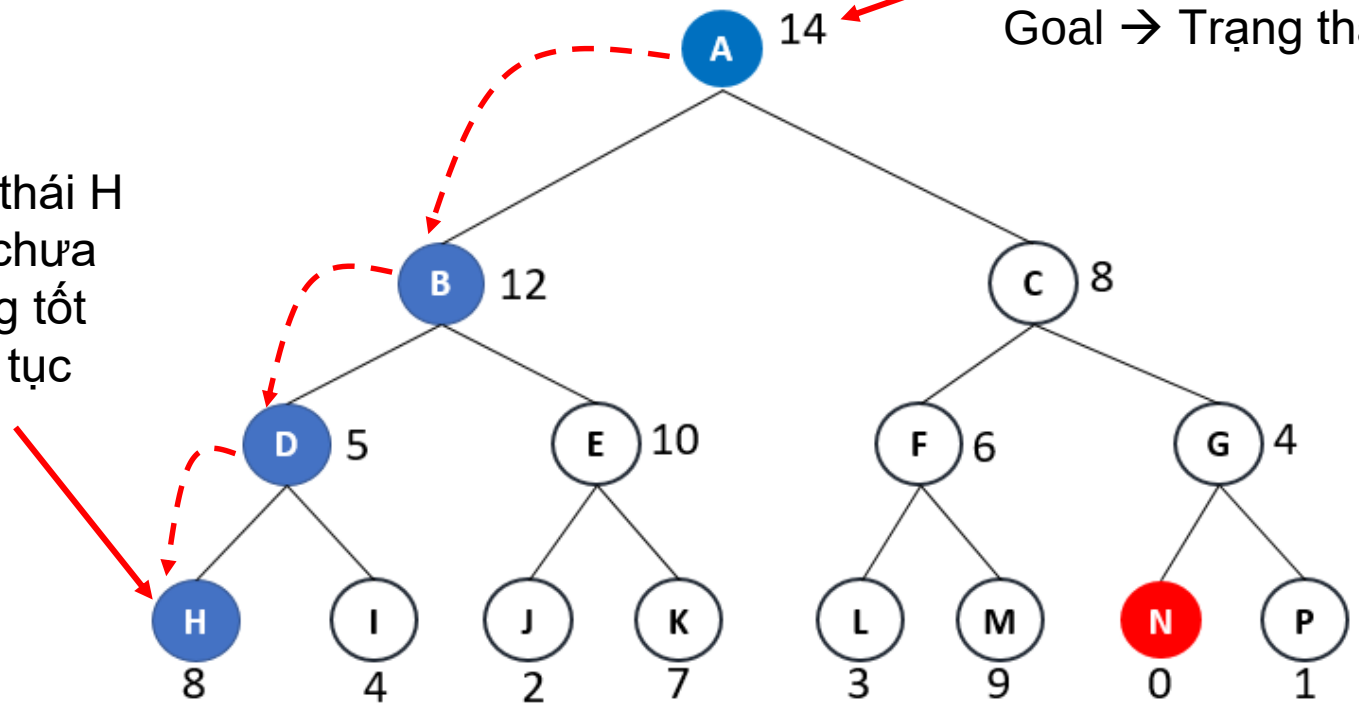


Thứ tự duyệt: A, B, D

Simple Hill climbing (Leo đồi đơn giản)

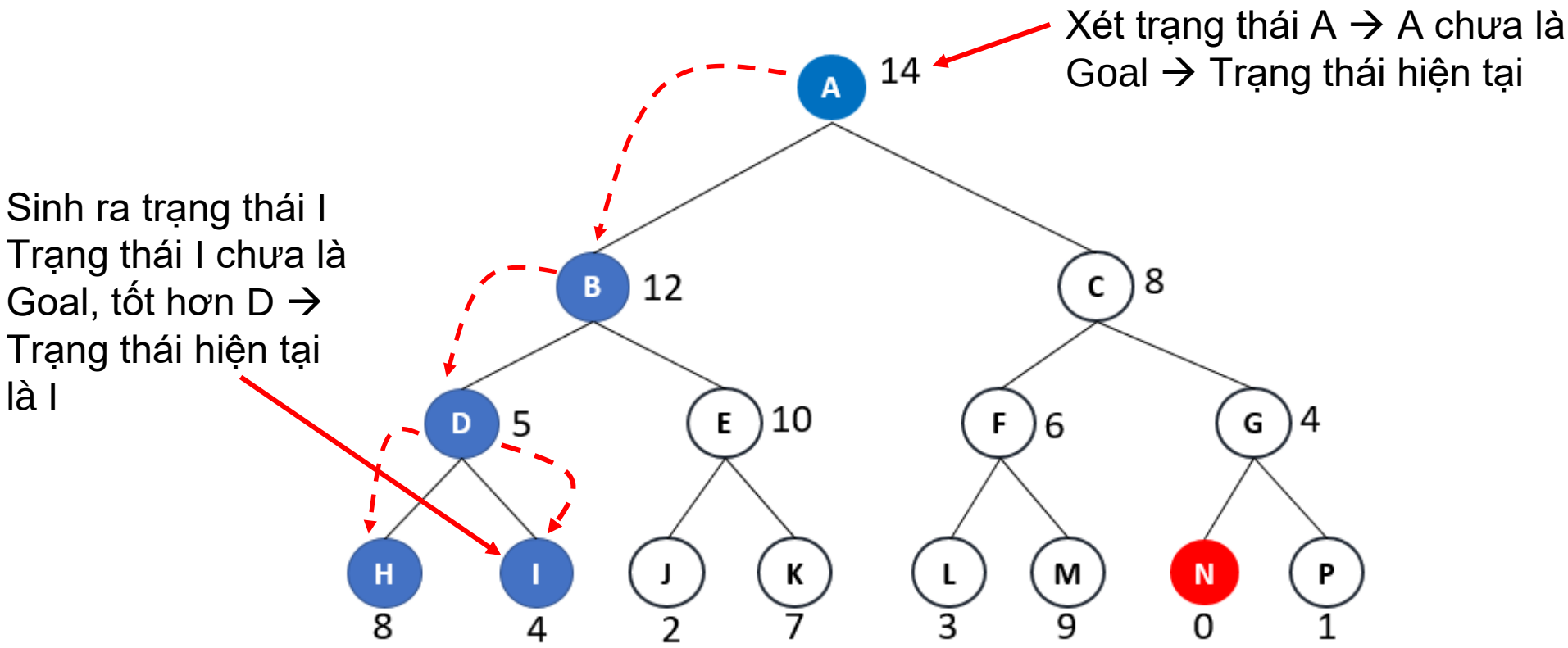
Xét trạng thái A → A chưa là Goal → Trạng thái hiện tại

Sinh ra trạng thái H
Trạng thái H chưa là Goal, không tốt hơn D → tiếp tục sinh con I



Thứ tự duyệt: A, B, D, H

Simple Hill climbing (Leo đồi đơn giản)



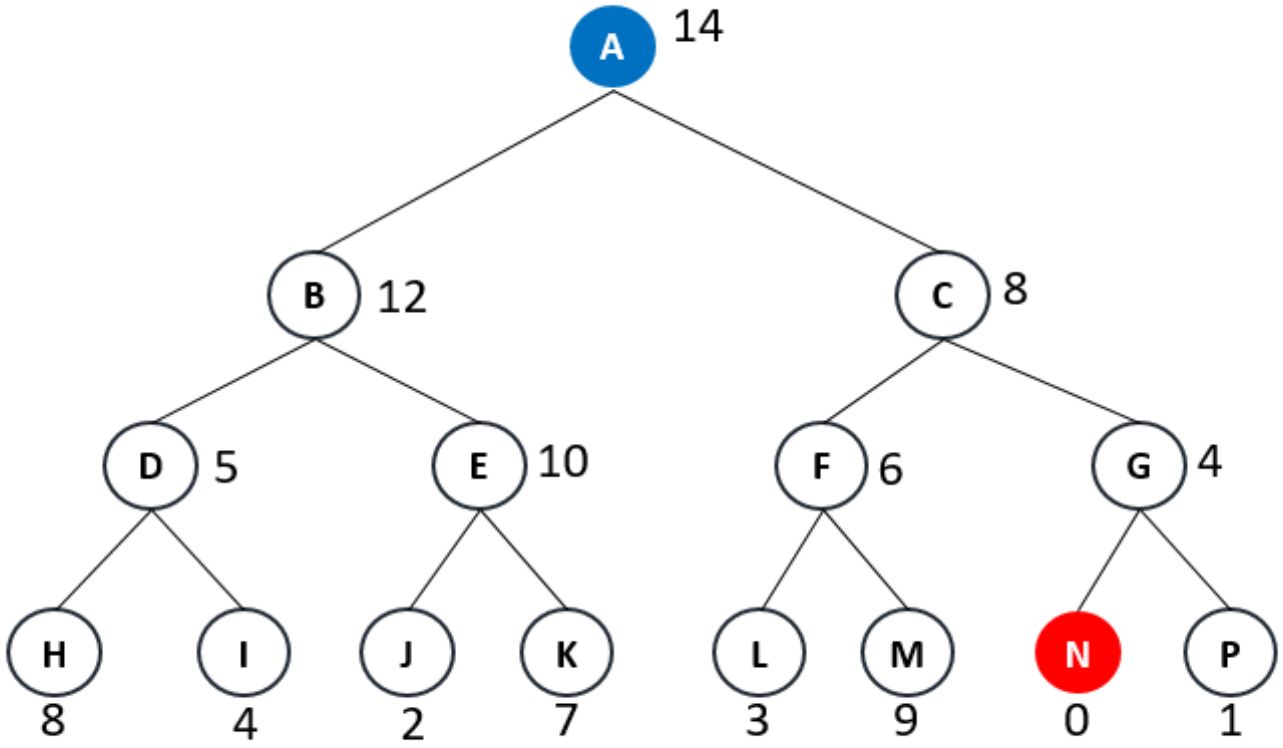
Thứ tự duyệt: A, B, D, H, I (giải thuật dừng)

Steepest-Ascent Hill climbing (Leo đồi dốc đứng)

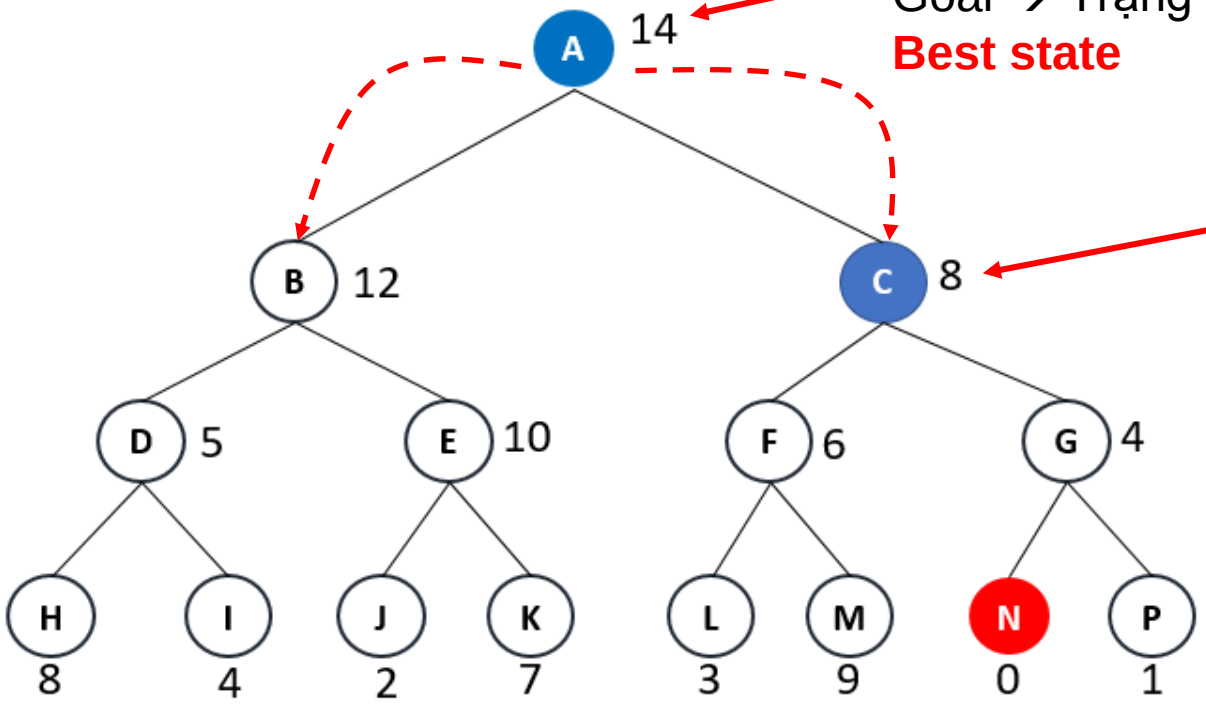
1. Xét trạng thái bắt đầu:
- ❑ Nếu trạng thái đích, giải thuật dừng
- ❑ Ngược lại, thiết lập trạng thái bắt đầu như trạng thái hiện tại.
2. Lặp đến khi: gặp trạng thái đích hoặc không còn toán tử (Operators) nào có thể áp dụng được vào trạng thái hiện tại.
- ❑ Khởi tạo trạng thái tốt nhất là trạng thái hiện tại. Chọn toán tử phù hợp để sinh ra các trạng thái mới từ trạng thái hiện tại.
- ❑ Với tập trạng thái mới được sinh ra, chọn trạng thái tốt nhất:
  - Nếu trạng thái này là đích → giải thuật dừng.
  - Nếu trạng thái này không là trạng thái đích và tốt hơn trạng thái tốt nhất, chọn nó là trạng thái tốt nhất.
- ❑ Thiết lập trạng thái tốt nhất là trạng thái hiện tại và trở lại bước lặp số 2.

Steepest-Ascent Hill climbing (Leo đồi dốc đứng)

Cho đồ thị như bên dưới, nếu áp dụng giải thuật tìm kiếm leo đồi dốc đứng và giá trị f(n) được tính sẵn để bên cạnh các đỉnh. Cho biết thứ tự duyệt các nút, biết trạng thái bắt đầu là A và kết thúc là N.



Steepest-Ascent Hill climbing (Leo đồi dốc đứng)

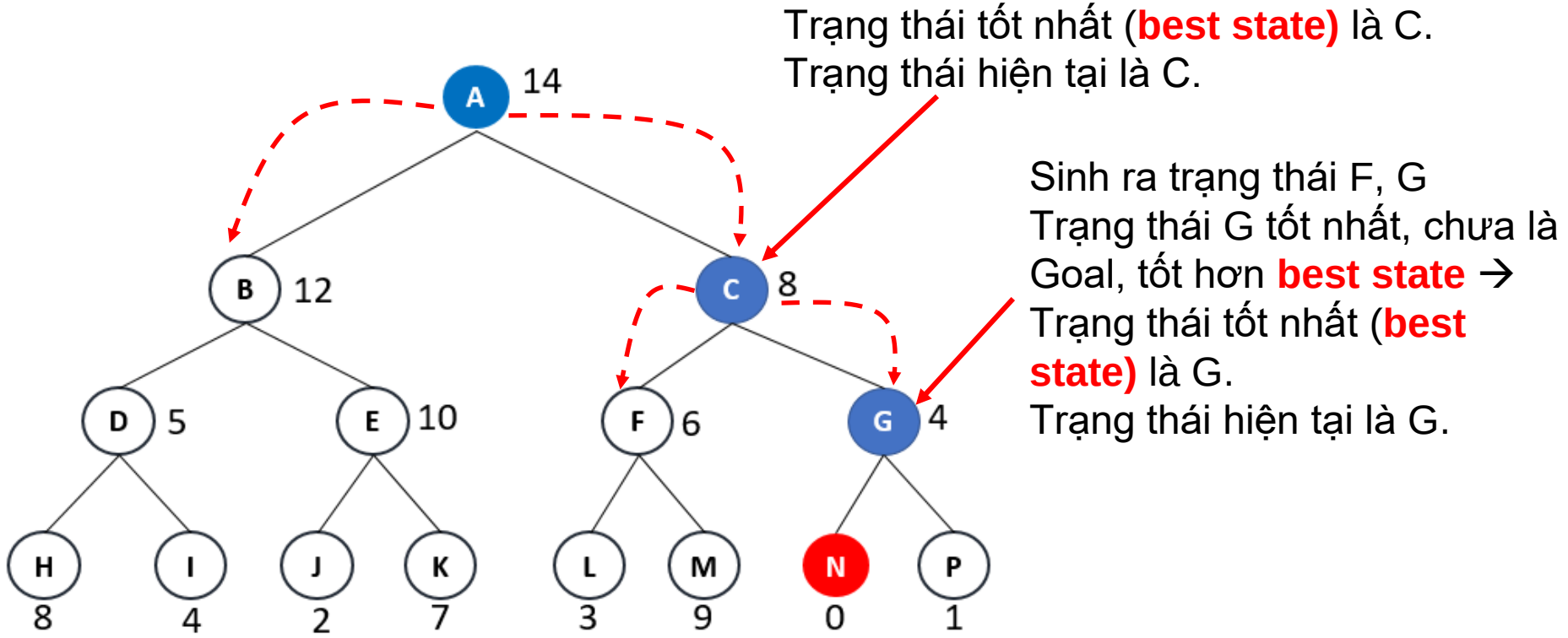


Xét trạng thái A → A chưa là Goal → Trạng thái hiện tại → **Best state**

Sinh ra trạng thái B, C
Trạng thái C tốt nhất, chưa là Goal, tốt hơn **best state** → Trạng thái tốt nhất (**best state)** là C.
Trạng thái hiện tại là C.

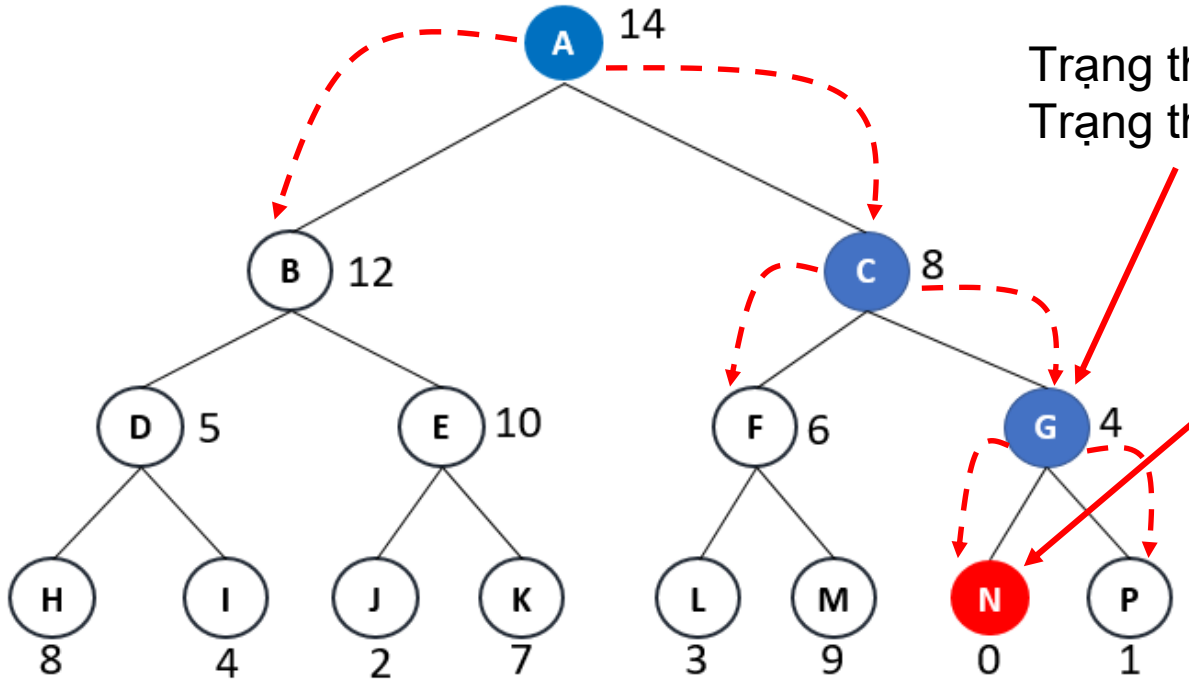
Thứ tự duyệt: A, C

Steepest-Ascent Hill climbing (Leo đồi dốc đứng)



Thứ tự duyệt: A, C, G

Steepest-Ascent Hill climbing (Leo đồi dốc đứng)



Trạng thái tốt nhất (**best state)** là G.
Trạng thái hiện tại là G.

Sinh ra trạng thái N, P
Trạng thái N tốt nhất và là Goal → Giải thuật dừng

Thứ tự duyệt: A, C, G, N

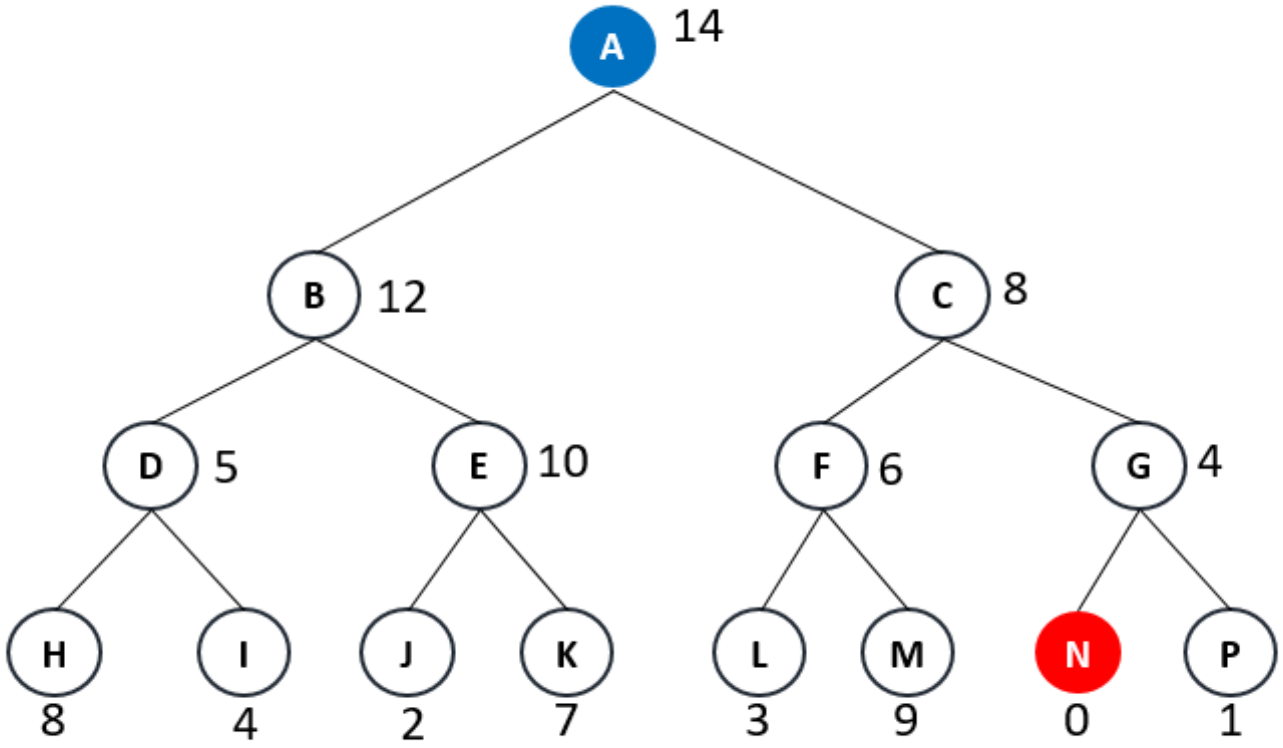
Stochastic hill climbing (Leo đồi ngẫu nhiên)

1. Xét trạng thái bắt đầu:
- ❑ Nếu trạng thái đích, giải thuật dừng
- ❑ Ngược lại, thiết lập trạng thái bắt đầu như trạng thái hiện tại.

2. Lặp đến khi: gặp trạng thái đích hoặc không còn toán tử (Operators) nào có thể áp dụng được vào trạng thái hiện tại.
- ❑ Chọn toán tử phù hợp để sinh ra các trạng thái mới từ trạng thái hiện tại.
- ❑ Với mỗi trạng thái mới được sinh ra mà tốt hơn trạng thái hiện tại, chọn ngẫu nhiên một trong các trạng thái này:
  - Nếu là trạng thái đích → giải thuật dừng.
  - Ngược lại, thiết lập trạng thái này là trạng thái hiện tại và trở lại bước lặp số 2.

Stochastic hill climbing (Leo đồi ngẫu nhiên)

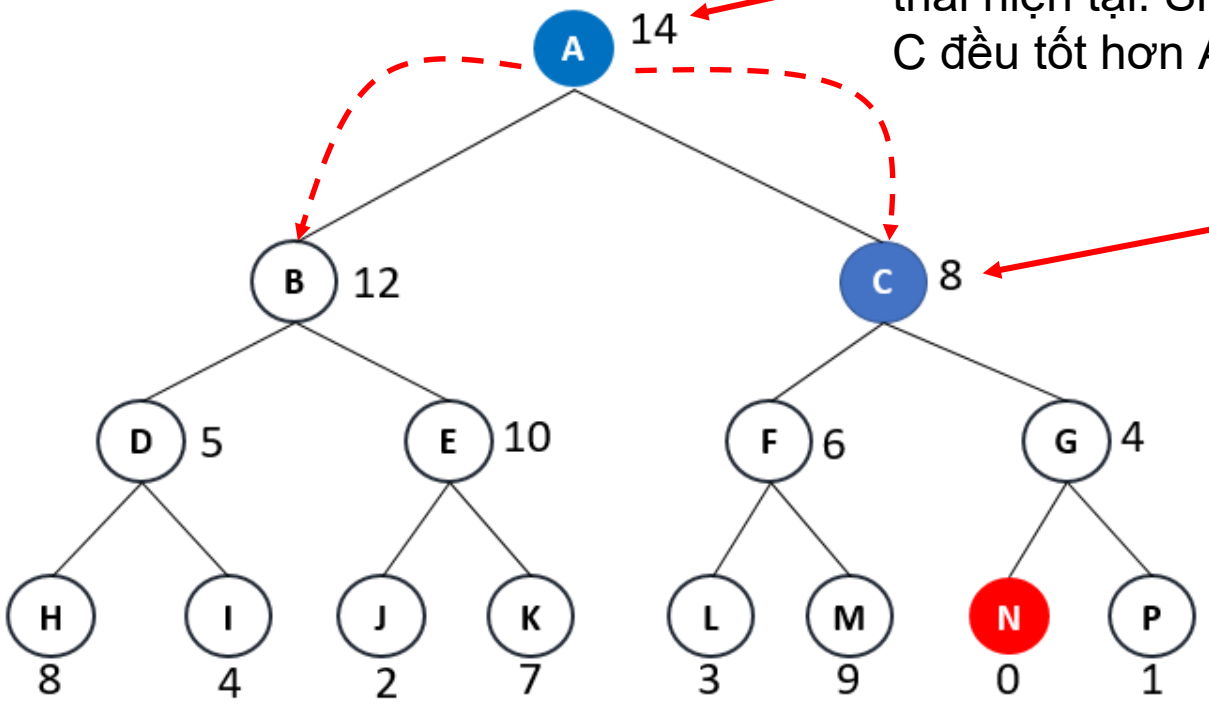
Cho đồ thị như bên dưới, nếu áp dụng giải thuật tìm kiếm leo đồi ngẫu nhiên và giá trị f(n) được tính sẵn để bên cạnh các đỉnh. Cho biết thứ tự duyệt các nút, biết trạng thái bắt đầu là A và kết thúc là N.



Stochastic hill climbing (Leo đồi ngẫu nhiên)

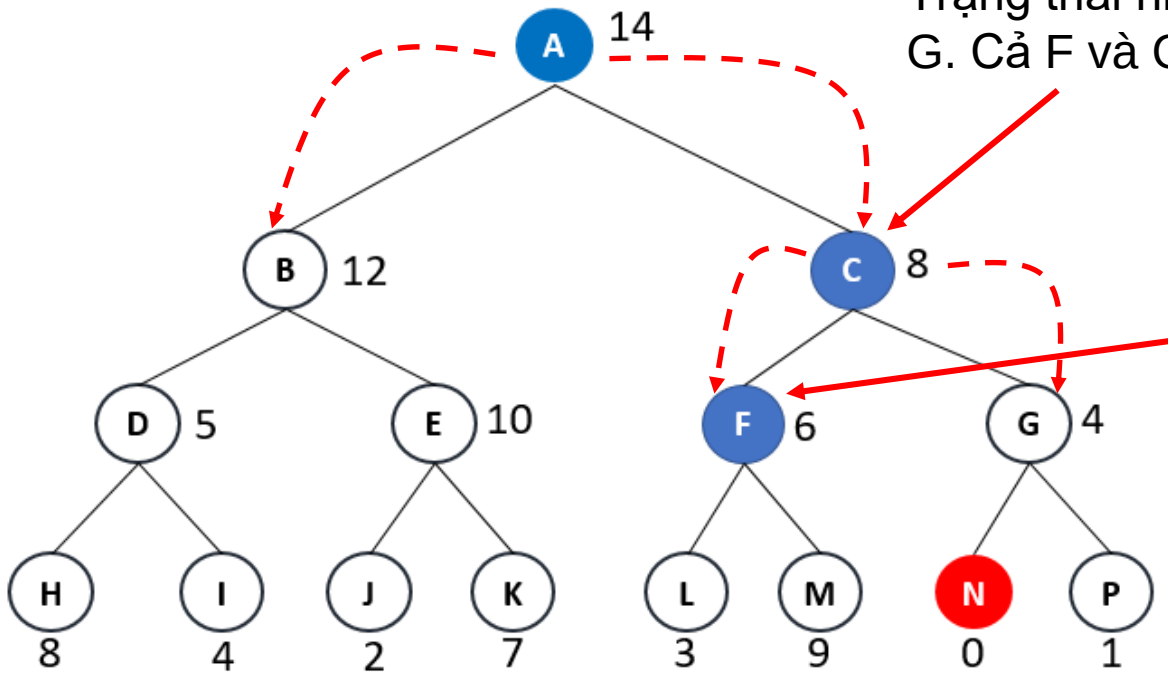
Xét trạng thái A → A chưa là Goal → Trạng thái hiện tại. Sinh ra trạng thái B, C. Cả B và C đều tốt hơn A.

Chọn ngẫu nhiên C đi tiếp. C không là Goal → Thiết lập trạng thái hiện tại là C.



Thứ tự duyệt: A, C

Stochastic hill climbing (Leo đồi ngẫu nhiên)



Trạng thái hiện tại là C. Sinh ra trạng thái F, G. Cả F và G đều tốt hơn C.

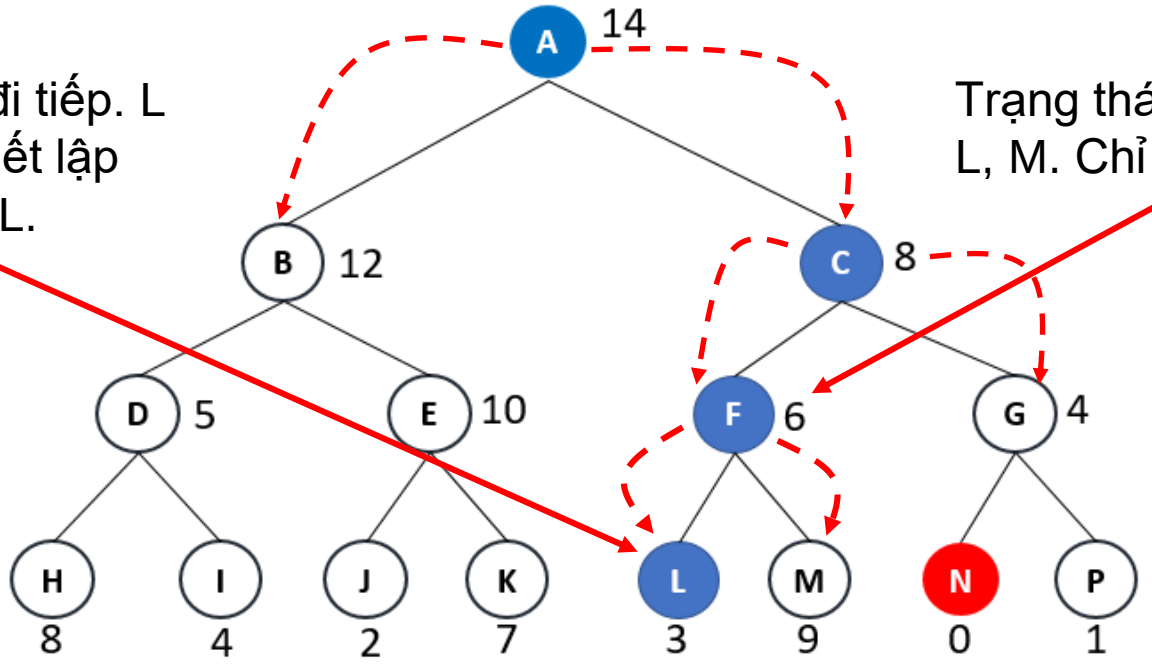
Chọn ngẫu nhiên F đi tiếp. F không là Goal → Thiết lập trạng thái hiện tại là F.

Thứ tự duyệt: A, C, F

Stochastic hill climbing (Leo đồi ngẫu nhiên)

Chọn ngẫu nhiên L đi tiếp. L không là Goal → Thiết lập trạng thái hiện tại là L.

Trạng thái hiện tại là F. Sinh ra trạng thái L, M. Chỉ có L tốt hơn F.



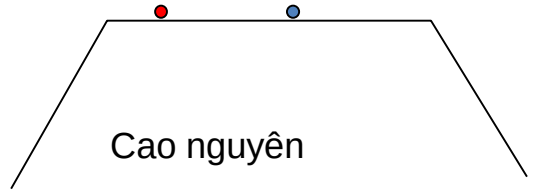
Thứ tự duyệt: A, C, F, L, (giải thuật dừng)

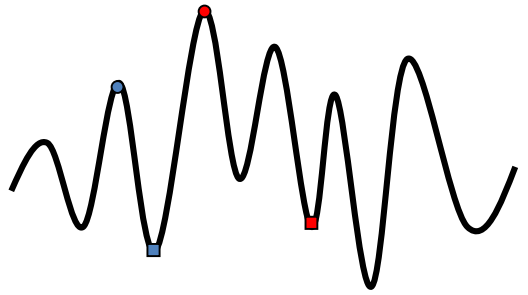
Ưu điểm của giải thuật leo đồi

- Hill Climbing là một thuật toán đơn giản và trực quan, dễ hiểu và dễ thực hiện.
- Nó có thể được sử dụng trong nhiều vấn đề tối ưu hóa khác nhau, bao gồm cả những vấn đề có không gian tìm kiếm lớn và các ràng buộc phức tạp.
- Hill Climbing thường rất hiệu quả trong việc tìm kiếm tối ưu cục bộ, khiến nó trở thành một lựa chọn tốt cho các vấn đề cần giải pháp tốt một cách nhanh chóng.

Nhược điểm của giải thuật leo đồi

- Không thể phục hồi lại từ những thất bại trong chiến lược của nó.
- Hiệu quả hoạt động chỉ có thể được cải thiện trong một phạm vi giới hạn nào đó.
- Lời giải tìm được không tối ưu hoặc không tìm được lời giải mặc dù có tồn tại lời giải do:
  - Có khuynh hướng sa lầy ở cực đại cục bộ.
  - Cao nguyên.
  - Chỏm chỉ với một phép toán, không cho ra trạng thái « tốt hơn », nhưng với một vài phép toán có thể chuyển đến trạng thái « tốt hơn »





Cực trị địa phương

Nhược điểm của giải thuật leo đồi

Một vài giải pháp xử lý các vấn đề này:

- **Quay lui** « một vài bước » trước đó và thử đi theo một hướng khác. Để thực thi chiến lược này, duy trì một danh sách các bước đã trải qua. Giải pháp này đặc biệt phù hợp để xử lý tình huống « Local Optima »
- **Tạo ra một « bước nhảy đột phá » theo một hướng**: để chuyển sang một « vùng » mới trong không gian tìm kiếm. Phù hợp để xử lý tình huống « Plateau »
- **Áp dụng nhiều hơn một toán tử** để nhận được một trạng thái sau đó mới kiểm thử. Phù hợp để xử lý tình huống « Ridge »


The End