Mục lục

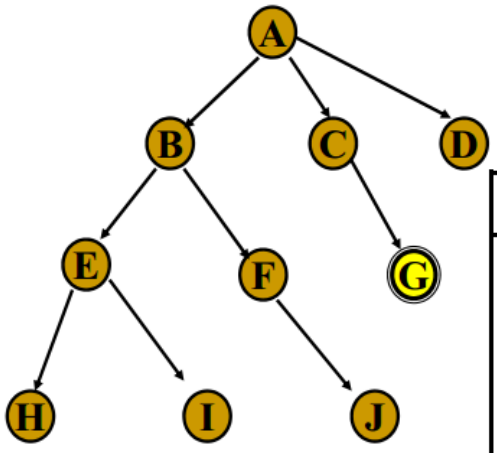
1. Tìm kiếm rộng (Breadth-first search)

FIFO (First in First out) - QUEUE – Vào trước Ra trước

- **Mã giả thuật toán BFS**

```
Open :=[START]
Close:= ∅
previous(START)=NULL
WHILE (Open không chứa GOAL và khác rỗng) do
  Lấy TT s nằm bên trái nhất trong Open
  Đặt s vào Close;
  for mỗi TT s' trong succs(s)
    IF s chưa gán nhãn (chưa xét) then
        Đặt previous(s') := s;
        Đưa s' vào bên PHẢI nhất của Open
IF Open rỗng return FAILURE
Else Xây dựng lời giải.
  Định nghĩa Sk = GOAL; Đường đi được tính dựa
  trên hàm previous với Sk-1 = previous(Sk). Cho
  đến khi Sk-1 là START.
```

- **Thực hiện thuật toán BFS**



Lần lặp	X	Open	Close
0		[A]	[ ]
1	A	[B C D]	[A]
2	B	[C D E F]	[A B]
3	C	[D E F G]	[A B C]
4	D	[E F G]	[A B C D]
5	E	[F G H I]	[A B C D E]
6	F	[G H I J]	[A B C D E F]
7	**G**	[H I J]	[A B C D E F]

2. Tìm kiếm sâu (Depth-first search)

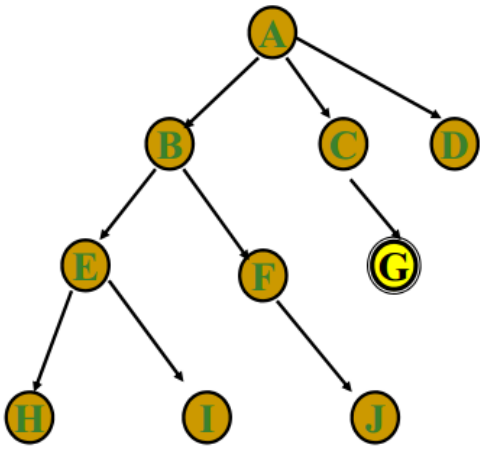
Depth first search có khả năng **lặp vô tận** do các trạng thái con sinh ra liên tục. Độ sâu tăng vô tận. Khắc phục bằng cách giới hạn độ sâu của giải thuật

LIFO(Last in first out) – STACK – Vào sau ra trước

- **Mã giả thuật toán DFS**

```
Open :=[START]
Close:= ∅
previous(START)=NULL
WHILE (Open không chứa GOAL và khác rỗng) do
  Lấy TT s nằm bên trái nhất trong Open
  Đặt s vào Close;
  for mỗi TT s' trong succs(s)
    IF s chưa gán nhãn (chưa xét) then
       Đặt previous(s') := s;
       Đưa s' vào bên TRÁI nhất của Open
IF Open rỗng return FAILURE
Else Xây dựng lời giải.
  Định nghĩa S_k = GOAL; Đường đi được tính dựa
  trên hàm previous với S_k-1 = previous(S_k). Cho
  đến khi S_k-1 là START.
```

- **Thực hiện thuật toán DFS**



Lần lặp	X	Open	Close
0		[A]	[ ]
1	A	[B C D]	[A]
2	B	[E F C D]	[A B]
3	E	[H I F C D]	[A B E]
4	H	[I F C D]	[A B E H]
5	I	[F C D]	[A B E H I]
6	F	[J C D]	[A B E H I F]
7	J	[C D]	[A B E H I F J]
8	C	[G D]	[A B E H I F J C]
9	**G**		

3. Tìm kiếm sâu dần (Iterative deepening search) IDS

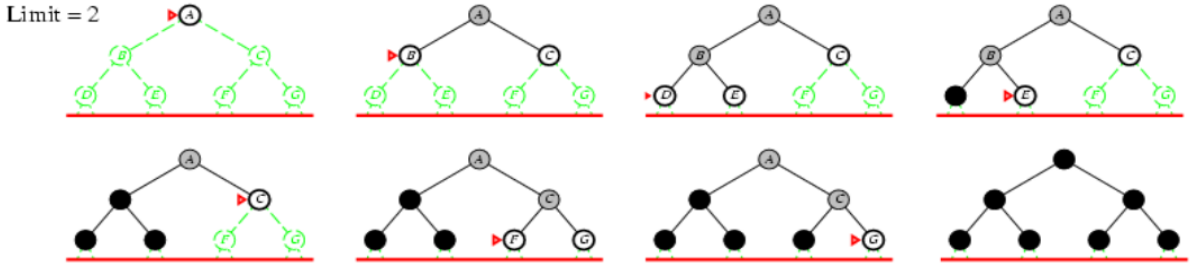
Như DFS nhưng **giới hạn(limit) độ sâu**

Vd:

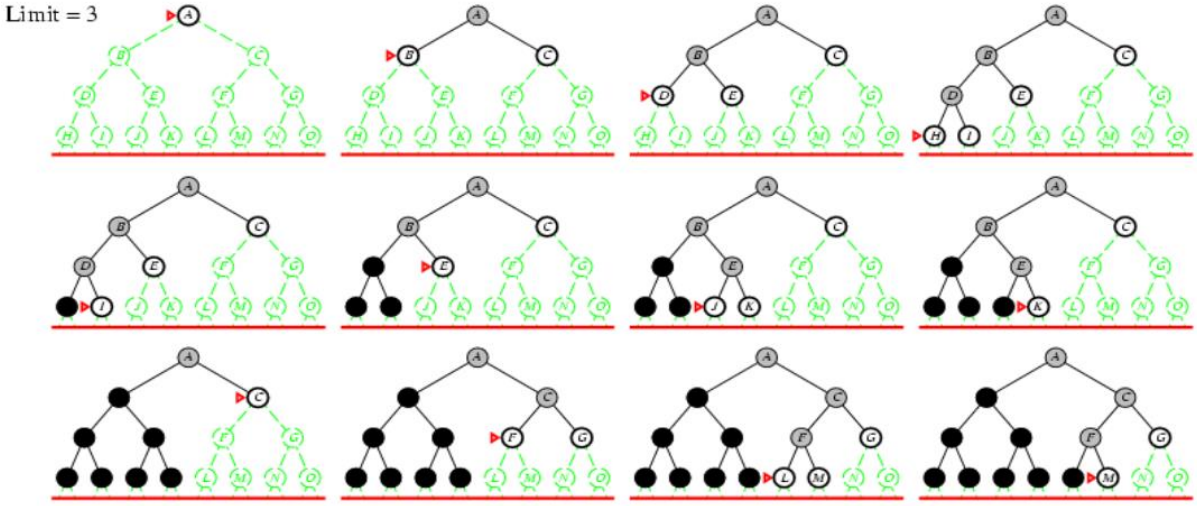
1. **Độ sâu limit = 0**

2. **Độ sâu limit = 1**

3. **Độ sâu limit = 2**



4. **Độ sâu limit = 3**



4. Tìm kiếm leo đồi

Tìm kiếm leo đồi là một trường hợp đặc biệt của tìm kiếm theo chiều sâu nhưng không thể quay lui. Trong tìm kiếm leo đồi, việc lựa chọn trạng thái tiếp theo được quyết định dựa trên một hàm Heuristic.

Hàm heuristic là gì ?

Heuristic là hàm ước lượng về khả năng dẫn đến lời giải. Ký hiệu là h

Đọc thêm:

- Chọn một trạng thái tốt hơn trạng thái hiện hành để mở rộng. Nếu không, thuật toán phải dừng
- Nếu chỉ chọn một trạng thái tốt hơn: leo đồi đơn giản; trạng thái tốt nhất: leo đồi dốc đứng.
- Sử dụng hàm H để biết trạng thái nào tốt hơn
- Khác với tìm kiếm sâu, leo đồi không lưu tất cả các con mà chỉ lưu đúng một trạng thái được chọn nếu có.

4.1 Leo đồi đơn giản (học lý thuyết)

Leo đồi đơn giản chỉ chọn đi theo trạng thái kế tiếp đầu tiên tốt hơn trạng thái hiện hành mà nó tìm thấy.

Trình bày các bước của thuật giải leo đồi đơn giản.

Bước 1: Nếu trạng thái bắt đầu (T_0) là trạng thái đích: thoát và báo là đã tìm được lời giải.

Ngược lại, đặt trạng thái hiện hành (T_i) là trạng thái khởi đầu (T_0)

Bước 2: Lặp lại cho đến khi đạt đến trạng thái kết thúc hoặc cho đến khi không tồn tại một trạng thái tiếp theo **hợp lệ** (T_k) của trạng thái hiện hành:

a. Đặt T_k là một trạng thái tiếp theo hợp lệ của trạng thái hiện hành T_i.
b. Đánh giá trạng thái T_k mới :
   b.1. Nếu là trạng thái đích thì trả về trị này và thoát.
   b.2. Nếu không phải là trạng đích nhưng tốt hơn trạng thái hiện hành thì cập nhật nó thành trạng thái hiện hành.
   b.3. Nếu nó không tốt hơn trạng thái hiện hành thì tiếp tục vòng lặp.

4.2. Leo đồi dốc đứng (học làm bài tập)

Giống như leo đồi đơn giản, chỉ khác ở điểm là leo đồi dốc đứng sẽ duyệt tất cả các hướng đi có thể và chọn đi theo trạng thái tốt nhất trong số các trạng thái kế tiếp có thể có. (trong khi đó leo đồi đơn giản chỉ chọn đi theo trạng thái kế tiếp đầu tiên tốt hơn trạng thái hiện hành mà nó tìm thấy).

Trình bày các bước của thuật giải leo đồi dốc đứng.

Bước 1: Nếu trạng thái bắt đầu cũng là trạng thái đích thì thoát và báo là đã tìm được lời giải.

Ngược lại, đặt trạng thái hiện hành (Ti) là trạng thái khởi đầu (T0)

Bước 2: Lặp lại cho đến khi đạt đến trạng thái kết thúc hoặc cho đến khi (Ti) không tồn tại một trạng thái kế tiếp (Tk) nào **tốt hơn** trạng thái hiện tại (Ti)

a) Đặt S bằng tập tất cả trạng thái kế tiếp có thể có của Ti và tốt hơn Ti.

b) Xác định T_{kmax} là trạng thái tốt nhất trong tập S

Đặt $T_i = T_{kmax}$

Ví dụ bài tập:

Bước 1: n:=Startnode;

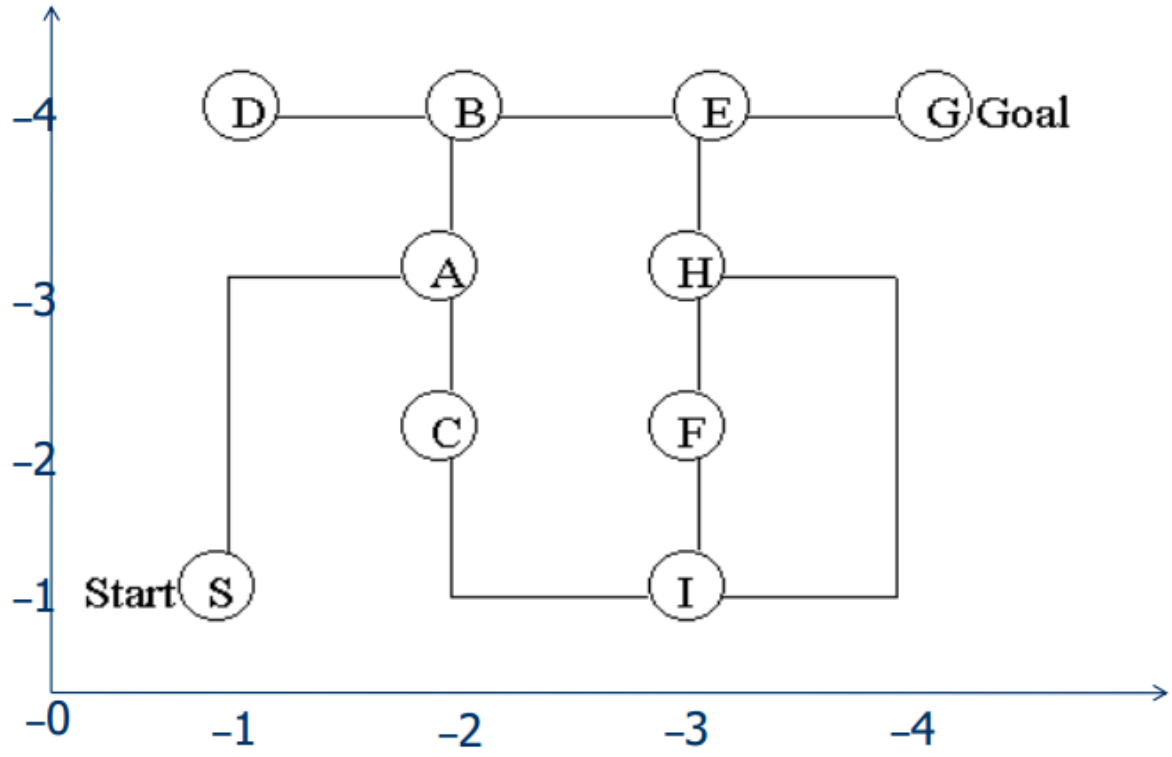
Bước 2: Nếu n là đích thì dừng (Success).

Bước 3: Triển khai n, Tính hàm h với ni là con của n. Chọn ni tương ứng với h nhỏ nhất và gọi là nextn.

Bước 4: Nếu không có ni thì thoát (Fail).

Bước 5: n:=nextn;

Bước 6: Nhảy sang bước 2



H(n) = |Tọa độ x của đích – Tọa độ x của n | + |Tọa độ y của đích – Tọa độ y của n |

n:=S

h(S)=|4-1|+|4-1|= 6 (min)

h(A)=|4-2|+|4-3|= 3 (min) < h(S)

NextS = A

n:=A

h(B)=|4-2|+|4-4|=2 (min) h(C)=|4-2|+|4-2|=4

h(B)<h(A)

NextA = B

n:=B

h(E)=|4-3|+|4-4|=1 (min) < h(B)

NextB = E

n:=E

h(G)=|4-4|+|4-4|=0 (min) h(H)=|4-3|+|4-3|=2

h(G)<h(E)

NextE = G (Đích- Dừng)

5. Thuật giải GTS (Greedy-Traveling Saleman)

5.1. GTS1

Xây dựng một lịch trình du lịch có chi phí Cost tối thiểu cho bài toán trong trường hợp phải qua n thành phố với ma trận chi phí C và **bắt đầu tại một đỉnh U** nào đó.

Thuật giải:

Bước 1: {Khởi đầu}

–Đặt Tour := {};

–Cost := 0;

–V := U; {V là đỉnh hiện tại đang làm việc}

Bước 2: {Thăm tất cả các thành phố}

–For k := 1 To n Do

–qua bước 3;

Bước 3: {Chọn cung kế tiếp}

–Đặt (V, W) là cung có chi phí nhỏ nhất tình từ V đến các đỉnh W chưa dùng:

–Tour := Tour + {(V,W)};

–Cost := Cost + Cost(V,W);

–Nhãn W được sử dụng

–Đặt V := W; {Gán để xét bước kế tiếp}

Bước 4: {Chuyến đi hoàn thành}

–Đặt Tour := Tour + {(V,U)};

–Cost := Cost + Cost(V,U);

–Dừng.

Bài tập:

-U = A

-Tour = {}

$$C = \begin{bmatrix} \infty & 1 & 2 & 7 & 5 \\ 1 & \infty & 4 & 4 & 3 \\ 2 & 4 & \infty & 1 & 2 \\ 7 & 4 & 1 & \infty & 3 \\ 5 & 3 & 2 & 3 & \infty \end{bmatrix}$$

-Cost = 0

-V = A

-W ∈ {B, C, D, E} {Các đỉnh có thể đến từ A}

-→ W = B {Vì qua B có giá thành bé nhất}

-Tour = {(A, B)}

-Cost = 1

-V = B

-W ∈ {C, D, E}

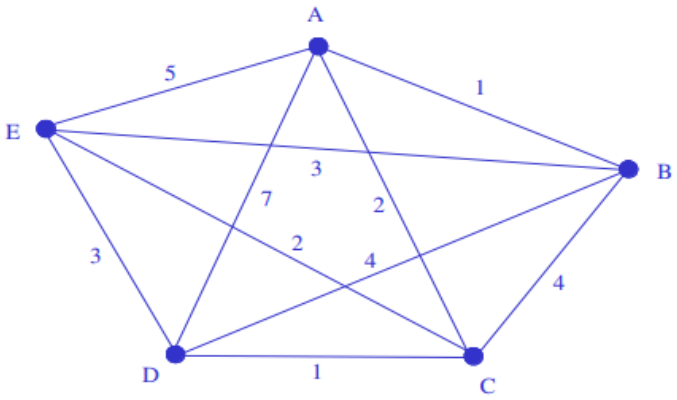
-→ W = E

-Tour = {(A, B),(B, E)}

-Cost = 1 + 3 = 4

-V = E

-W ∈ {C, D}



-→ W = C

-Tour = {(A, B), (B, E), (E, C)}

-Cost = 4 + 2 = 6

-V = C

-W ∈ {D}

-→ W = D

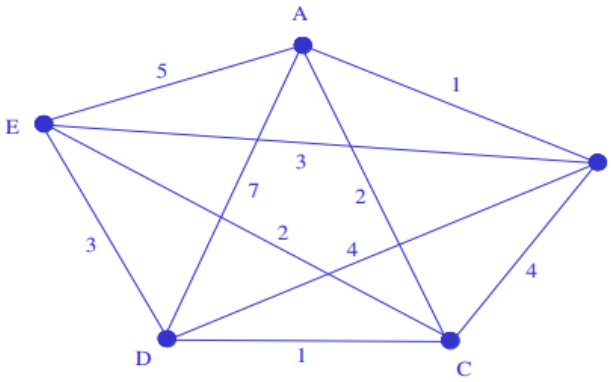
-Tour = {(A, B), (B, E), (E, C), (C, D)}

-Cost = 6 + 1 = 7

-V = D

-Tour = {(A, B), (B, E), (E, C), (C, D), (D, A)}

-Cost = 7 + 7 = 14



-Kết quả: Tour du lịch A → B → E → C → D → A với giá thành Cost = 14.

-Nhận xét: Tuy nhiên kết quả nhỏ nhất sẽ là A → B → D → C → E → A với Cost=13. Sở dĩ không tối ưu do "háu ăn": cứ hướng nào có chi phí thấp thì đi, bất chấp về sau.

5.2. GTS2

Tạo ra lịch trình từ **p thành phố xuất phát** riêng biệt. Tìm chu trình của người bán hàng qua n thành phố (1<p<n) và p chu trình được tạo ra và chỉ **chu trình tốt nhất trong p chu trình** được giữ lại mà thôi (thuật giải này đòi hỏi phải nhập n, p và C).

Thuật giải:

Bước 1: {Khởi đầu}

–k := 0; {Đếm số thành phố đi qua}

–Best := {}; {Ghi nhớ chu trình tốt nhất tìm thấy có chi phí là Cost}

–Cost := ∞;

Bước 2: {Bắt đầu chu trình mới}

– Chuyển qua bước 3 khi k<p, ngược lại dừng.

Bước 3: {Tạo chu trình mới}

–k := k + 1;

–Call (GTS1(Vk)) : Trả về một chu trình T(k) ứng với chi phí C(k).

Bước 4: {Cập nhật chu trình tốt nhất}

–Nếu C(k)< Cost thì Best := T(k);

–Cost := C(k);

Ví dụ: (Giải lại ví dụ trên)

p=3

K=0 Best={} Cost=∞

K=0<p V0=**A** Call(GTS1(A)) ->T(0)={(A,B),(B,E),(E,C),(C,D),(D,A)}, C(0)=14

C(0)=14<Cost ->Best=T(0) **Cost=14**

K=1<p V1=**B** Call(GTS1(B)) ->T(1)={(B,A),(A,C),(C,D),(D,E),(E,B)}, C(1)=10

C(1)=10<Cost ->Best=T(1) **Cost=10**

K=2<p V2=**C** Call(GTS1(C)) ->T(2)={(C,D),(D,E),(E,B),(B,A),(A,C)}, C(2)=10

C(2)=10>=Cost

K=3>=p Dừng.

Vậy nếu nhập p=3 chu trình thì chúng ta bắt đầu từ thành phố B và có Cost nhỏ nhất. Nếu ta chọn p khác thì ta sẽ có kết quả khác có thể tốt hơn kết quả trên.

Ví dụ p = 4 các bạn giải thử xem.

6. Thuật toán tô màu tối ưu trên đồ thị

Thuật toán: Lặp lại các bước sau cho đến khi nào tô màu hết các đỉnh

Bước 1: Chọn đỉnh có bậc lớn nhất tô màu i.

Bước 2: Hạ bậc:

Đỉnh đã tô màu: bậc = 0

Những đỉnh có liên hệ: bậc := bậc – 1

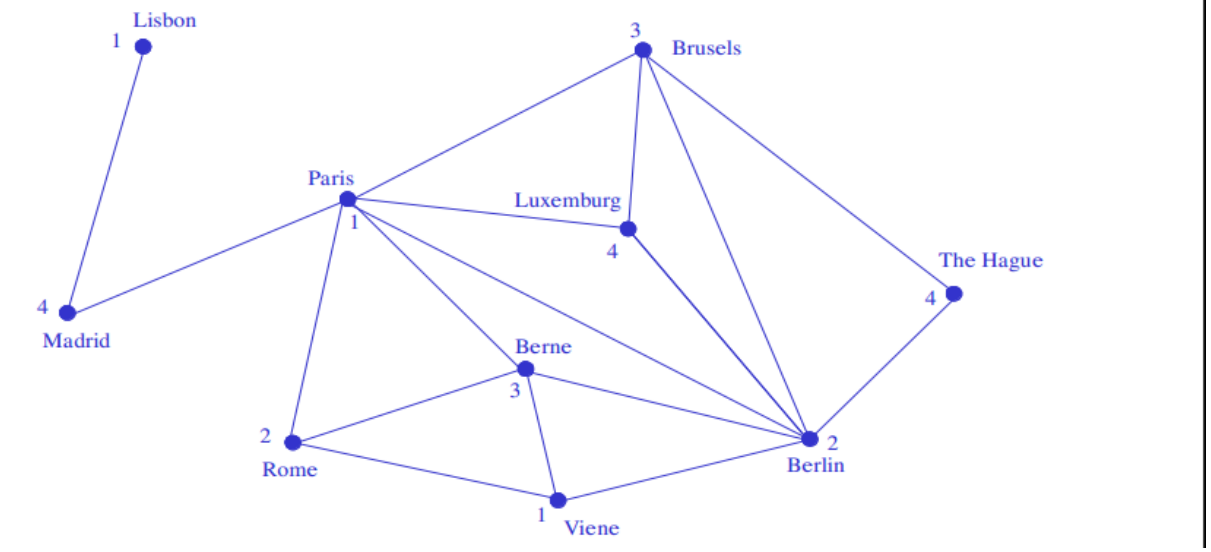
Bước 3: Đánh dấu các đỉnh liên hệ (bậc vừa trừ đi 1) cấm tô màu i

---

Ví dụ 1:

Yêu cầu: tô toàn bộ bản đồ mà chỉ sử dụng 4 màu sao cho không có bất kỳ 2 nước láng giềng nào có cùng chung một màu

Đỉnh	Lisbon L	Madrid M	Paris P	Berne Be	Rome R	Viene V	Berlin Ber	Luxemburg Lx	Brusen Bru	Hague H
Bậc	1	2	6	4	3	3	6	3	4	2



Giải:

Màu cấm tô										
				2		3		2	2	3
		1		1	1	2	1	1	1	2
Đỉnh	L	M	P	Be	R	V	Ber	Lx	Bru	H
Màu tô	1	2	1	3	2	1	2	4	3	1
Bậc	1	2	6*	4	3	3	6	3	4	2
Hạ bậc lần 1	1	1	0	3	2	3	5*	2	3	2
Hạ bậc lần 2	1	1		2	2*	2	0	1	2	1
Hạ bậc lần 3	1	1		1	0	1		1	2*	1
Hạ bậc lần 4	1*	1		1		1		0	0	0
Hạ bậc lần 5		0		1*		1		0		0
Hạ bậc lần 6		0		0		0		0		0

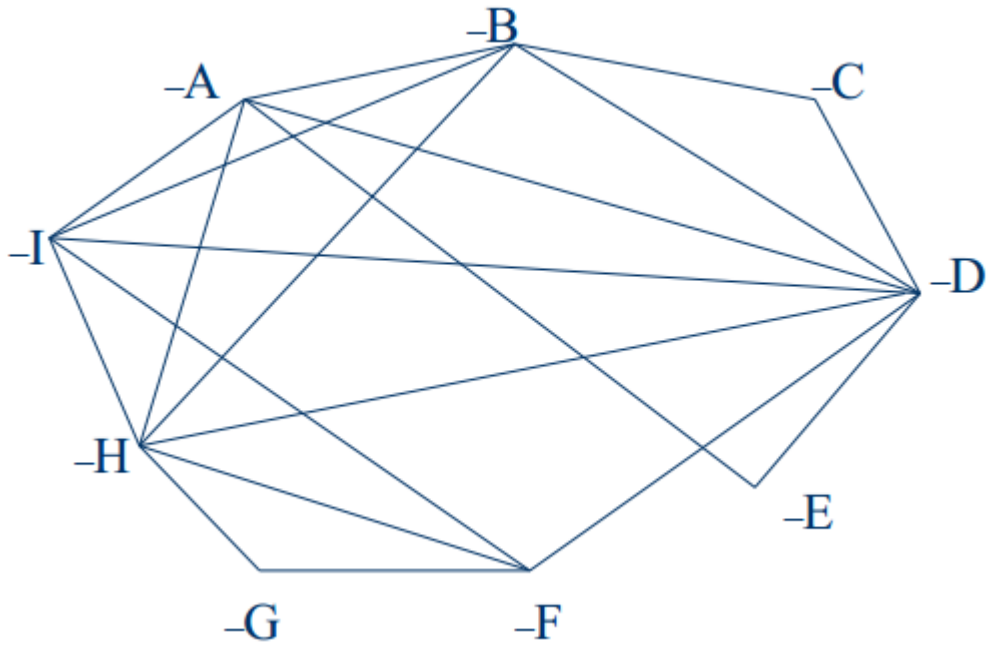
Ví dụ 2: Phân công, lịch công tác, lịch thi đấu:

Có một cuộc hội thảo khoa học với 9 chủ đề khác nhau, mỗi chủ đề diễn ra trong một buổi.

Các chủ đề sau không được đồng thời: AE, BC, CD, ED, ABD, AHI, BHI, DFI, DHI, FGH.

Xây dựng lịch sao cho số buổi diễn ra là ít nhất. Gợi ý: số màu = số buổi.

=>Giải:



									4
		3							3
	2	2	4		3	2	3		2
Màu cấm tô	1	1	1		1	1	2	1	1
Đỉnh	A	B	C	D	E	F	G	H	I
Màu tô	3	4	2	1	2	3	1	2	5
Bậc	5	5	2	7*	2	4	2	6	5
Hạ bậc	4	4	1	0	1	3	2	5*	4
	3*	3	1		1	2	1	0	3
	0	2*	1		0	2	1		2
		0	0			2*	1		1
						0	0		0

Kết luận:

Buổi 1: G, D; **Buổi 2**: C, E, H; **Buổi 3**: A, F; **Buổi 4**: B; **Buổi 5**: I

7. Thuật giải AT (Algorithm for Tree)

Trình bày các bước của thuật giải AT

Bước 1:

+ Mọi đỉnh n, mọi giá trị g(n) đều là ẩn.

+ Mở đỉnh đầu tiên và gọi đó là đỉnh S. Đặt g(S) = 0.

Bước 2: Chọn đỉnh mở với giá thành g tương ứng là nhỏ nhất và gọi đó là đỉnh N.

+ Nếu N là mục tiêu: đường đi từ đỉnh ban đầu đến N là đường đi ngắn nhất và bằng g(N). Dừng (Success).

+ Nếu không tồn tại một đỉnh mở nào nữa: cây biểu diễn vấn đề không có đường đi tới mục tiêu. Dừng (Fail).

+ Nếu tồn tại nhiều hơn 1 đỉnh N (nghĩa là có 2 đỉnh N trở lên) mà có cùng giá thành g(N) nhỏ nhất. Kiểm tra xem trong số đó có đỉnh nào là đích hay không.

Nếu có: đường đi từ đỉnh ban đầu đến đỉnh N là ngắn nhất và bằng g(N), dừng (Success).

Nếu không có: Chọn ngẫu nhiên một trong các đỉnh đó và gọi là đỉnh N.

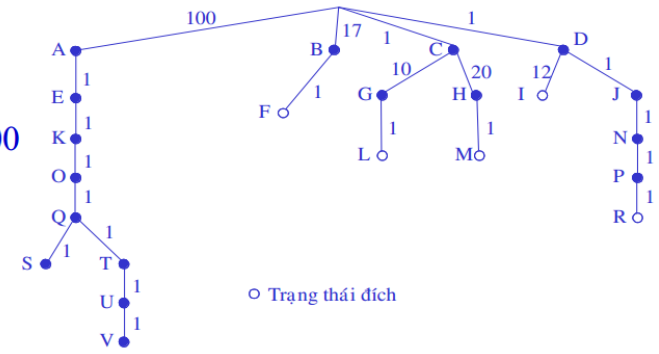
Bước 3: Đóng đỉnh N và mở các đỉnh sau N (là những đỉnh có cung hướng từ N tới). Tại mọi đỉnh S sau N tính :

g(S) = g(N) + cost(N->S)

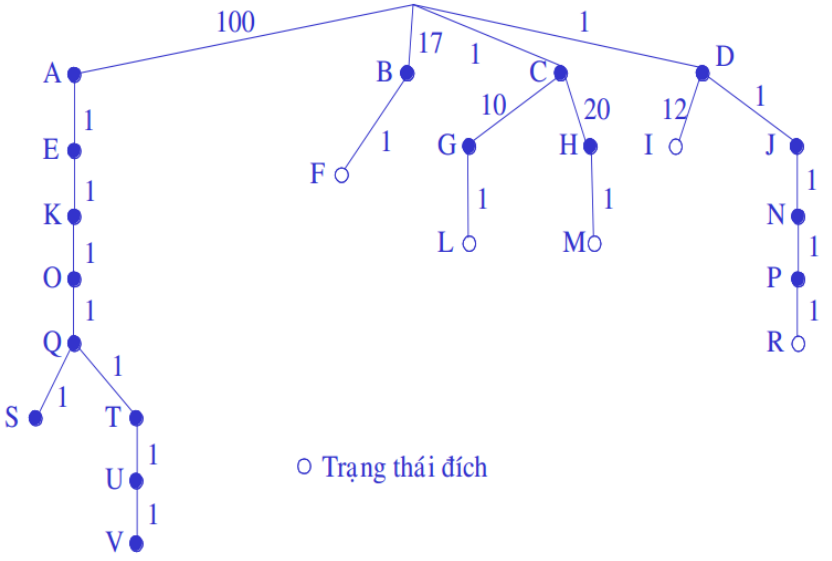
Bước 4: Quay lại bước 2

Bài tập ví dụ:

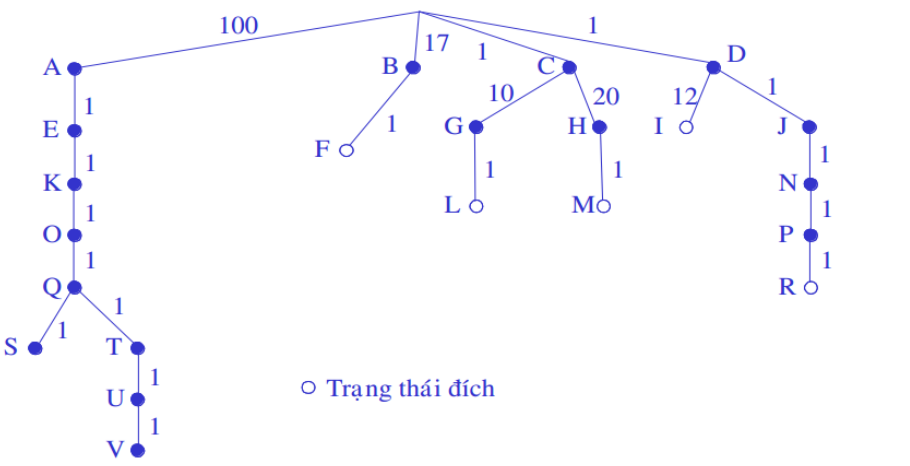
Mọi đỉnh n, g(n) chưa biết.
B1: Mở S, đặt g(S) = 0.
B2: Đóng S; mở A, B, C, D
g(A) = g(S) + gt(S→A) = 0 + 100 = 100
g(B) = 0 + 17 = 17
g(C) = g(D) = 0 + 1 = 1 (min)
Chọn ngẫu nhiên giữa C, D: chọn C
B3: Đóng C, mở G, H:
g(A) = 100
g(B) = 17
g(D) = 1 (min)
g(G) = 11
g(H) = 21



B4: Đóng D, mở I, J:
g(A) = 100
g(B) = 17
g(I) = 13
g(J) = 2 (min)
g(G) = 11
g(H) = 21
B5: Đóng J, mở N:
g(A) = 100
g(B) = 17
g(I) = 13
g(G) = 11
g(H) = 21
g(N) = 3 (min)



B6: Đóng N, mở P:
g(A) = 100
g(B) = 17
g(I) = 13
g(G) = 11
g(H) = 21
g(P) = 4 (min)
B7: Đóng P, mở R:
g(A) = 100
g(B) = 17
g(I) = 13
g(G) = 11
g(H) = 21
g(R) = 5 (min)
R là đích. Vậy đường đi là:



$S \xrightarrow{1} D \xrightarrow{1} J \xrightarrow{1} N \xrightarrow{1} P \xrightarrow{1} R$

Nhận xét: Thuật toán này chỉ sử dụng 3 thông tin: đỉnh, cung và giá thành của cung.

8. Thuật giải AKT – Tìm kiếm với tri thức bổ sung (Algorithm for Knowledgeable Tree Search)

Bước 1:

Mọi đỉnh, cũng như các hàm g, h, f chưa biết.

Mở đỉnh đầu tiên S, gán g(S) = 0

Sử dụng tri thức bổ sung để ước tính hàm h(S)

Tính f(S) = g(S) + h(S)

Bước 2: Chọn đỉnh mở có f là nhỏ nhất và gọi là đỉnh N

Nếu N là đích: đường đi từ đỉnh ban đầu đến đỉnh N là ngắn nhất và và bằng g(N). Dừng (Success).

Nếu không tồn tại đỉnh mở nào: cây biểu diễn vấn đề không tồn tại đường đi tới mục tiêu. Dừng (Fail).

Nếu có 2 đỉnh mở trở lên có cùng giá trị f nhỏ nhất: Chúng ta phải kiểm tra xem những đỉnh đó có đỉnh nào là đích hay không.

+ Nếu có: đường đi từ đỉnh ban đầu đến đỉnh N là ngắn nhất và bằng g(N). Dừng (Success).

+ Nếu không có: chọn ngẫu nhiên một trong các đỉnh đó và gọi đỉnh đó là N.

Bước 3:

Đóng đỉnh N, mở mọi đỉnh sau N. Với mỗi đỉnh S sau N, tính:

g(S) = g(N) + cost(S->N)

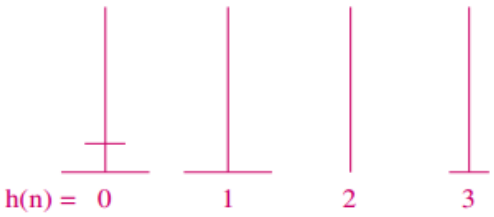
Sử dụng tri thức bổ sung để tính h(S) và f(S): f(S) = g(S) + h(S)

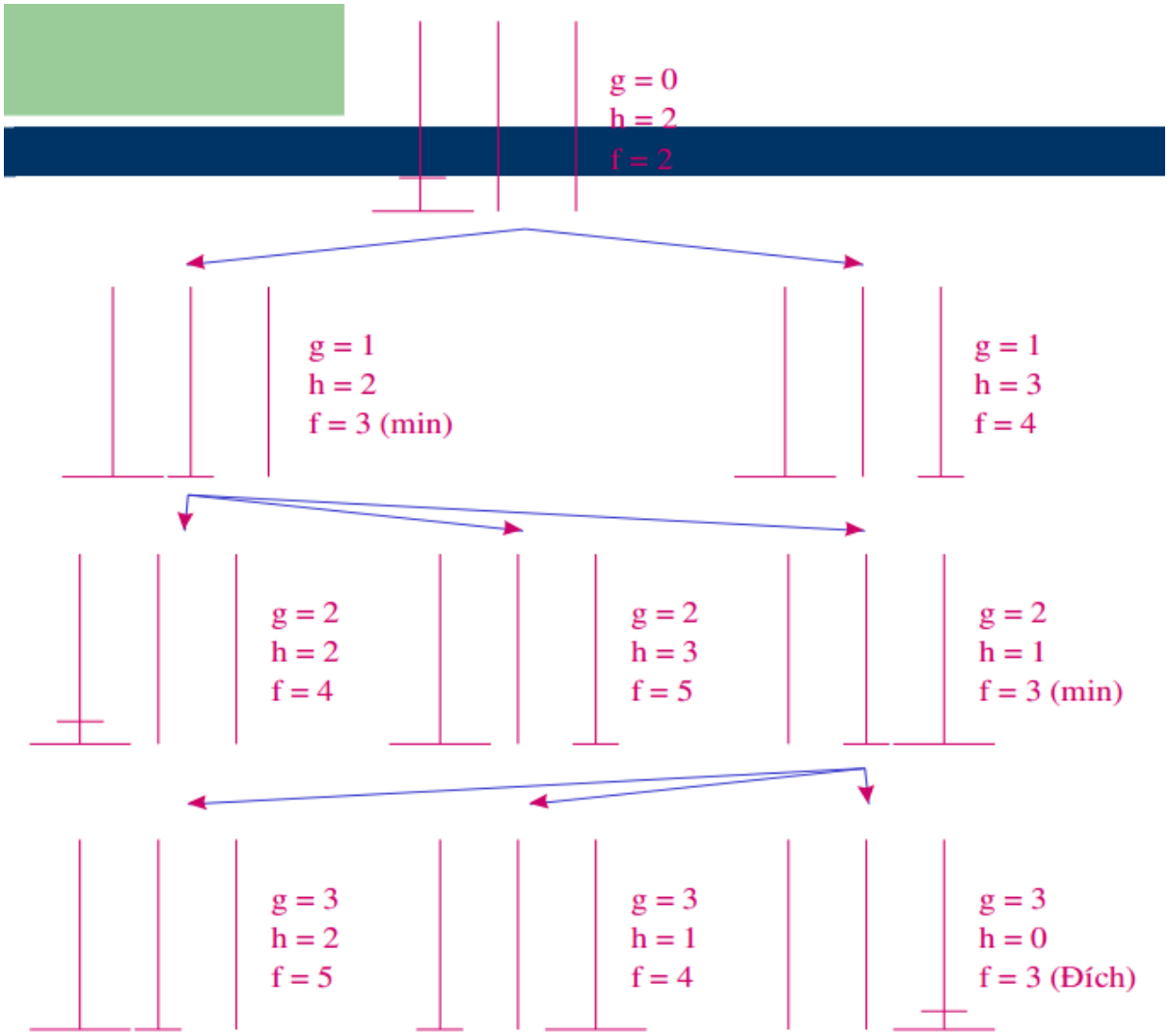
Bước 4: Quay lại bước 2.

Ví dụ:

Bài toán Tháp Hà Nội với n = 2

-Các trường hợp của bài toán là với trạng thái cột thứ ba:

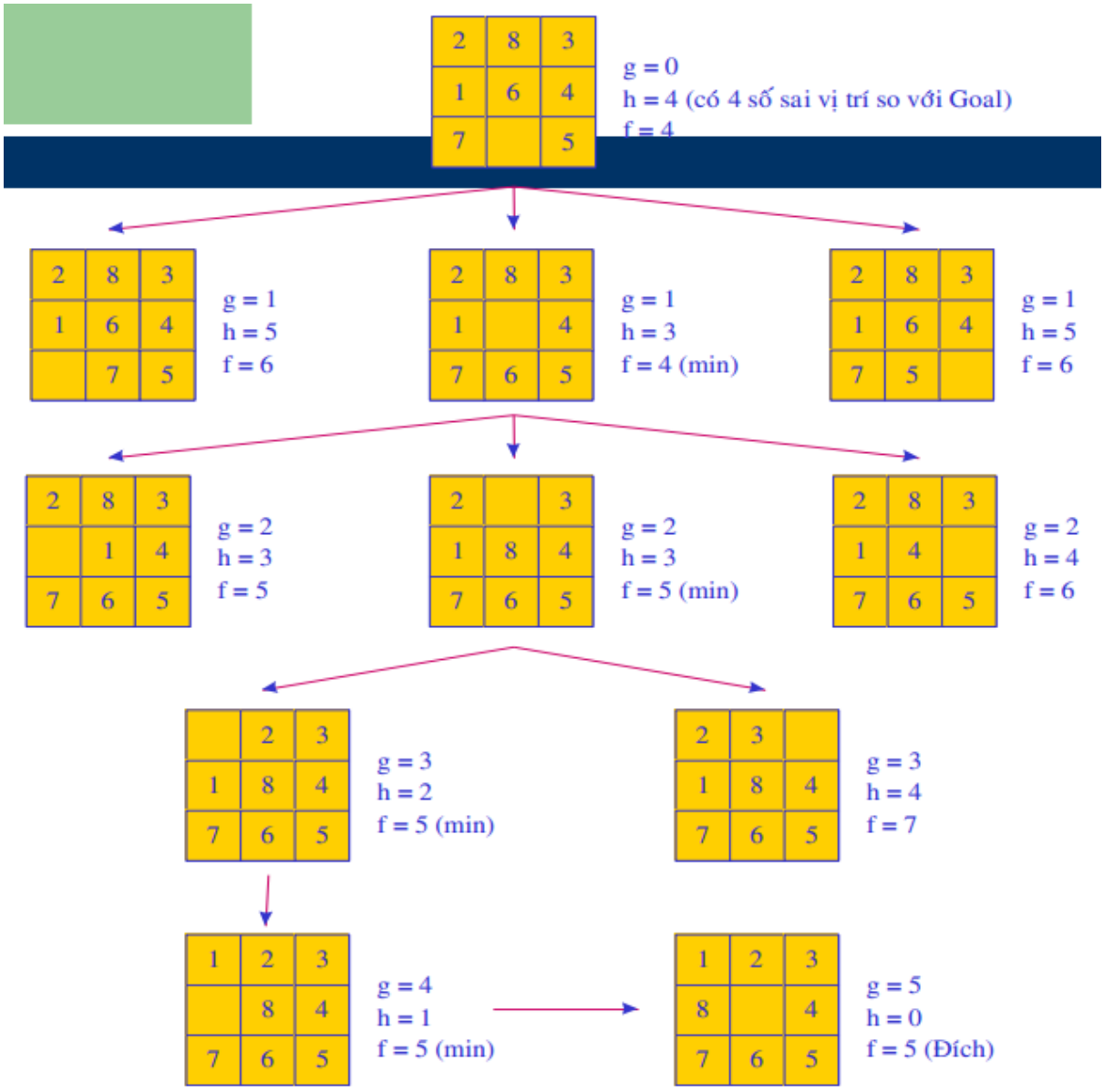




Bài toán taci

☞**Cách 1**:

$$H = \sum_{i=1}^{t} \delta(a_i, b_i) \text{ với } \delta(a_i, b_i) = \begin{cases} 0 \text{ nếu } a_i = b_i \\ 1 \text{ nếu } a_i \neq b_i \end{cases}$$



☞**Cách 2:** $H = \sum_{i=1}^{t} \eta(a_i, b_i)$

– với $\eta(a_i,b_i)$ là số lần ít nhất phải đẩy ô a_i = a theo chiều dọc hay ngang về đúng vị trí b_i = b.

– Giả sử:

-2	-8	-3
-1	-6	-4
-7		-5

-Số	-1	-2	-3	-4	-5	-6	-7	-8	-Cộng
-Vị trí	-1	-1	-0	-0	-0	-1	-0	-2	-5

9. Thuật giải A* - tìm kiếm đường đi trên đồ thị tổng quát

Trình bày các bước của thuật giải A*

Bước 1: Mọi đỉnh và

–Mọi đỉnh, cũng như các hàng g, h, f chưa biết.

–Mở đỉnh đầu tiên S, gán g(S) = 0

–Ước lượng hàm h(S)

–Gán f(S) = h(S)+ g(S)

Bước 2: Chọn đỉnh mở có f(S) là nhỏ nhất và gọi là đỉnh N

- Nếu N là đích: đường đi từ đỉnh ban đầu đến đỉnh N là ngắn nhất và và bằng g(N). Dừng (Success).
- Nếu không tồn tại đỉnh mở nào: cây biểu diễn vấn đề không tồn tại đường đi tới mục tiêu. Dừng (Fail).
- Nếu có 2 đỉnh mở trở lên có cùng giá trị f(S) nhỏ nhất: ta phải kiểm tra xem những đỉnh đó có đỉnh nào là đích hay không.

+ Nếu có: đường đi từ đỉnh ban đầu đến đỉnh N là ngắn nhất và bằng g(N). Dừng (Success).

+ Nếu không có: chọn ngẫu nhiên một trong các đỉnh đó và gọi đỉnh đó là N.

Bước 3:

–Đóng đỉnh N, và đối với mỗi đỉnh S sau N, chúng ta tính:

–g'(S) = g(N) + cost(S->N)

–Nếu đỉnh S đã mở và g(S) <= g'(S) thì bỏ qua S

–Ngược lại mở S và đặt g(S) = g'(S), tính h(S) và f(S): f(S) = g(S) + h(S)

Bước 4: Quay lại bước 2.

Bài toán tháp Hà Nội

Gọi n là tổng số đĩa cần chuyển.

m là số đĩa đã nằm đúng vị trí ở cột thứ 3.

k là số đĩa nằm sai vị trí ở cột thứ 3.

Có thể thấy bạn cần chuyển các đĩa nằm sai vị trí ra khỏi cột 3 (k đĩa), sau đó chuyển các đĩa chưa đúng vị trí vào đúng vị trí của nó (n-m-k đĩa), cuối cùng chuyển k

đĩa sai vị trí vào lại.

Như vậy bạn sẽ có công thức là: k + (n-m-k) + k = n-m+k.

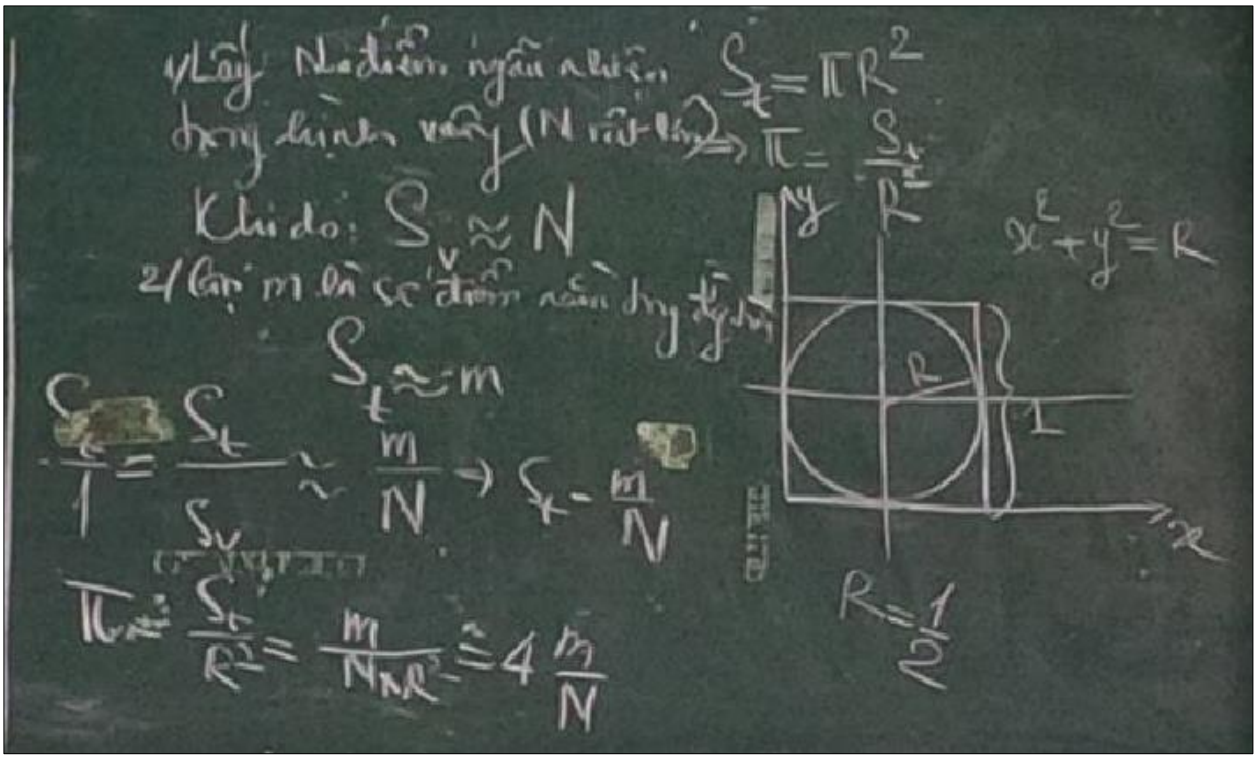
Nhận xét

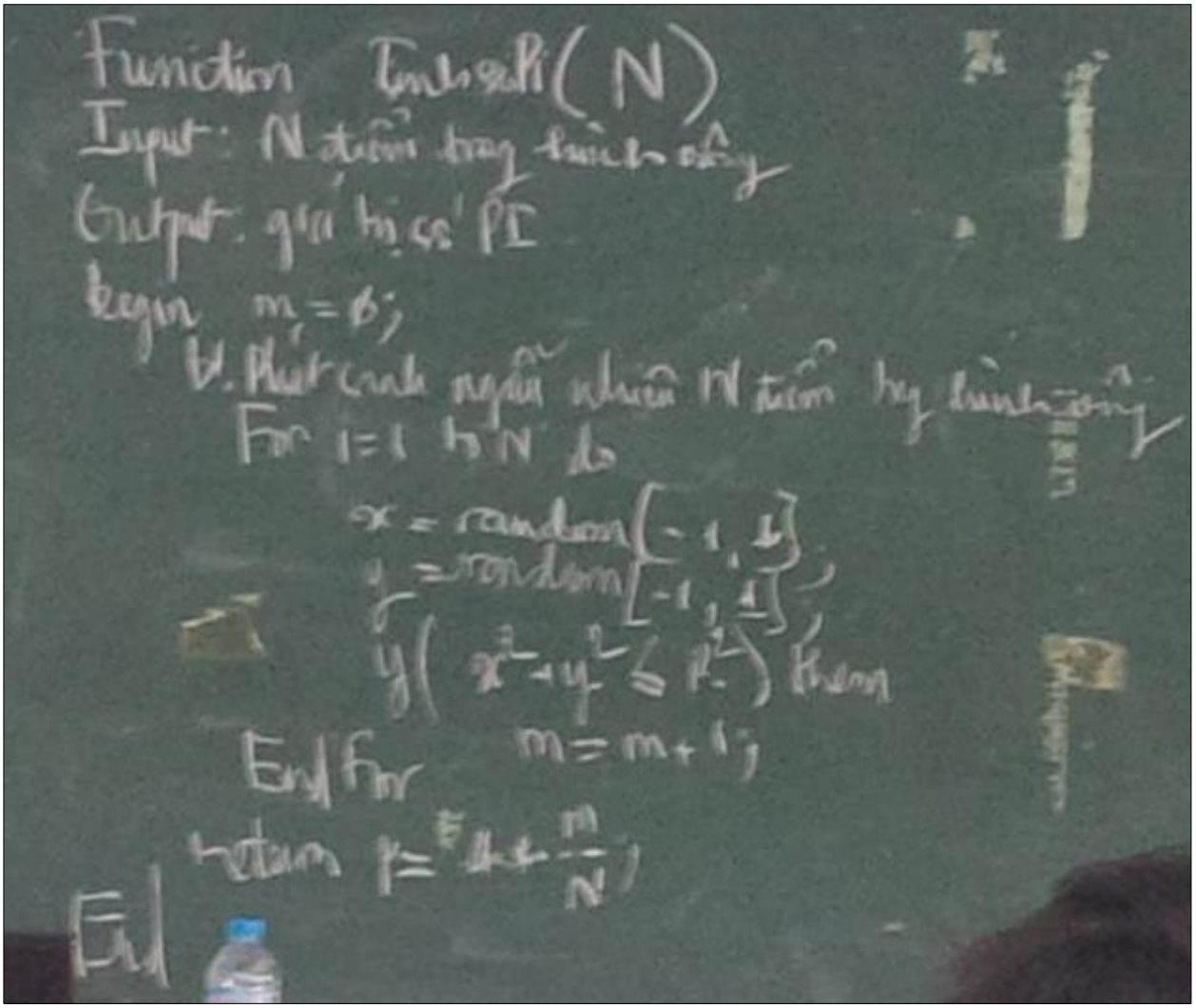
A^T	A^{KT}	A^*
đỉnh	đỉnh	đỉnh
Cung	Cung	Cung
Giá thành cung	Giá thành cung	Giá thành cung
	Tri thức bổ sung	Tri thức bổ sung
Thao tác trên cây	Thao tác trên cây	Thao tác trên đồ thị

- **Mối quan hệ giữa A^T, A^{KT}, A^*:**
- **$f(S) = (1 - \alpha)\, g(S) + \alpha\, h(S)$ với $0 \le \alpha \le 1$**
- **Nếu $\alpha = 0$ → A^T (không có tri thức bổ sung)**
- **Nếu $\alpha = 1$ → A^{KT} (Phụ thuộc vào tri thức bổ sung)**
- **Nếu $\alpha = ½$ → A^***

Bổ xung

Tính số PI


1/ Lấy N điểm ngẫu nhiên
$S_t = \pi R^2$
$\pi = \frac{S_t}{R^2}$
Khi đó: $S_t \approx N$
2/ Gọi m là số điểm nằm trong
$S_t \approx m$
$\frac{S_t}{S_V} \approx \frac{m}{N} \rightarrow S_t = \frac{m}{N}$
$\pi = \frac{S_t}{R^2} = \frac{m}{N R^2} = 4\frac{m}{N}$
$x^2 + y^2 = R$
R
1
$R = \frac{1}{2}$


Function TinhSoPi(N)
Input: N điểm trong hình vuông
Output: giá trị số PI
begin m = 0;
For i = 1 to N do
x = random(-1, 1);
y = random(-1, 1);
(x^2 + y^2 <= R^2) then
m = m + 1;
End For
return pi = 4 * m / N;
End