CHƯƠNG 6: ĐỒ THỊ (GRAPH)

DATA STRUCTURES AND ALGORITHMS

L/O/G/O

Nội dung buổi 2

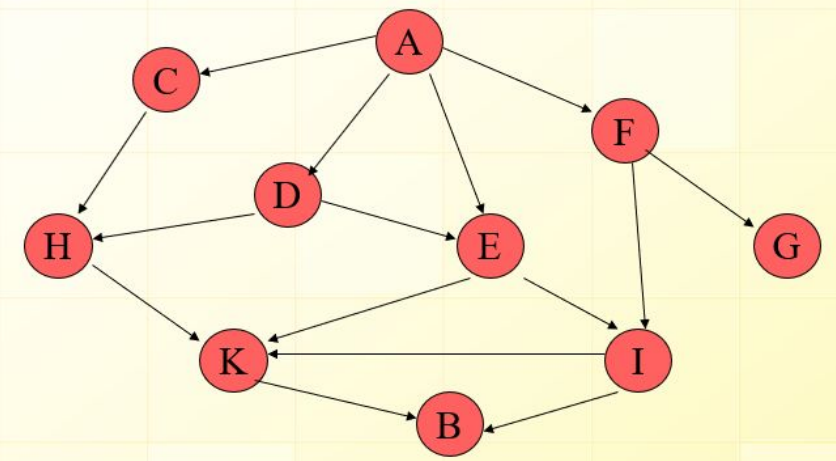
6.1 Các khái niệm trên đồ thị

6.2 Biểu diễn đồ thị trên máy tính

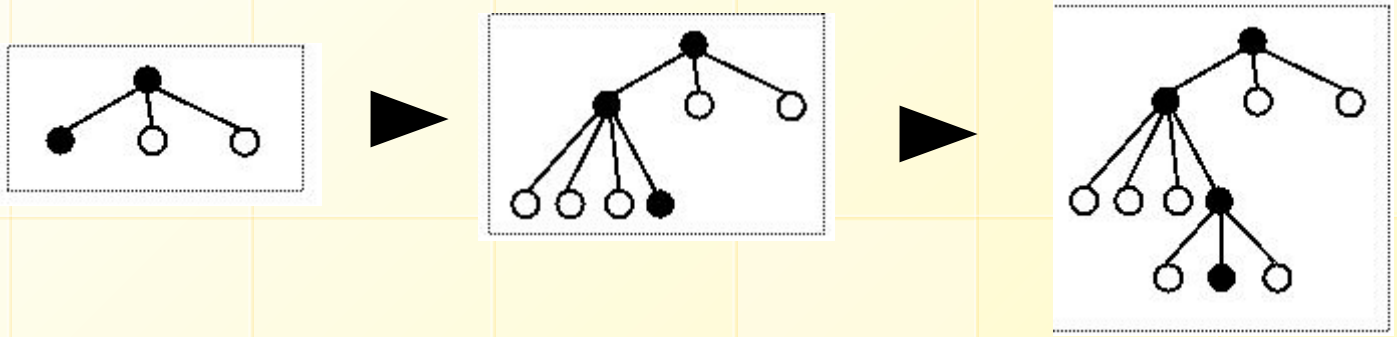
6.3 Duyệt đồ thị

Duyệt đồ thị
(Graph traversal)

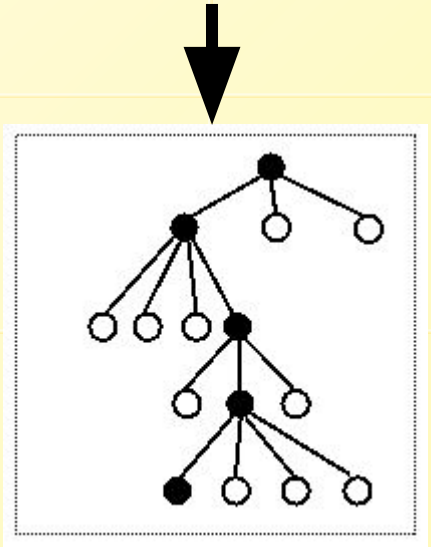
- Là quá trình đi qua các đỉnh của đồ thị một cách có hệ thống (truy cập vào các đỉnh theo một thứ tự nhất định)
- Hai phép duyệt phổ biến: duyệt (còn gọi là tìm kiếm) theo chiều rộng và theo chiều sâu
- Mục đích:
  - tìm kiếm trong đồ thị
  - kiểm tra tính liên thông
  - xác định thành phần liên thông
  - tìm chu trình
  - kiểm tra đồ thị lưỡng phân
  - sắp xếp đỉnh, …



Tìm kiếm theo chiều sâu
(Depth-First Search, viết tắt DFS)



- Khi đã chọn (duyệt) 1 đỉnh u thì bước tiếp theo là chọn trong số các đỉnh kề của u để đi tiếp
- Duyệt theo một hướng/nhánh nào đó sâu nhất có thể trước khi quay lại duyệt những nhánh khác



Tìm kiếm theo chiều sâu

- Ý tưởng: (Bài toán Tìm kiếm đường đi giữa 2 đỉnh cho trước)

  - Tại đỉnh hiện hành (d_{hh}), chọn một đỉnh kế tiếp trong tập các đỉnh kề của d_{hh} làm d_{hh} mới, rồi tiếp tục cho đến khi hoặc đến ngõ cụt hoặc đến đích
  - Nếu tại d_{hh} không thể đi đến đỉnh nào khác (ngõ cụt) quay lui lại bước chọn trước đó để chọn đường khác
  - Nếu đã quay lui đến đỉnh xuất phát mà vẫn thất bại kết luận không có lời giải.

Tìm kiếm theo chiều sâu

❖ Cài đặt: (Bài toán Tìm kiếm đường đi giữa 2 đỉnh cho trước)

Có nhiều cách cài đặt, ví dụ:

- Cách 1: sử dụng một ngăn xếp (stack) để lưu trữ các đỉnh chờ duyệt, đỉnh được chọn là đỉnh được sinh ra sau cùng trong số các đỉnh đang chờ

- Cách 2: dùng kỹ thuật đệ quy quay lui (backtracking)

Ký hiệu

Gọi Open : tập các đỉnh có thể xét ở bước kế tiếp, các đỉnh đã được "sinh ra" nhưng chưa được chọn/duyệt, đỉnh chờ duyệt

Close : tập các đỉnh đã xét/đi qua (cần lưu lại để mỗi đỉnh chỉ được đi qua một lần duy nhất)

s : đỉnh xuất phát

g : đỉnh đích

p : đỉnh đang xét, đỉnh hiện hành

$\Gamma(p)$: tập các đỉnh kề của p

$q \in \Gamma(p)$: một đỉnh kề của p

Tìm kiếm theo chiều sâu

❖ Giải thuật tham khảo:

Bước 1: Khởi tạo

Open := {s}

Close :={}

Bước 2: While (Open ≠ {})

2.1 Lấy p từ đầu Open (xoá p khỏi Open)

2.2 Nếu p là trạng thái kết thúc thì thoát, thông báo kết quả

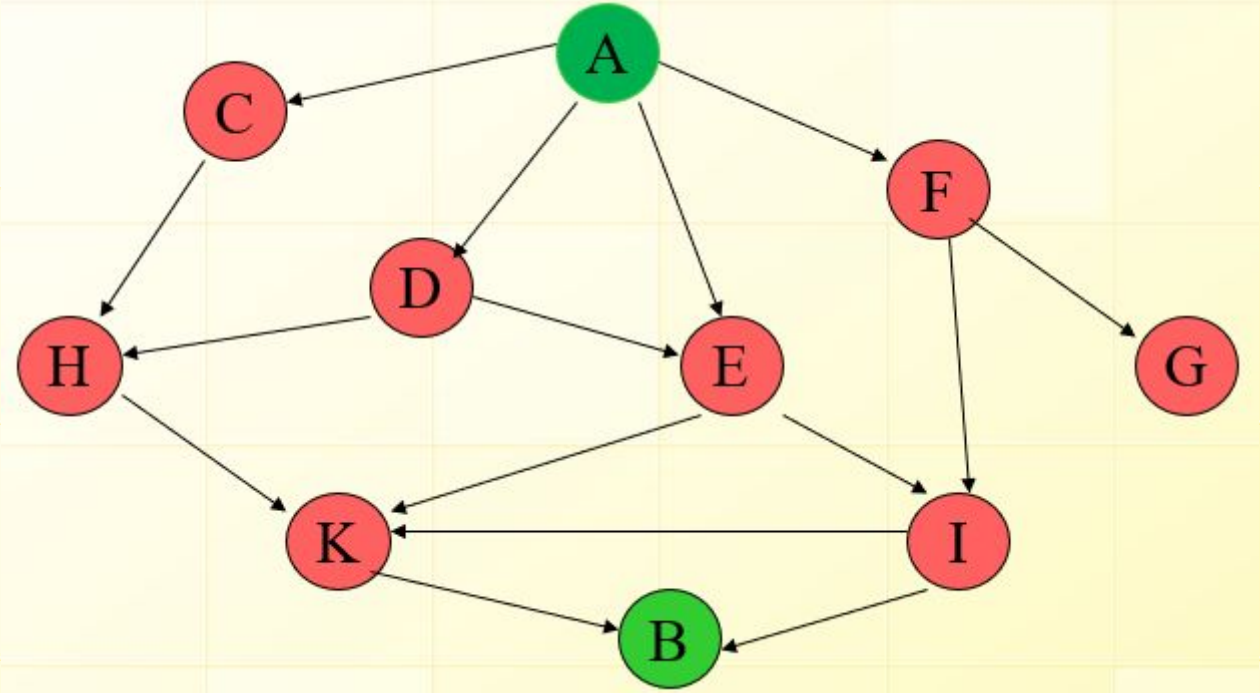
2.3 Bỏ p vào Close

2.4 Với mỗi đỉnh q kề với p,

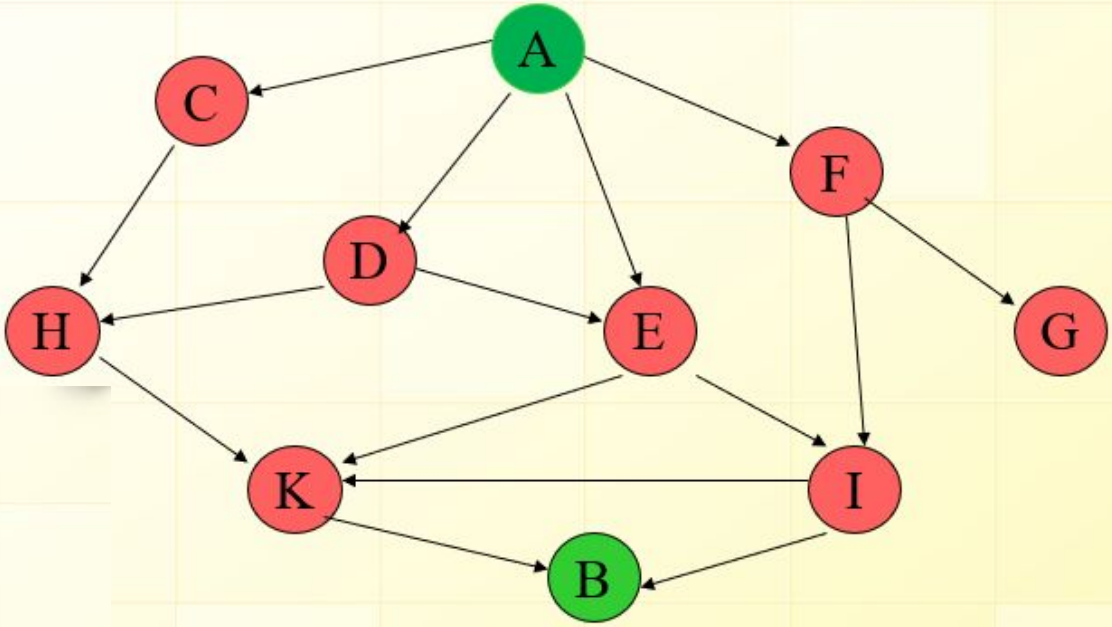
Nếu q không thuộc Close thì bỏ q vào **đầu Open**

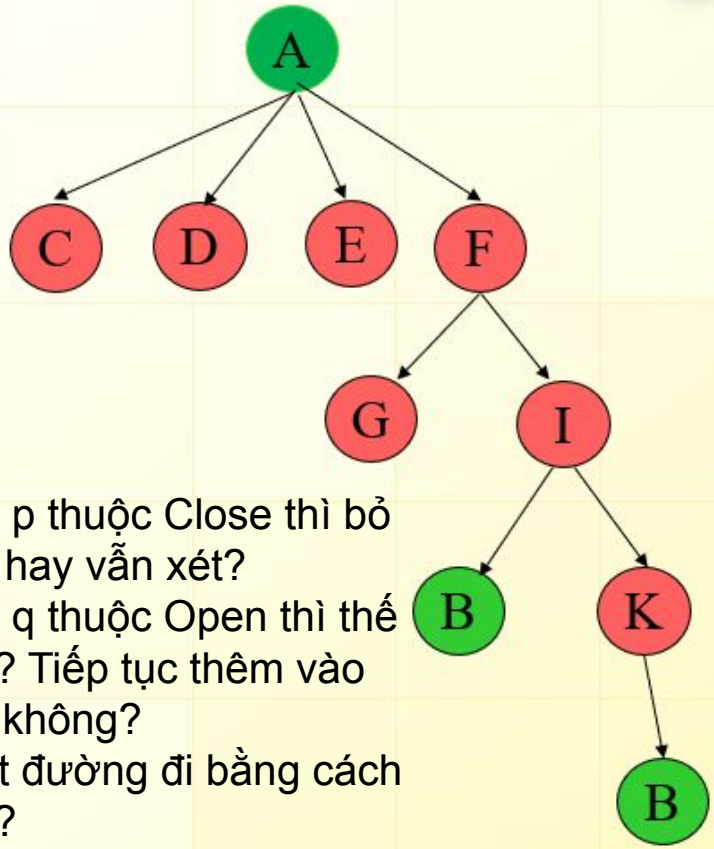
Bước 3: Không tìm thấy kết quả

Ví dụ: Tìm đường đi từ A đến B



DFS





p	Γ(p)	Open	Close
		{A}	{}
A	{C,D,E,F}	{F,E,D,C}	{A}
F	{G,I}	{I,G,E,D,C}	{A,F}
I	{B,K}	{K,B,G,E,D,C}	{A,F,I}
K	{B}	{B,G,E,D,C}	{A,F,I,K}
B	(Đến đích)		

- Nếu p thuộc Close thì bỏ qua hay vẫn xét?
- Nếu q thuộc Open thì thế nào? Tiếp tục thêm vào hay không?
- Xuất đường đi bằng cách nào?
- Độ phức tạp?

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Tìm kiếm theo chiều sâu

❖ Một số gợi ý về cài đặt:

1. Tổ chức cấu trúc dữ liệu

- vector<vector<int> > matrix; // ma trận kề của đồ thị
- vector<string> v_list; // lưu danh sách các tên đỉnh (chuỗi)
- map<string, int> v_index; // ánh xạ từ tên đỉnh sang index để có thể truy xuất thông tin trong ma trận kề
- stack<string> open; // lưu các đỉnh chờ duyệt
- vector<bool> close(v, 0); // lưu thông tin đỉnh nào đã đi qua rồi
- map<string, string> parent; // lưu thông tin cha con, parent[u]=v nghĩa là cha của u và v

Tìm kiếm theo chiều sâu

- Một số gợi ý về cài đặt:

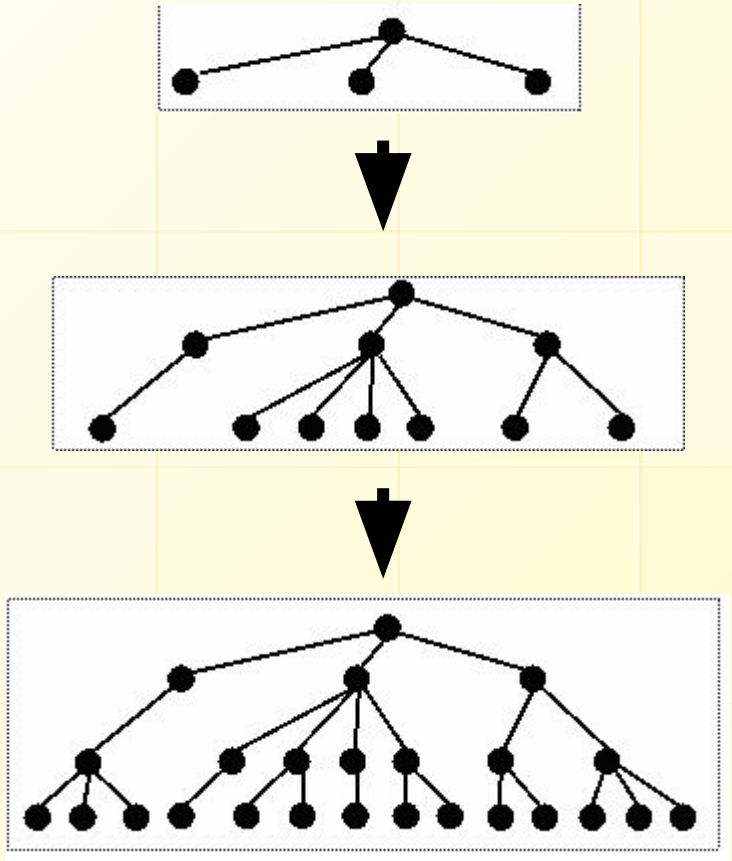
Bước 1: open.push(s);

Bước 2: while (!open.empty())

```
string p = open.top();  open.pop();
if (p == g) {found = true; break; }
close[v_index[p]] = 1;
vector<string> q = dinhke(p);
for (int i = 0; i < q.size(); i++)
        if (close[v_index[q[i]]]== 0) open.push(q[i]);
                parent[dinhke[i]] = p;
```

Bước 3: Xử lý xuất đường đi hoặc kết luận không tìm thấy

Tìm kiếm theo chiều rộng
(Breadth-First Search, viết tắt BFS)



- Duyệt qua tất cả các đỉnh ở cùng một mức (level) trước khi đi vào mức tiếp theo
- Đỉnh nào được "sinh ra" trước sẽ được chọn duyệt trước

5/28/2023

Tìm kiếm theo chiều rộng

❖ Ý tưởng: (vết dầu loang)

- Từ đỉnh xuất phát s, xây dựng tập hợp S bao gồm các đỉnh kề của s
- Ứng với mỗi đỉnh d_i trong tập S, xây dựng tập S_i bao gồm các đỉnh kề của d_i rồi lần lượt "bổ sung" các S_i vào S
- Lặp lại cho đến lúc S có chứa đỉnh đích hoặc S không thay đổi sau khi đã bổ sung tất cả S_i vào S

Tìm kiếm theo chiều rộng

- Giải thuật tham khảo:

Bước 1: Khởi tạo

Open := {s};

Close :={};

Bước 2: While (Open ≠ {})

2.1 Lấy p từ đầu Open (xoá p khỏi Open)

2.2 Nếu p là trạng thái kết thúc thì thoát, thông báo kết quả

2.3 Bỏ p vào Close

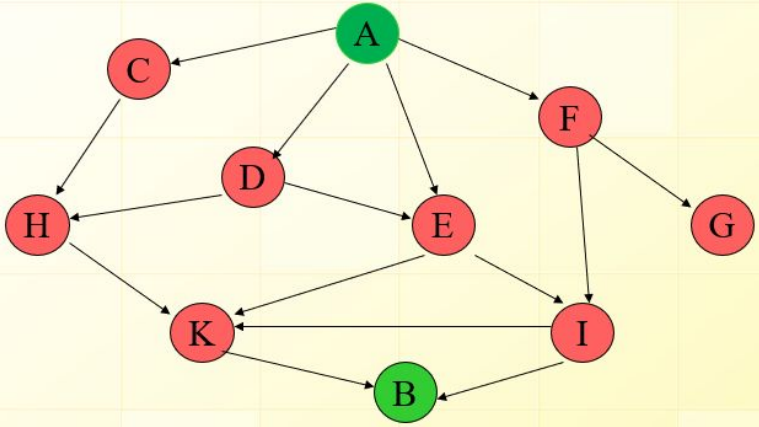
2.4 Với mỗi đỉnh q kề với p,

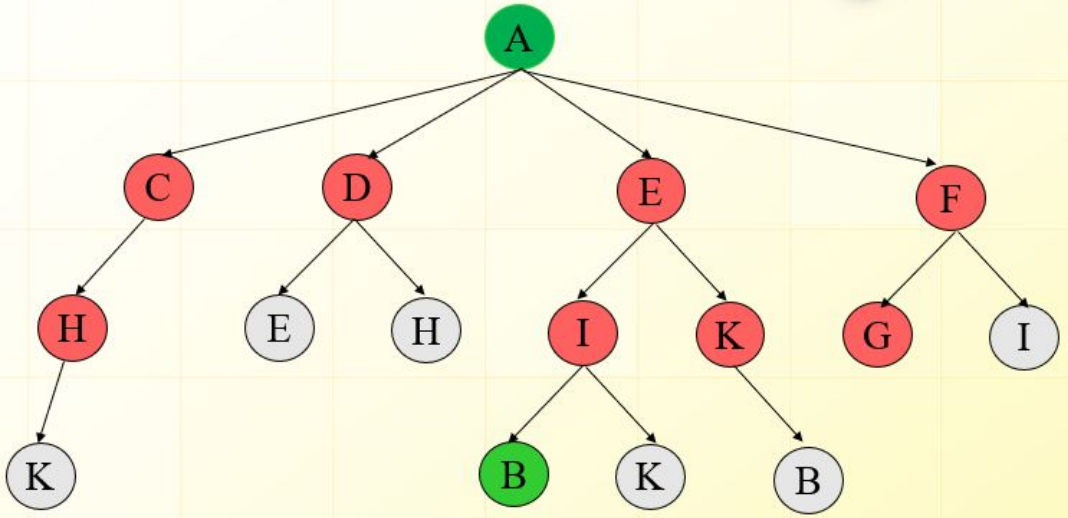
Nếu q không thuộc Close thì bỏ q vào **cuối Open**

Bước 3: Không tìm thấy kết quả

BFS

- Nếu p thuộc Close mà vẫn xét và nếu q thuộc Open mà vẫn tiếp tục thêm q vào Open thì thế nào?
- Xuất đường đi bằng cách nào?
- Độ phức tạp?



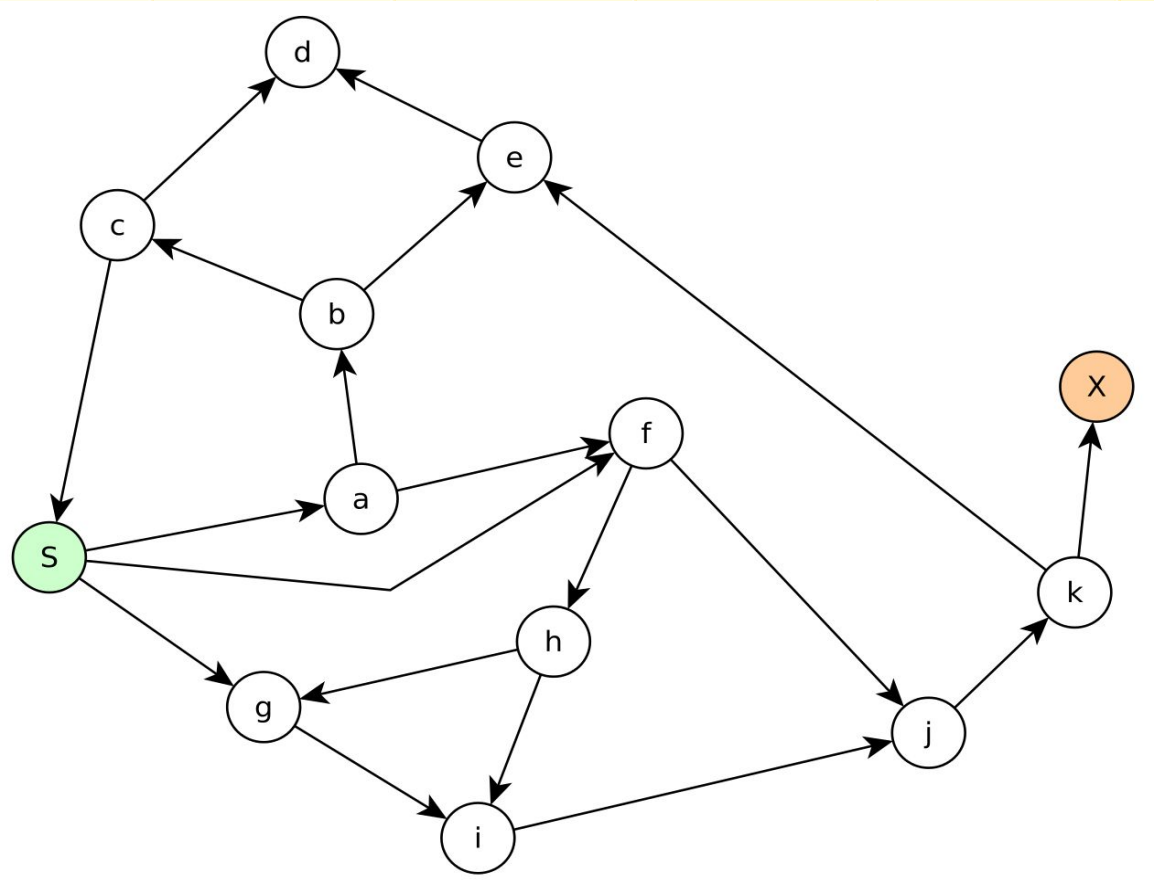


p	Γ(p)	Open	Close
		{A}	{}
A	{C,D,E,F}	{C,D,E,F}	{A}
C	{H}	{D,E,F,H}	{A,C}
D	{E,H}	{E,F,H}	{A,C,D}
E	{I,K}	{F,H,I,K}	{A,C,D,E}
F	{G,I}	{H,I,K,G}	{A,C,D,E,F}
H	{K}	{I,K,G}	{A,C,D,E,F,H}
I	{B,K}	{K,G,B}	{A,C,D,E,F,H,I}
K	{B}	{G,B}	{A,C,D,E,F,H,I,K}
G	{}	{B}	{A,C,D,E,F,H,I,K,G}
B	(Đến đích)		

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Bài tập trên lớp

Tìm đường đi từ s đến x theo chiến lược DFS/BFS

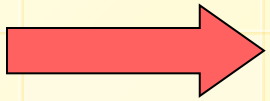


Câu hỏi thảo luận: So sánh DFS với BFS?

- Tính đầy đủ/hoàn tất (Completeness)
  - Tìm thấy lời giải (đường đi) nếu bài toán có lời giải?
- Tối ưu (Optimality)
  - Trả về đường đi có chi phi nhỏ nhất?
- Độ phức tạp (Complexity)
  - Hiệu quả về thời gian (thời gian tìm kiếm)
  - Hiệu quả về không gian (bộ nhớ sử dụng)

Câu hỏi thảo luận: Áp dụng DFS/BFS vào game 8-puzzle thì thế nào? Có chiến lược tìm kiếm nào khác tốt hơn không?

2	8	3
1	6	4
7		5



1	2	3
8		4
7	6	5

- Tìm một dãy các chuyển dịch để biến đổi từ trạng thái ban đầu thành trạng thái mục tiêu. Giả sử thứ tự thực hiện các chuyển dịch để xét mở rộng các đỉnh là đi lên, đi xuống, qua trái, qua phải. Mỗi lần di chuyển chỉ được di chuyển một ô nằm cạnh ô trống về phía ô trống

So sánh, đánh giá

Tiêu chí	BFS	DFS
Thời gian (Time complexity)	$O(b^d)$	$O(b^m)$
Không gian (Space complexity)	$O(b^d)$	O(b x m)
Hoàn tất (Complete)	Có	Không (Chỉ hoàn tất khi cây tìm kiếm là hữu hạn)
Tối ưu (Optimal)	Có (step cost = 1)	Không
Số đỉnh được sinh ra	$1 + b + b^2 + \ldots + b^d = (b^{d+1}-1)/(b-1) = O(b^d)$	$1 + b + b^2 + \ldots + b^m = O(b^m)$ □ worst case

- b: branching factor, d: depth of shallowest goal, m: maximal depth of a leaf node

Tìm kiếm theo độ sâu giới hạn (Depth Limited Search)

❖ Ý tưởng:

- Depth – First có khả năng lặp vô tận do các trạng thái con sinh ra liên tục. Mắc kẹt ở nhánh vô hạn, độ sâu tăng vô hạn
- Khắc phục bằng cách giới hạn độ sâu của giải thuật.
- Sâu bao nhiêu thì vừa?
- Giới hạn độ sâu co hẹp không gian trạng thái có thể mất nghiệm

Tìm kiếm lặp sâu dần
(Iterative Deepening Search)

❖ Ý tưởng:

- Thực hiện việc tìm kiếm với độ sâu ở mức giới hạn d nào đó.
- Nếu không tìm ra nghiệm ta tăng độ sâu lên d+1 và lại tìm kiếm theo độ sâu tới mức d+1 thử tất cả các giới hạn độ sâu có thể
- Quá trình trên được lặp lại với d lần lượt là 1, 2,...đến độ sâu max nào đó. Giải thuật sẽ quay lui khi trạng thái đang xét đạt đến độ sâu max