Search for Solutions

(ARTIFICIAL INTELLIGENCE)

(slides adapted from: Lecture Foundations of Artificial Intelligence, Malte Helmert, University of Basel

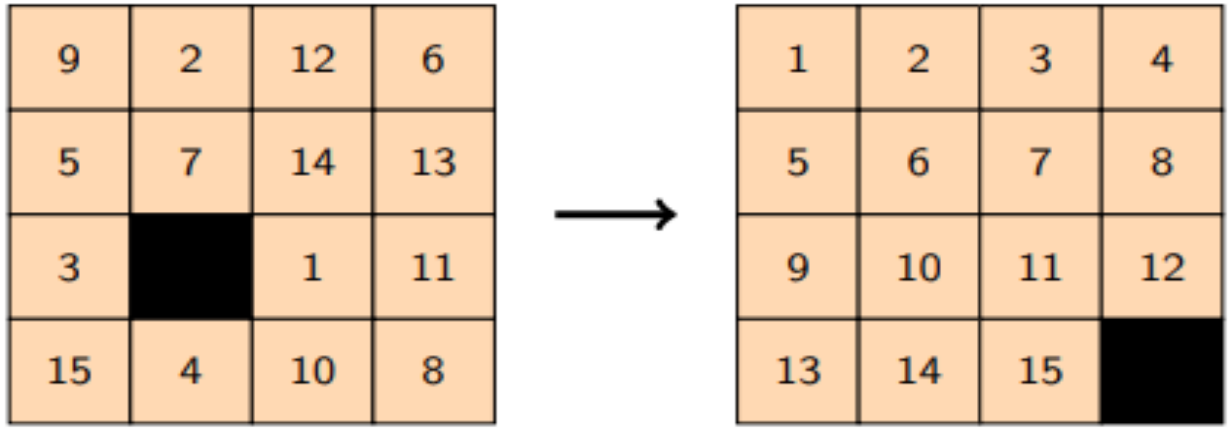
State-Space Search Problems

- **Tìm kiếm:** Tìm kiếm là một quy trình từng bước để giải quyết một vấn đề tìm kiếm trong một không gian tìm kiếm nhất định. Một vấn đề tìm kiếm có thể có ba yếu tố chính:
  - **Không gian tìm kiếm:** một tập hợp các giải pháp khả thi mà một hệ thống có thể có.
  - **Trạng thái bắt đầu:** trạng thái mà tác nhân bắt đầu **tìm kiếm** .
  - **Kiểm tra mục tiêu:** hàm quan sát trạng thái hiện tại và trả về kết quả trạng thái mục tiêu có đạt được hay không.

State-Space Search Problems

- Các bài toán tìm kiếm không gian trạng thái nằm trong số các lớp bài toán AI "đơn giản nhất" và quan trọng nhất.

- mục tiêu của tác nhân:
  - từ trạng thái ban đầu đã cho
  - áp dụng một chuỗi hành động
  - để đạt được trạng thái mục tiêu
- biện pháp hiệu suất: giảm thiểu tổng chi phí hành động



State-Space Search Problems

- environment:
  - static vs. dynamic
  - deterministic vs. non-deterministic vs. stochastic
  - fully vs. partially vs. not observable
  - discrete vs. continuous
  - single-agent vs. multi-agent
- problem solving method:
  - problem-specific vs. general vs. learning

State-Space Search Problems

- environment:
  - static vs. dynamic
  - deterministic vs. non-deterministic vs. stochastic
  - fully vs. partially vs. not observable
  - discrete vs. continuous
  - single-agent vs. multi-agent
- problem solving method:
  - problem-specific vs. general vs. learning

Example: State-Space Search Problems

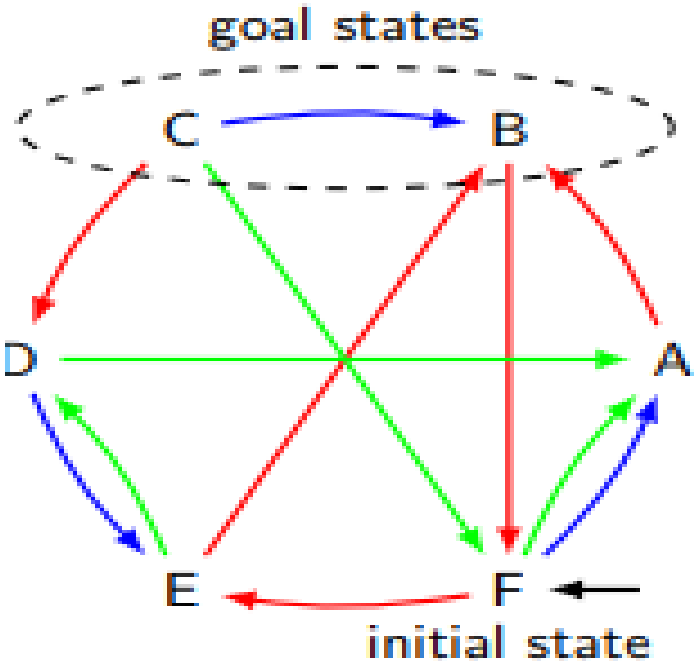
- vấn đề đồ chơi: câu đố kết hợp
  - (Khối Rubik, câu đố 15 ô, tòa tháp Hà Nội, . . .)
- lên lịch sự kiện, chuyến bay, nhiệm vụ sản xuất
- tối ưu hóa truy vấn trong cơ sở dữ liệu
- tối ưu hóa mã trong trình biên dịch
- xác minh phần mềm và phần cứng
- căn chỉnh trình tự trong tin sinh học
- lập kế hoạch tuyến đường (ví dụ: Google Maps)
- . . .
- hàng ngàn ví dụ thực tế

State-Space Search Problems

- nhận xét sơ bộ:
  - để nghiên cứu các vấn đề tìm kiếm một cách rõ ràng, chúng ta cần một mô hình chính thức
  - khái niệm cơ bản: không gian trạng thái
  - không gian trạng thái là đồ thị (có nhãn, có hướng)
  - đường dẫn đến các trạng thái mục tiêu biểu thị các giải pháp
  - đường dẫn ngắn nhất tương ứng với các giải pháp tối ưu

Example: State-Space

- Không gian trạng thái thường được mô tả dưới dạng đồ thị có hướng.
  - trạng thái: đỉnh đồ thị
  - chuyển tiếp: cung có nhãn (ở đây: màu thay vì nhãn)
  - trạng thái ban đầu: mũi tên đến
  - mục tiêu: được đánh dấu (ở đây: bằng hình elip đứt nét)
  - hành động: nhãn cung
  - chi phí hành động: được mô tả riêng (hoặc ngầm định = 1)



Example: State-Space

- Không gian trạng thái là một bộ 6 S = <S, A, cost, T, s_0, S^*> với
  - S: tập hợp hữu hạn các trạng thái
  - A: tập hợp hữu hạn các hành động
  - cost : chi phí hành động
  - T tập các chuyển tiếp
  - $s_0 \in S$ trạng thái ban đầu
  - $S^* \subseteq S$ tập hợp các trạng thái mục tiêu

- Không gian trạng thái là một bộ 6 S = <S, A, cost, T, s_0, S^*>
- Nếu có bộ 3 <s, a, s'> ∈ T ta gọi <s, a, s'> là trạng thái chuyển tiếp và được viết s → s'

German: Zustandsraum, Transitionssystem, Zust¨ande, Aktionen, Aktionskosten, Transitions-/Ubergangsrelation, deterministisch, ¨ Anfangszustand, Zielzust¨ande

State-Space

- **Hành động:** Cung cấp mô tả về tất cả các hành động có sẵn cho tác nhân.
- **Mô hình chuyển tiếp:** Mô tả về chức năng của từng hành động có thể được biểu diễn dưới dạng mô hình chuyển tiếp.
- **Chi phí đường đi:** Đây là hàm gán chi phí số cho mỗi đường đi.
- **Giải pháp:** Đây là một chuỗi hành động dẫn từ nút bắt đầu đến nút đích.
- **Giải pháp tối ưu:** Nếu một giải pháp có chi phí thấp nhất trong số tất cả các giải pháp.

State-Space Search

- Tìm kiếm không gian trạng thái là thuật toán tìm giải pháp trong không gian trạng thái hoặc chứng minh rằng không có giải pháp nào tồn tại.
- Trong tìm kiếm không gian trạng thái tối ưu, chỉ có thể trả về các giải pháp tối ưu.
- Với không gian trạng thái S = <S, A, cost, T, s_0, S^*>, những phương thức sau cần được xây dựng:
  - init(): tạo trạng thái ban đầu
    - result: trạng thái s0
  - is_goal(s): kiểm tra xem s có phải là trạng thái mục tiêu không
    - result: true nếu s ∈ S?; false nếu không
  - succ(s): tạo các hành động và hành động kế tiếp có thể áp dụng của s
    - result: chuỗi các cặp <a, s'> với $s \xrightarrow{a} s'$
  - cost(a): đưa ra chi phí của hành động a
    - result: cost(a) (∈ N0)

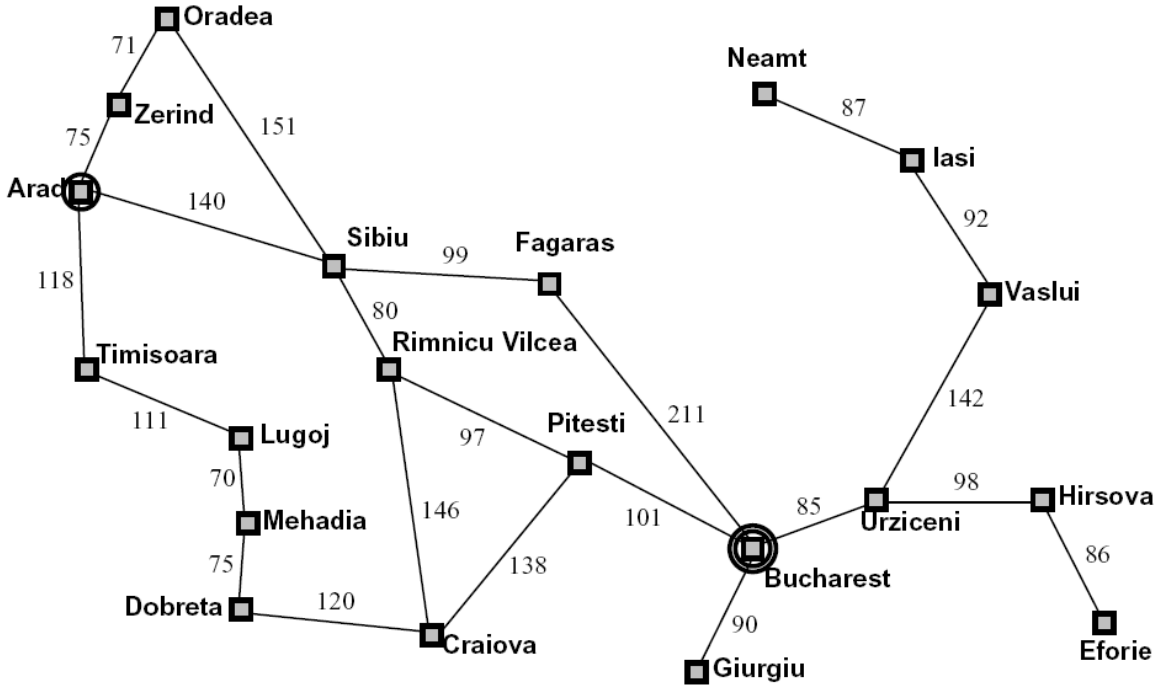
Search Problems

Các thành phần của một search problem

- Không gian trạng thái:
  - Các thành phố
- Tập các hành động và chi phí:
  - Hành động: di chuyển giữa 2 thành phố
  - Chi phí: khoảng cách giữa 2 thành phố
- Trạng thái bắt đầu:
  - Arad
- Trạng thái mục tiêu:
  - Is state == Bucharest?

- Giải pháp là gì?

Ví dụ: Traveling in Romania

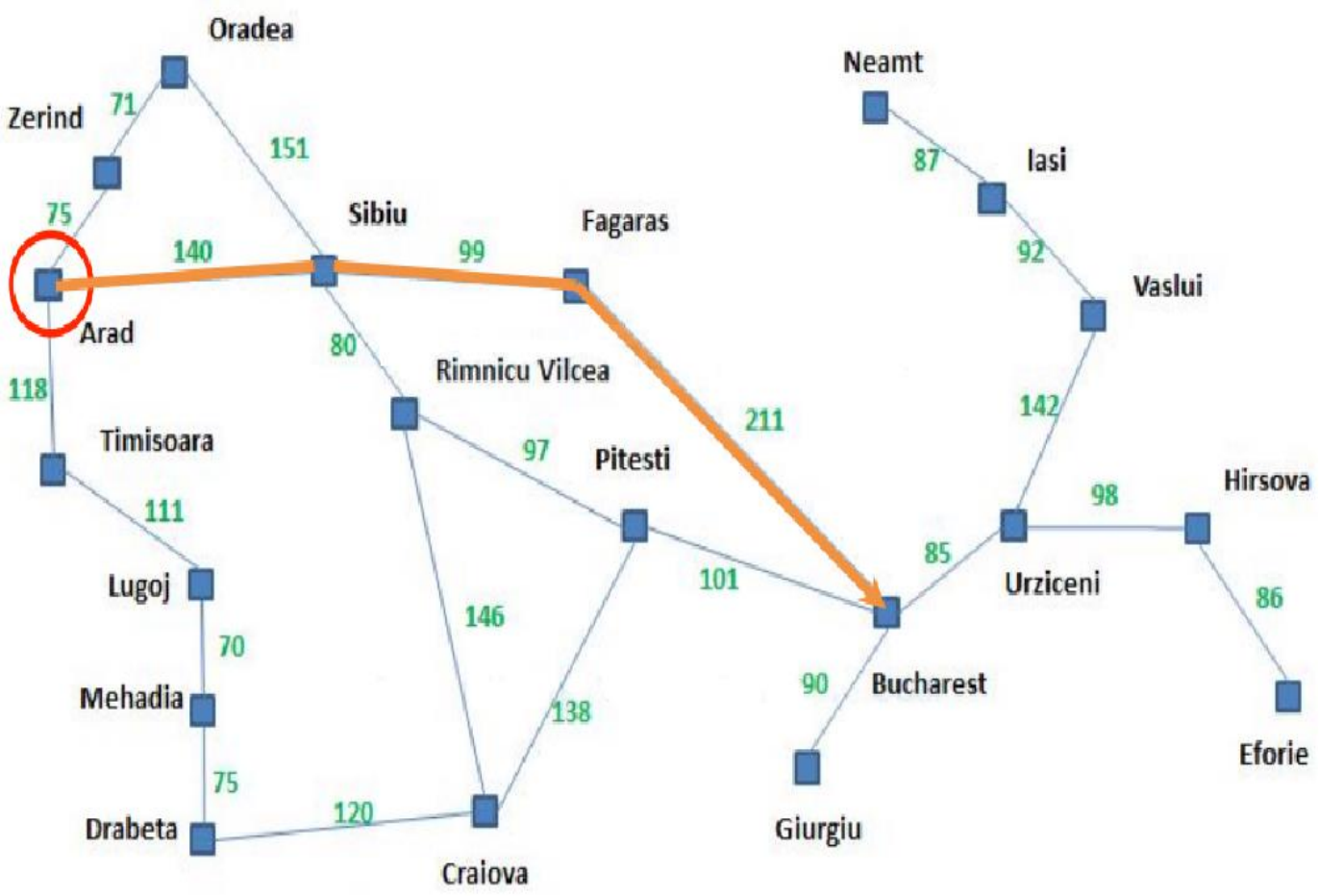


Search Problems

Các thành phần của một search problem

- Giải pháp là gì? e.g., Arad, Sibiu, Fagaras, Bucharest

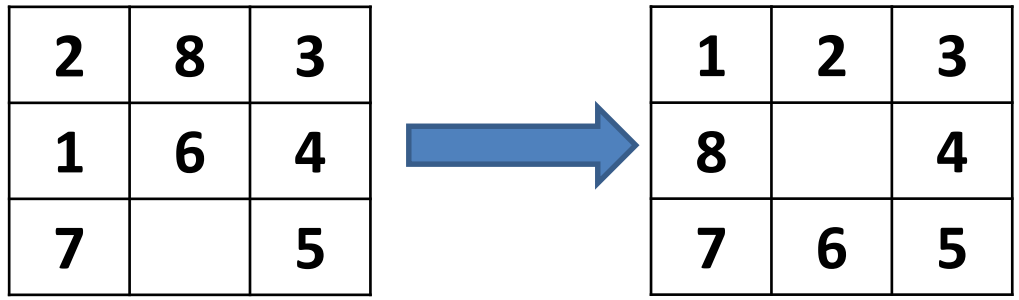
Ví dụ: Traveling in Romania



Search Problems

Bài toán ô chữ 8 số có thể phát biểu như môt search problem với các thành phần sau

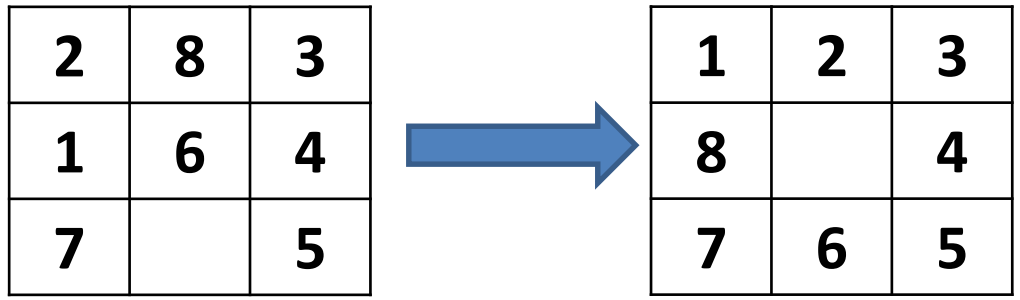
- Không gian trạng thái?
- Các hành động?
- Trạng thái xuất phát?
- Trạng thái mục tiêu?
- Chi phí?



Start state **Goal**

Search Problems

Bài toán ô chữ 8 số có thể phát biểu như môt search problem với các thành phần sau



Start state

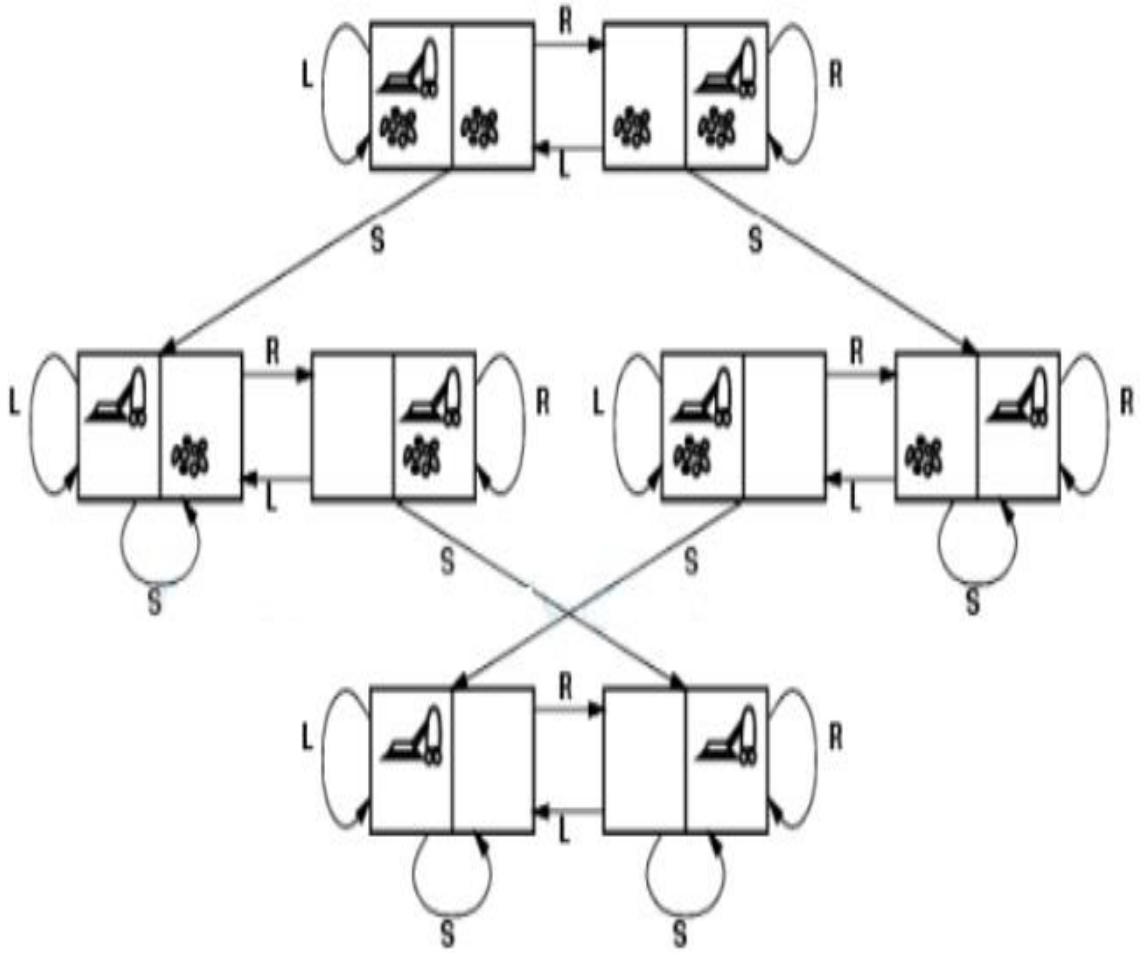
Goal

- Không gian trạng thái?
  - Các sắp xếp cụ thể vị trí các ô
- Các hành động và chi phí?
  - Hành động: di chuyển ô trống trái, phải, lên, xuống
  - Chi phí: Mỗi hành động có giá thành bằng 1
- Trạng thái xuất phát?
  - Trạng thái bên trái (ở hình trên)
- Trạng thái mục tiêu?
  - Trạng thái bên phải (ở hình trên)

Search Problems

Bài toán máy hút bụi có thể phát biểu như môt search problem với các thành phần sau

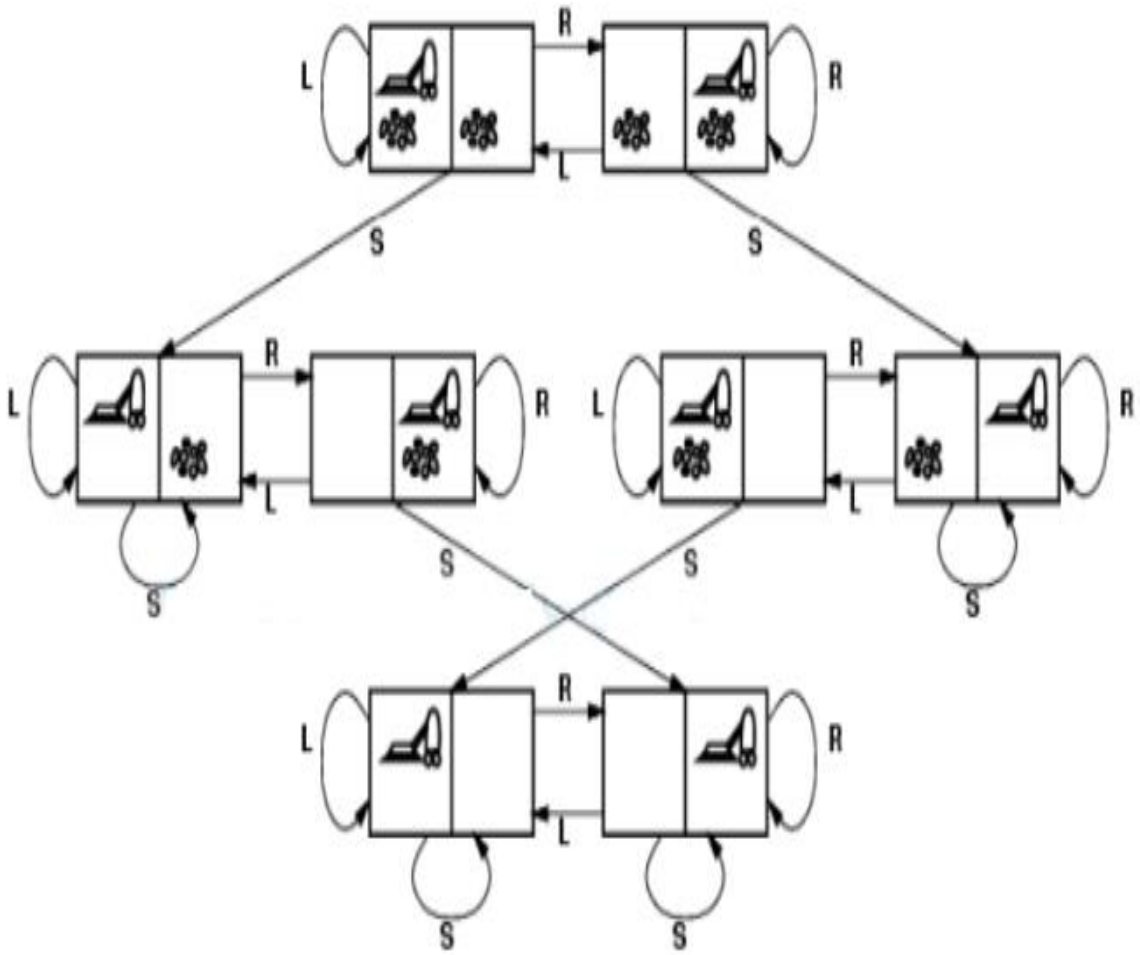
- Không gian trạng thái?
- Các hành động?
- Trạng thái xuất phát?
- Trạng thái mục tiêu?
- Chi phí?



Search Problems

Bài toán máy hút bụi có thể phát biểu như môt search problem với các thành phần sau

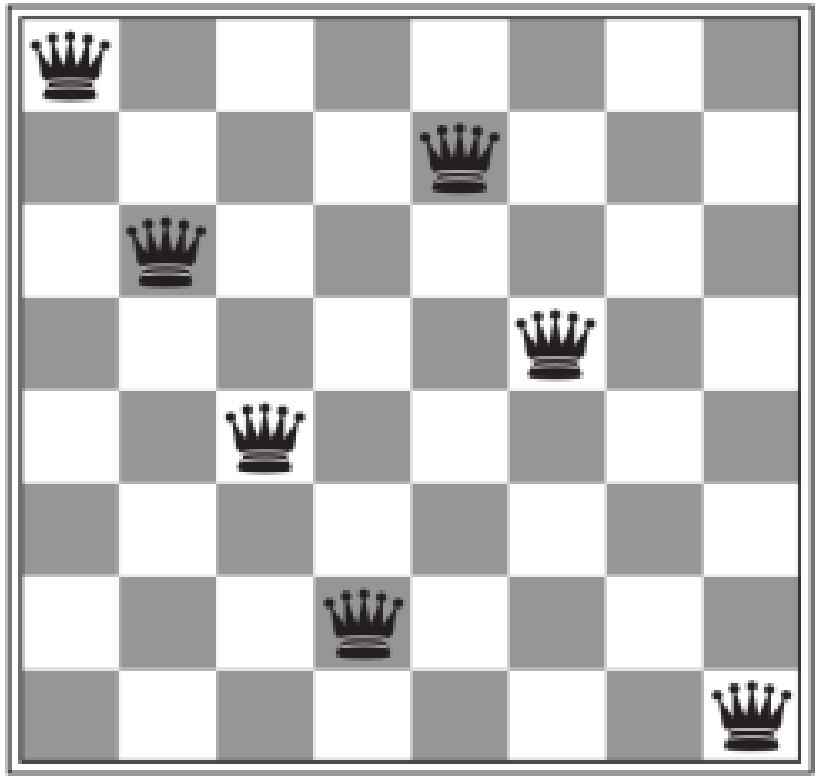
- Không gian trạng thái?
  - Chổ bẩn và vị trí máy hút bụi
- Các hành động và chi phí?
  - Hành động: Sang trái, sang phải, hút bụi, không làm gì
  - Chi phí: 1 (mỗi hành động), 0 (không làm gì)
- Trạng thái xuất phát?
  - Trạng thái bên trái (ở hình trên)
- Trạng thái mục tiêu?
  - Không còn vị trí nào bẩn



Search Problems

Bài toán 8 con hậu có thể phát biểu như môt search problem với các thành phần sau

- Không gian trạng thái?
- Các hành động và chi phí?
- Trạng thái xuất phát?
- Trạng thái mục tiêu?

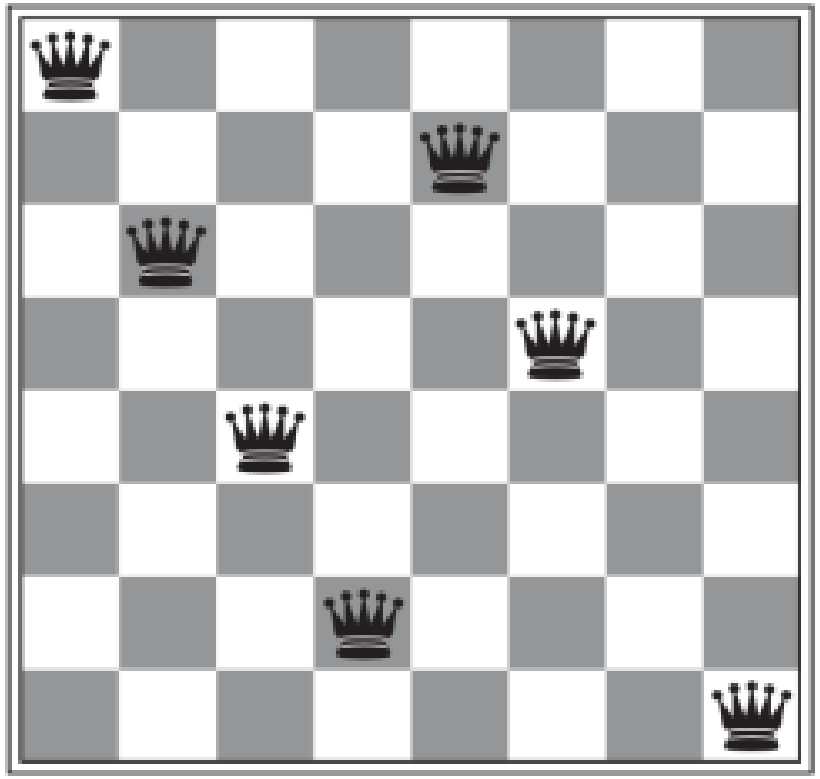


(© S. Russell & P. Norwig, AIMA)

Search Problems

Bài toán 8 con hậu có thể phát biểu như môt search problem với các thành phần sau

- Không gian trạng thái?
  - bất kỳ cách sắp xếp nào từ 0 đến 8 quân hậu trên b cờ
- Các hành động và chi phí?
  - Hành động: thêm một quân hậu vào bất kỳ ô trống nào
  - Chi phí: ?
- Trạng thái xuất phát?
  - không có quân hậu nào trên bàn cờ
- Trạng thái mục tiêu?
  - 8 quân hậu trên bàn cờ, không có quân hậu nào bị quân hậu khác tấn công



Search Problems

Các vấn đề về du lịch bằng máy bay theo một trang web lập kế hoạch du lịch:

- Không gian trạng thái?
- Các hành động và chi phí?
- Trạng thái xuất phát?
- Trạng thái mục tiêu?

Search Problems

Các vấn đề về du lịch bằng máy bay theo một trang web lập kế hoạch du lịch:

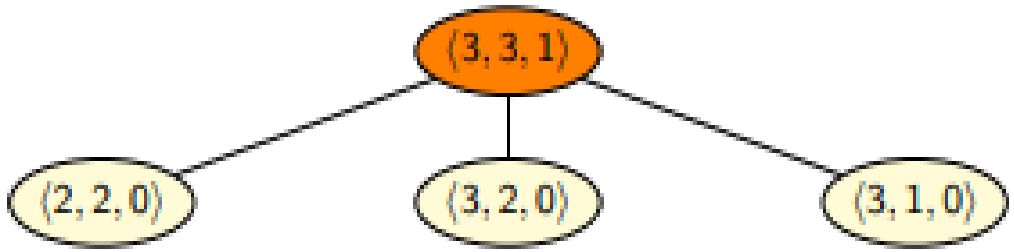
- Không gian trạng thái?
  - địa điểm, thời gian, giá vé cơ sở, chuyến bay nội địa/quốc tế, …
- Các hành động?
  - Đi bất kỳ chuyến bay nào từ vị trí hiện tại, ở bất kỳ hạng ghế nào, khởi hành sau giờ hiện tại, chừa đủ thời gian để trung chuyển trong sân bay nếu cần
  - Chi phí ?????
- Trạng thái xuất phát?
  - được chỉ định bởi truy vấn của người dùng
- Trạng thái mục tiêu?
  - đích đến chính xác do người dùng chỉ định

Example: Search Algorithm

- Bắt đầu với trạng thái ban đầu,

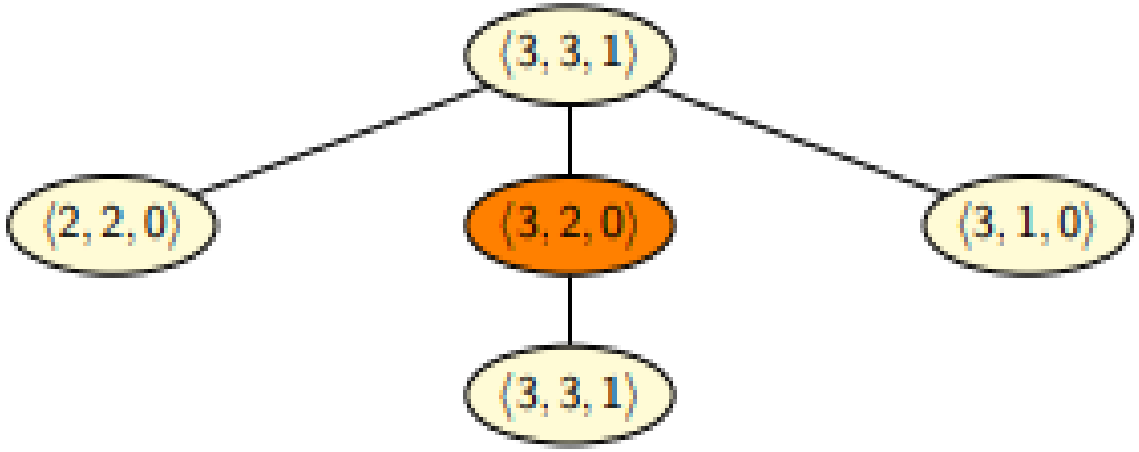
Example: Search Algorithm

- Bắt đầu với trạng thái ban đầu,
- mở rộng trạng thái nhiều lần bằng cách tạo ra các trạng thái kế thừa.



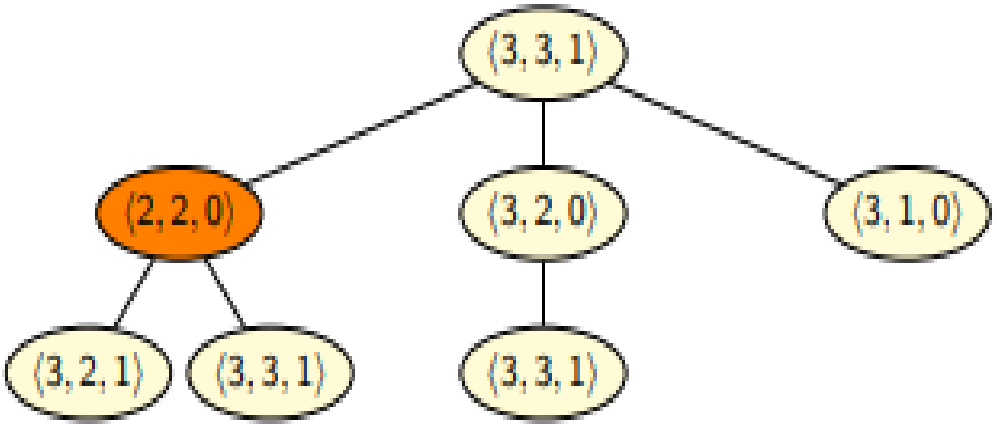
Example: Search Algorithm

- Bắt đầu với trạng thái ban đầu,
- mở rộng trạng thái nhiều lần bằng cách tạo ra các trạng thái kế thừa.



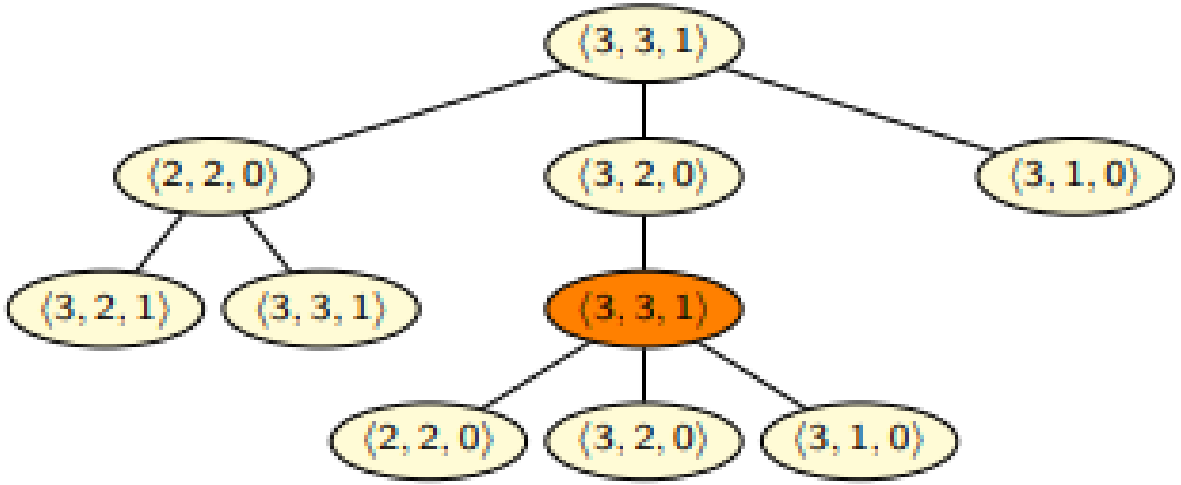
Example: Search Algorithm

- Bắt đầu với trạng thái ban đầu,
- mở rộng trạng thái nhiều lần bằng cách tạo ra các trạng thái kế thừa.



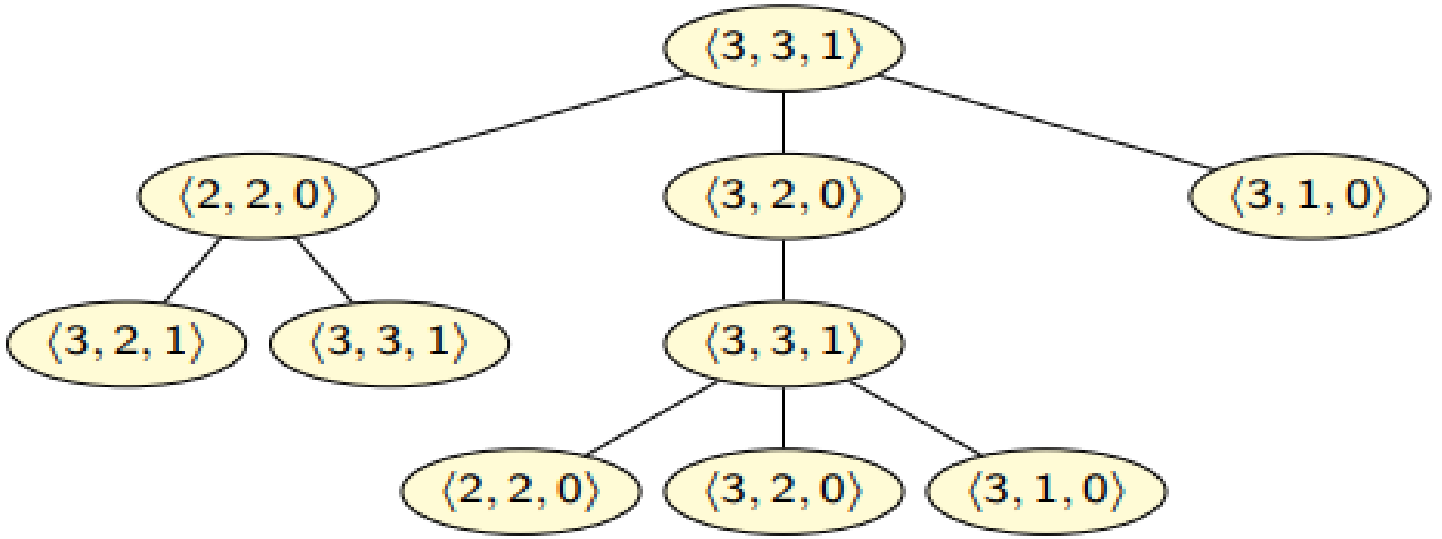
Example: Search Algorithm

- Bắt đầu với trạng thái ban đầu,
- mở rộng trạng thái nhiều lần bằng cách tạo ra các trạng thái kế thừa.



Example: Search Algorithm

- Bắt đầu với trạng thái ban đầu,
- mở rộng trạng thái nhiều lần bằng cách tạo ra các trạng thái kế thừa.
- Dừng lại khi trạng thái mục tiêu được mở rộng
- hoặc tất cả các trạng thái có thể đạt được đã được xem xét.

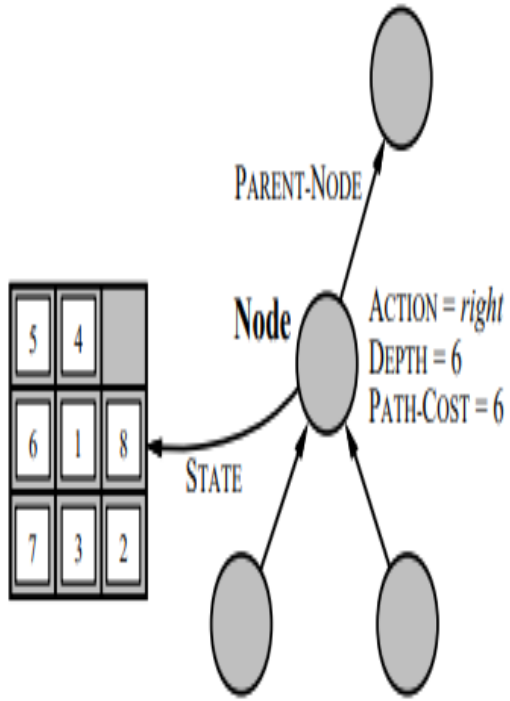


Cấu trúc dữ liệu cơ bản cho tìm kiếm

- Xem xét ba cấu trúc dữ liệu trừu tượng để tìm kiếm:
  - nút tìm kiếm: lưu trữ trạng thái đã đạt được, cách đạt được và chi phí là bao nhiêu
  - danh sách mở: sắp xếp hiệu quả các lá của cây tìm kiếm
  - danh sách đóng: ghi nhớ các trạng thái mở rộng để tránh việc mở rộng trùng lặp của cùng một trạng thái
    - các nút bên trong của cây tìm kiếm
- Không phải tất cả các thuật toán đều sử dụng cả ba cấu trúc dữ liệu, và đôi khi chúng được ngầm định (ví dụ: trong ngăn xếp CPU)

Search Nodes: Các thuộc tính

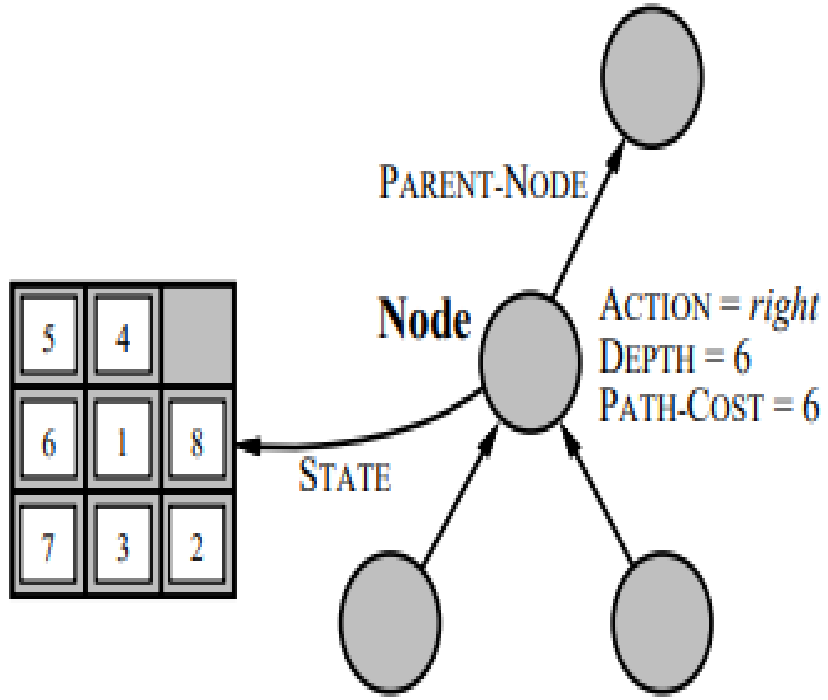
- Một nút tìm kiếm (gọi tắt là nút) lưu trữ trạng thái đã đạt được, cách đạt được và chi phí là bao nhiêu.
- Các thuộc tính của 1 nút tìm kiếm n:
  - n.state trạng thái liên kết với nút này
  - n.parent tìm kiếm nút đã tạo ra nút này (không có cho nút gốc)
  - n.action hành động dẫn từ n.parent đến n (không có cho nút gốc)
  - n.path-cost chi phí của đường dẫn từ trạng thái ban đầu đến n.state, đây kết quả từ việc theo các tham chiếu cha (theo truyền thống được biểu thị bằng g(n))



Search Nodes: Các phương thức

- make node

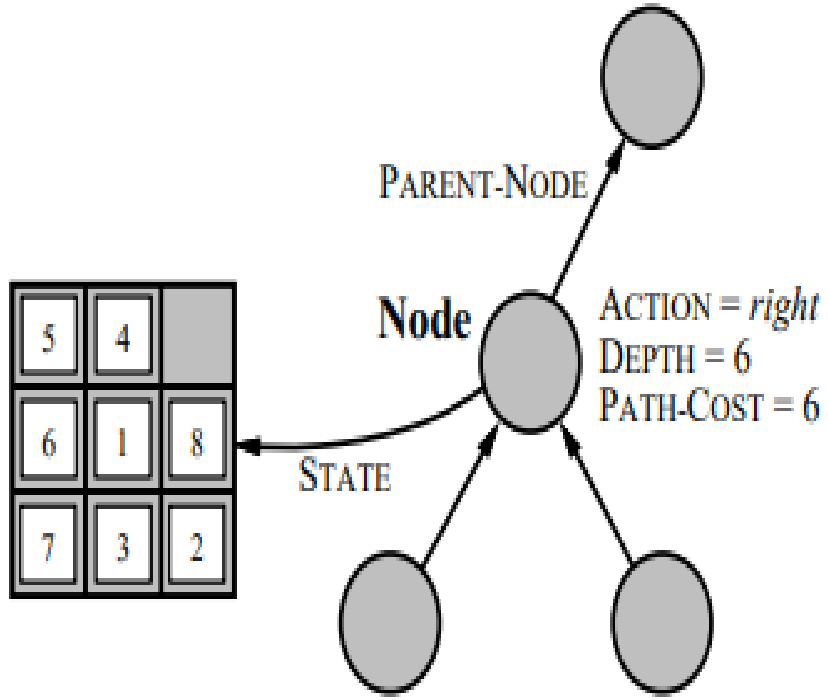
function make node(parent, action,state)
node := new SearchNode
node.state := state
node.parent := parent
node.action := action
node.path_cost := parent.path cost +
cost(action)
return node



Search Nodes: Các phương thức

- extract path

```
function extract_path(node)
path := <>
while node.parent ≠ none:
        path.append(node.action)
        node := node.parent
path.reverse()
return path
```



Open List

- Danh sách mở (còn gọi là biên) tổ chức các lá của cây tìm kiếm.
- Nó phải hỗ trợ hai thao tác hiệu quả:
  - xác định và xóa nút tiếp theo để mở rộng
  - chèn một nút mới là nút ứng viên để mở rộng

Note: bất chấp tên gọi, việc triển khai danh sách mở dưới dạng danh sách đơn giản thường là một ý tưởng rất tệ.

Open List: Các thương thức

- open.is empty() kiểm tra xem danh sách mở có trống không
- open.pop() xóa và trả về nút tiếp theo để mở rộng
- open.insert(n) chèn nút n vào danh sách mở

- Note:
  - Các thuật toán tìm kiếm khác nhau sử dụng các chiến lược khác nhau để quyết định nút nào sẽ trả về trong open.pop.
  - Việc lựa chọn cấu trúc dữ liệu phù hợp phụ thuộc vào chiến lược này (ví dụ: stack, deque, min-heap).

Closed List

- Danh sách đóng ghi nhớ các trạng thái mở rộng để tránh việc mở rộng trùng lặp cùng một trạng thái.
- Nó phải hỗ trợ hai thao tác hiệu quả:
  - chèn một nút có trạng thái chưa có trong danh sách đóng
  - kiểm tra xem nút có trạng thái nhất định có nằm trong danh sách đóng không; nếu có, trả về

Note: bất chấp tên gọi, việc triển khai danh sách đóng dưới dạng danh sách đơn giản thường là một ý tưởng rất tệ. (Tại sao?)

Closed List: Các thương thức

- closed.insert(n) chèn nút n vào closed;
- nếu một nút có trạng thái này đã tồn tại trong closed,
- thay thế nó
- closed.lookup(s) kiểm tra xem một nút có trạng thái s có tồn tại trong danh sách closed không; nếu có, trả về nó; nếu không, trả về none

- Note: Bảng băm với trạng thái làm khóa có thể đóng vai trò là triển khai hiệu quả của danh sách đóng.

Thuật toán tìm kiếm tổng quát

- **Nguyên lý chung:**

- **Ví dụ:**

3	1	6
5		8
2	7	4

Trạng thái xuất phát

	1	2
3	4	5
6	7	8

Trạng thái đích

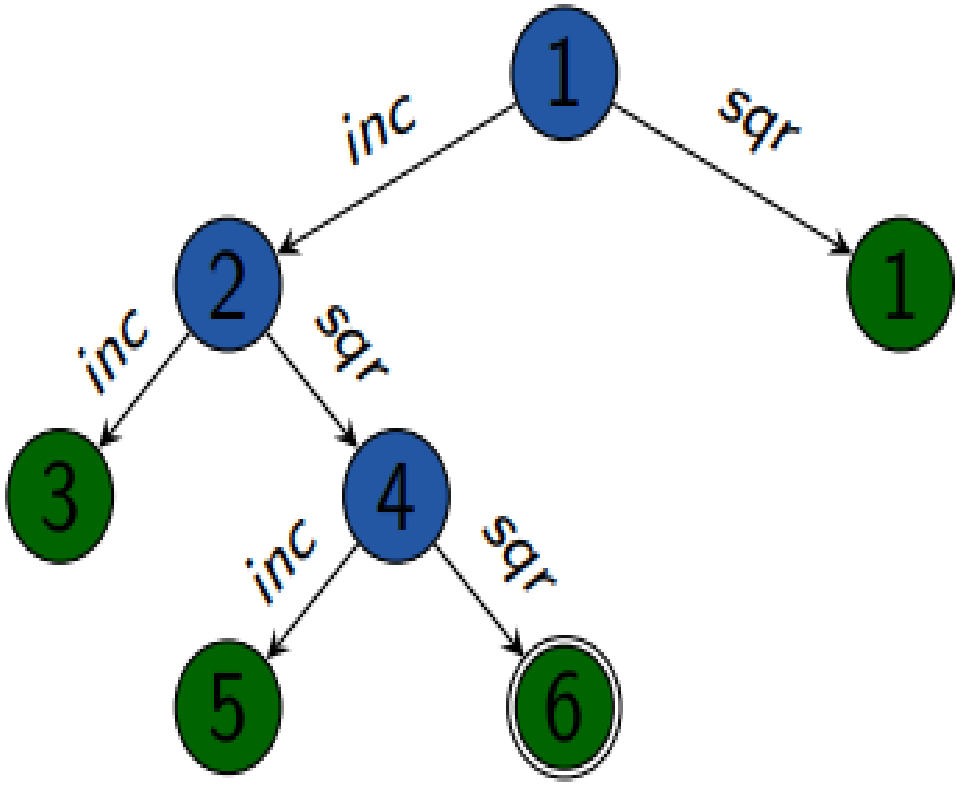
Search Algorithm

- tạo cây tìm kiếm theo từng bước:
  - bắt đầu với trạng thái ban đầu,
  - mở rộng trạng thái nhiều lần bằng cách tạo ra các trạng thái kế thừa của nó (trạng thái nào thì phụ thuộc vào thuật toán tìm kiếm được sử dụng)
  - dừng khi trạng thái mục tiêu được mở rộng (đôi khi: được tạo)
  - hoặc tất cả các trạng thái có thể đạt được đã được xem xét

Search Algorithm

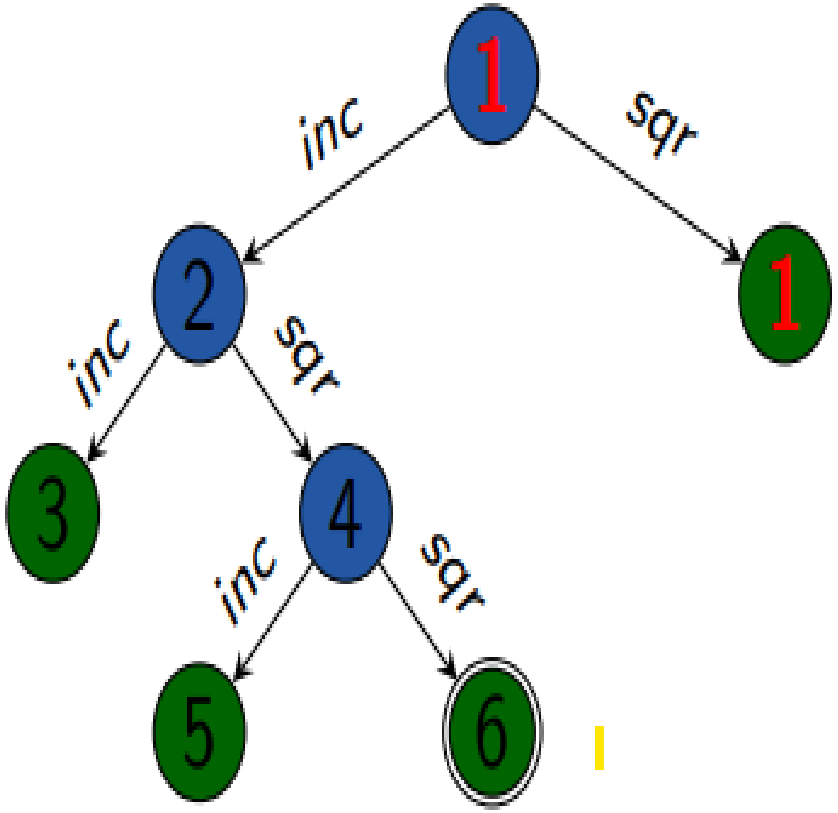
- Tree search
- Graph search

Tree Search: General Idea



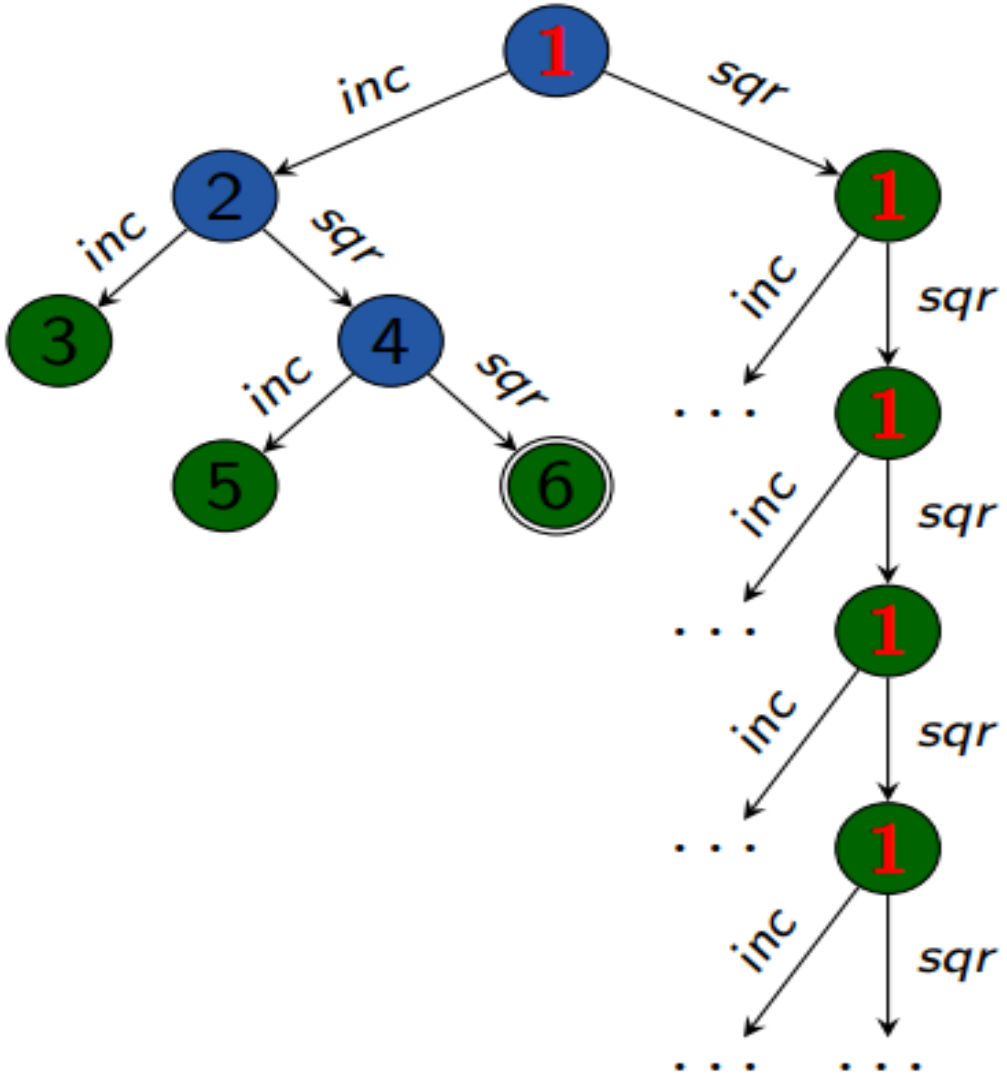
- các đường dẫn có thể được khám phá được sắp xếp theo cây
- các nút tìm kiếm tương ứng 1:1 với các đường dẫn từ trạng thái ban đầu

Tree Search: General Idea



- các đường dẫn có thể được khám phá được sắp xếp theo cây
- các nút tìm kiếm tương ứng 1:1 với các đường dẫn từ trạng thái ban đầu
- Có thể có nhiều nút có trạng thái giống hệt nhau

Tree Search: General Idea



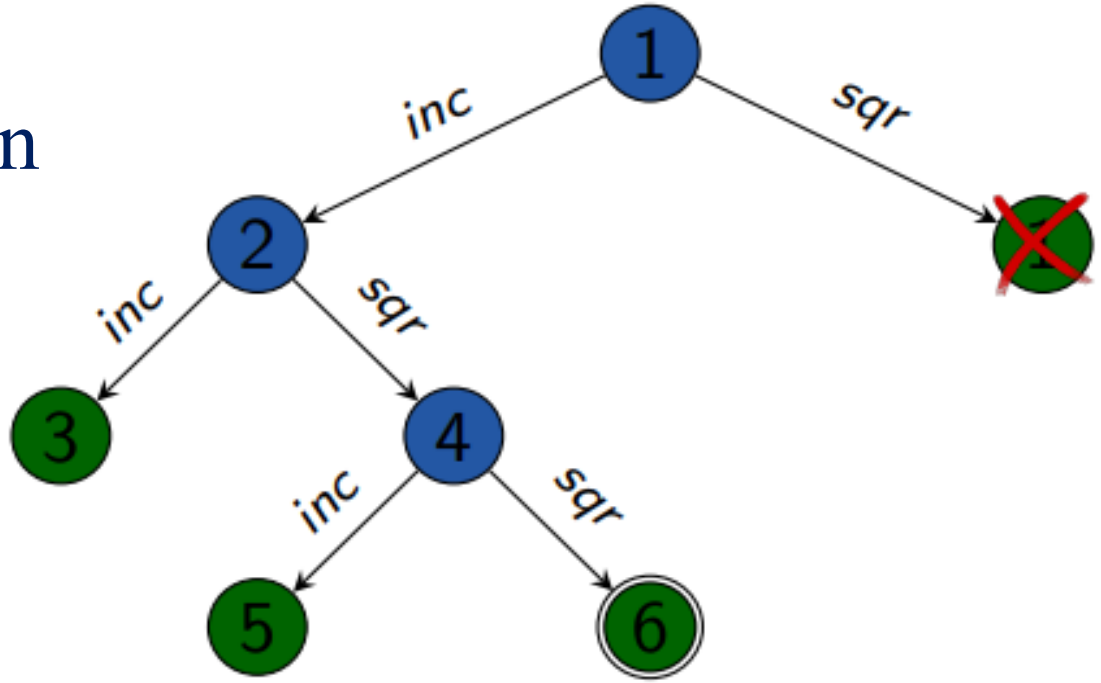
- các đường dẫn có thể được khám phá được sắp xếp theo cây
- các nút tìm kiếm tương ứng 1:1 với các đường dẫn từ trạng thái ban đầu
- Có thể có nhiều nút có trạng thái giống hệt nhau
- có thể có độ sâu không giới hạn

Generic Tree Search Algorithm

```
open := new OpenList
open.insert(make root node())
while not open.is empty():
        n := open.pop()
        if is goal(n.state):
                return extract path(n)
        for each ⟨a,s′ ⟩ ∈ succ(n.state):
                n′ := make node(n, a,s′ )
                open.insert(n′ )
return unsolvable
```

Graph Search

- sự khác biệt với tìm kiếm cây:
  - nhận dạng các bản sao: khi một trạng thái đạt được trên nhiều đường dẫn, chỉ giữ một nút tìm kiếm
  - các nút tìm kiếm tương ứng 1:1 với các trạng thái có thể đạt được
- độ sâu của cây tìm kiếm bị giới hạn

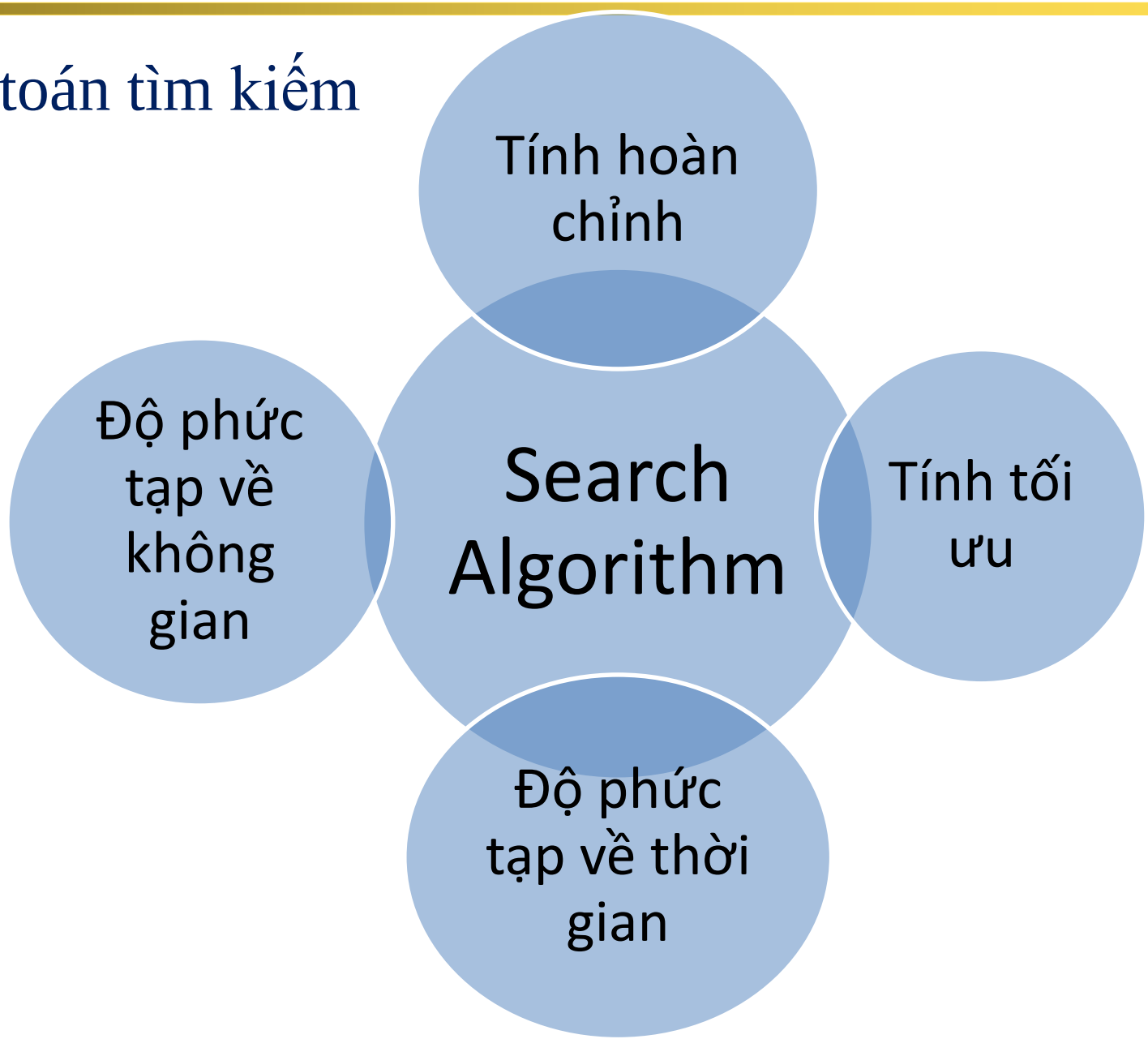


Generic Graph Search Algorithm

```
open := new OpenList
open.insert(make root node())
closed := new ClosedList
while not open.is empty():
        n := open.pop()
        if closed.lookup(n.state) = none:
                closed.insert(n)
                if is goal(n.state):
                        return extract path(n)
                for each ⟨a,s′⟩ ∈ succ(n.state):
                        n′ := make node(n, a,s′ )
                        open.insert(n′ )
return unsolvable
```

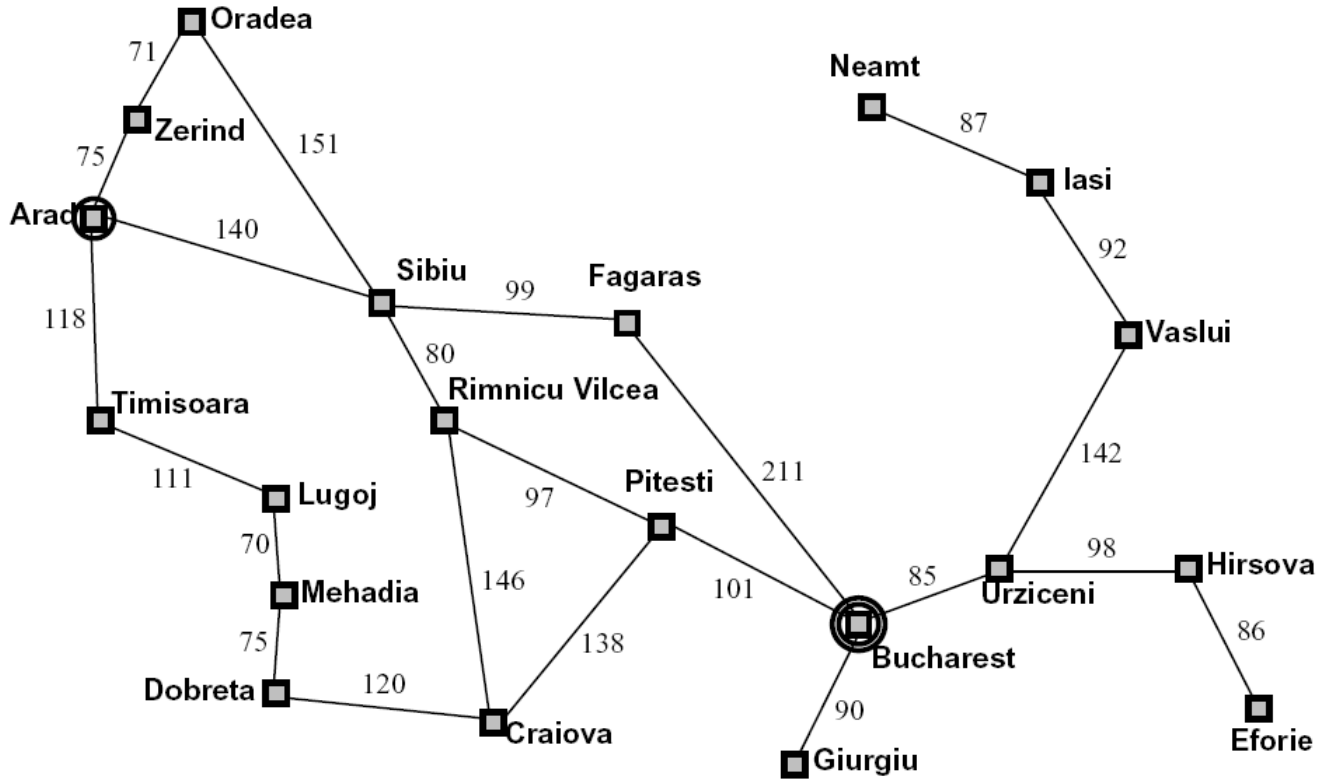
Đánh giá Search Algorithm

- Thuộc tính của thuật toán tìm kiếm



Assignment

- Path planning using Roma city map
  - ➢ Starting state: **Arad**
  - ➢ Goal state: **Bucharest**
- Building a searching tree (to search a solution)



trạng thái xuất phát

2	6	5
	8	7
4	3	1

trạng thái đích

1	2	3
4	5	6
7	8	

Uninformed search algorithms

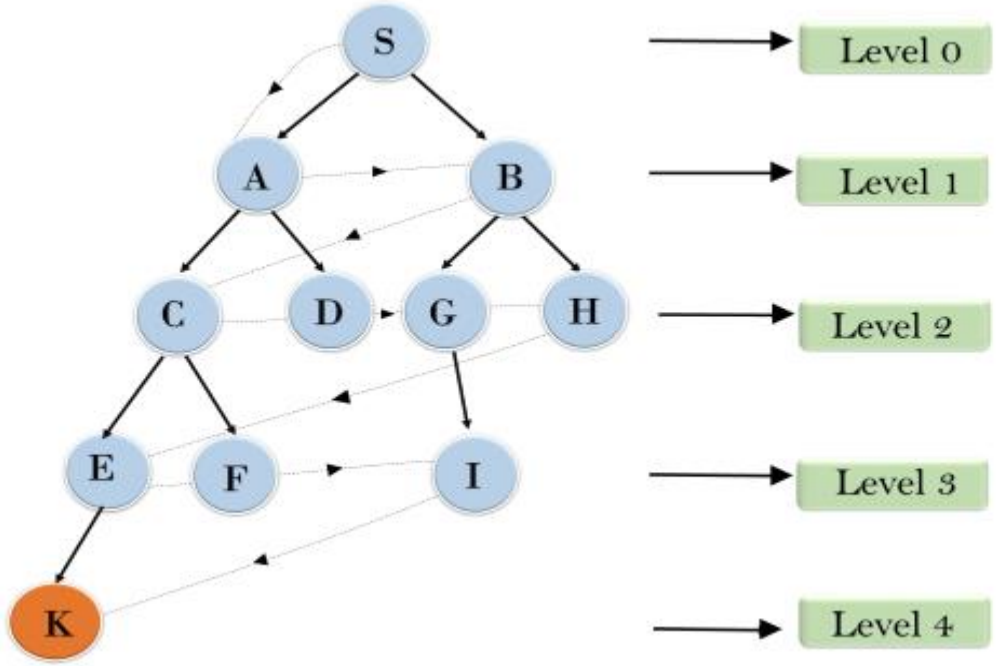
Breadth-first search

Depth-first search

Iterative Deepening

Uniform Cost Search

Breadth-first Search

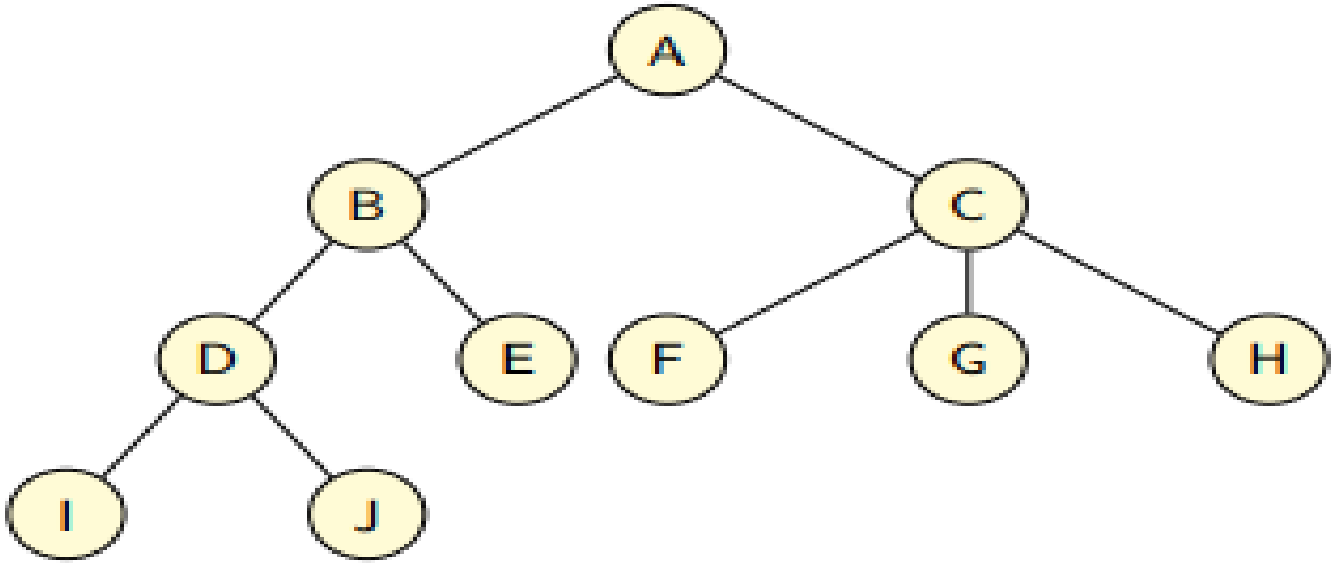


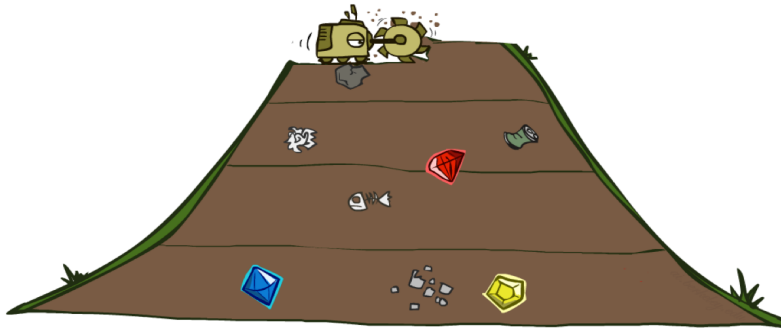
Tìm kiếm theo chiều rộng mở rộng các nút theo thứ tự tạo (FIFO).

ví dụ, danh sách mở dưới dạng danh sách liên kết hoặc deque

Breadth-first Search

Tìm kiếm theo chiều rộng mở rộng các nút theo thứ tự tạo (FIFO).
ví dụ, danh sách mở dưới dạng danh sách liên kết hoặc deque

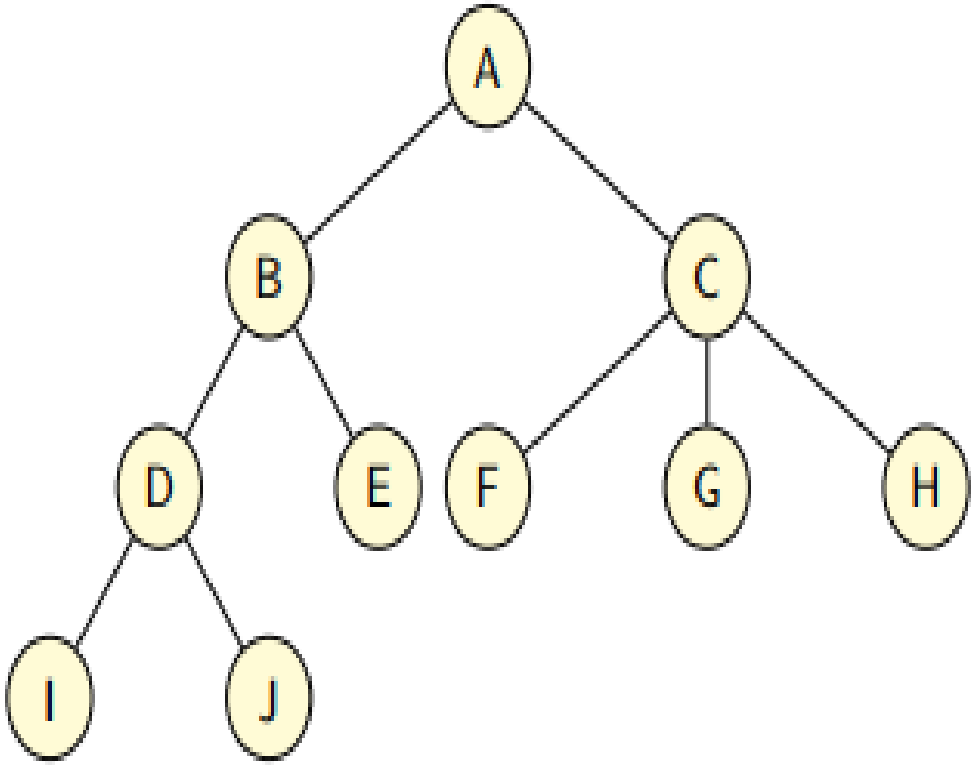




Breadth-first Search

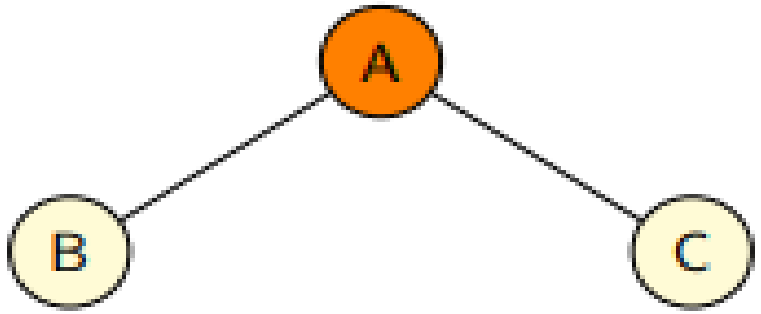
Tìm kiếm theo chiều rộng mở rộng các nút theo thứ tự tạo (FIFO).
ví dụ, danh sách mở dưới dạng danh sách liên kết hoặc deque

open: A

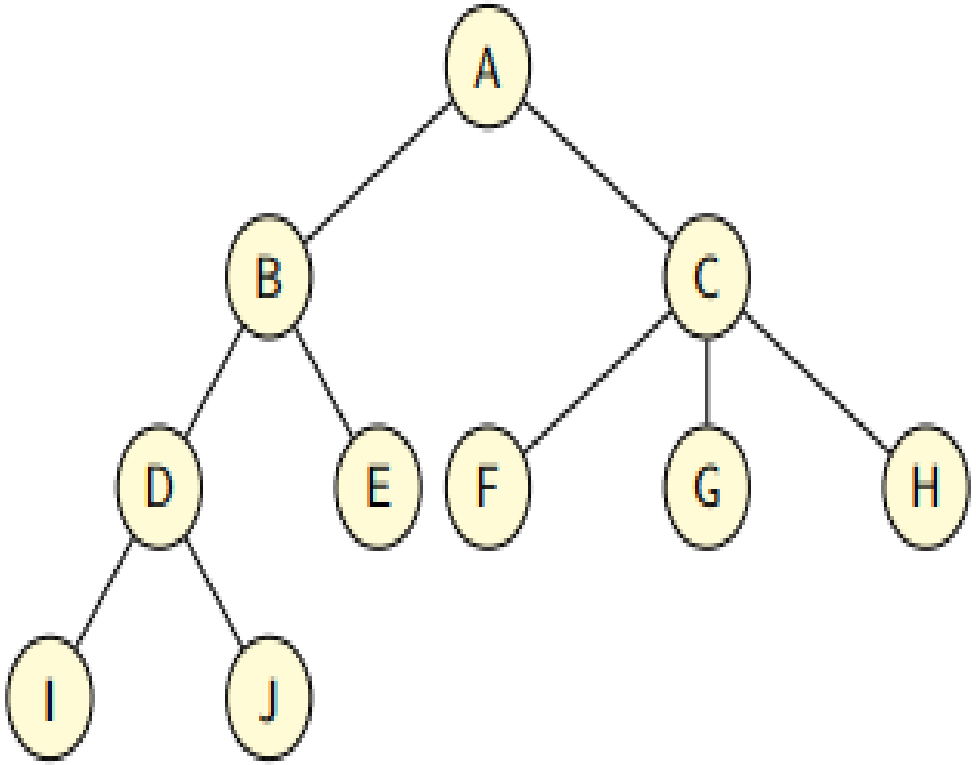


Breadth-first Search

Tìm kiếm theo chiều rộng mở rộng các nút theo thứ tự tạo (FIFO).
ví dụ, danh sách mở dưới dạng danh sách liên kết hoặc deque

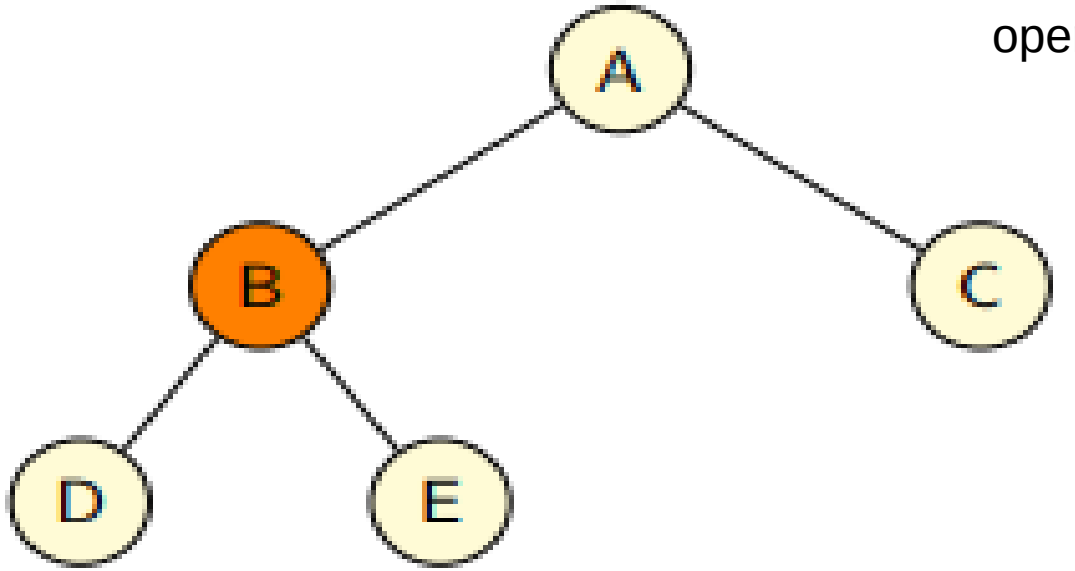


open: B, C

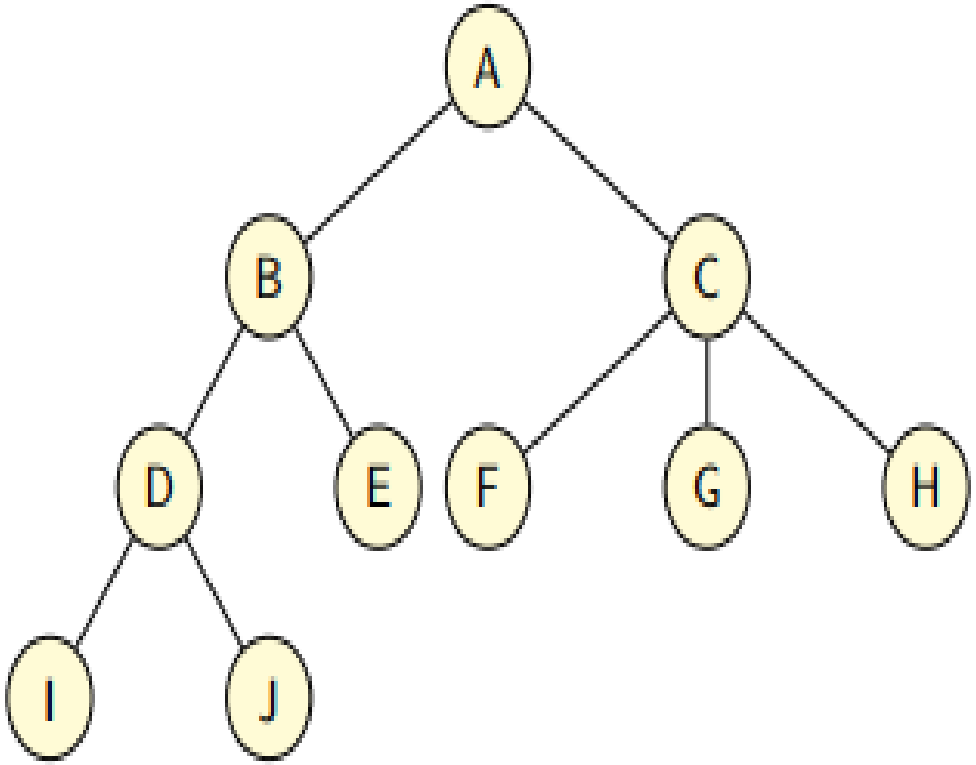


Breadth-first Search

Tìm kiếm theo chiều rộng mở rộng các nút theo thứ tự tạo (FIFO).
ví dụ, danh sách mở dưới dạng danh sách liên kết hoặc queue

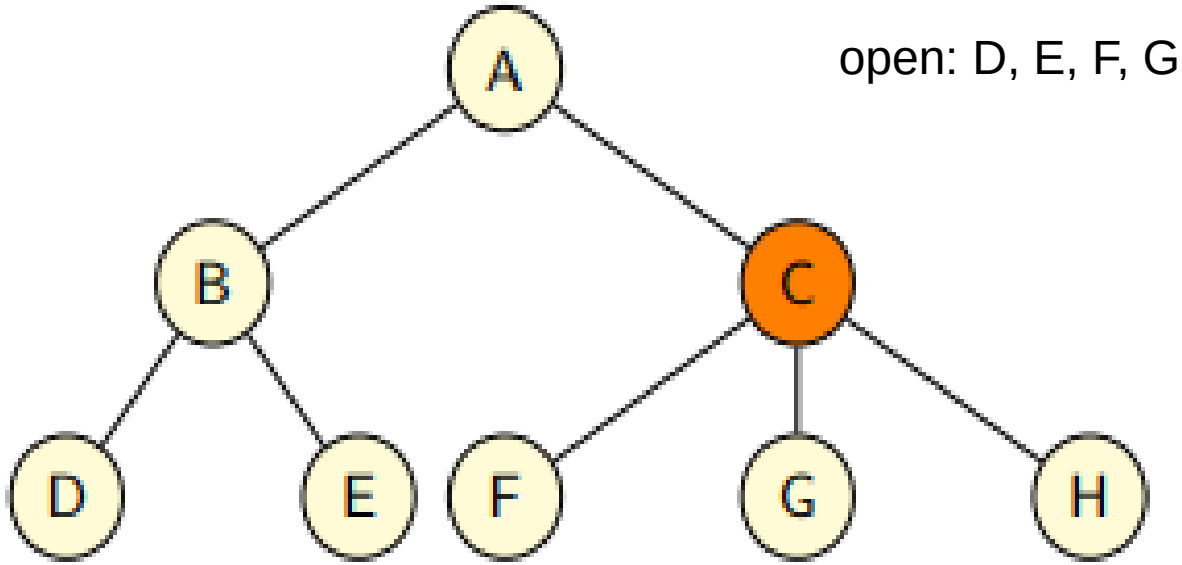


open: C, D, E

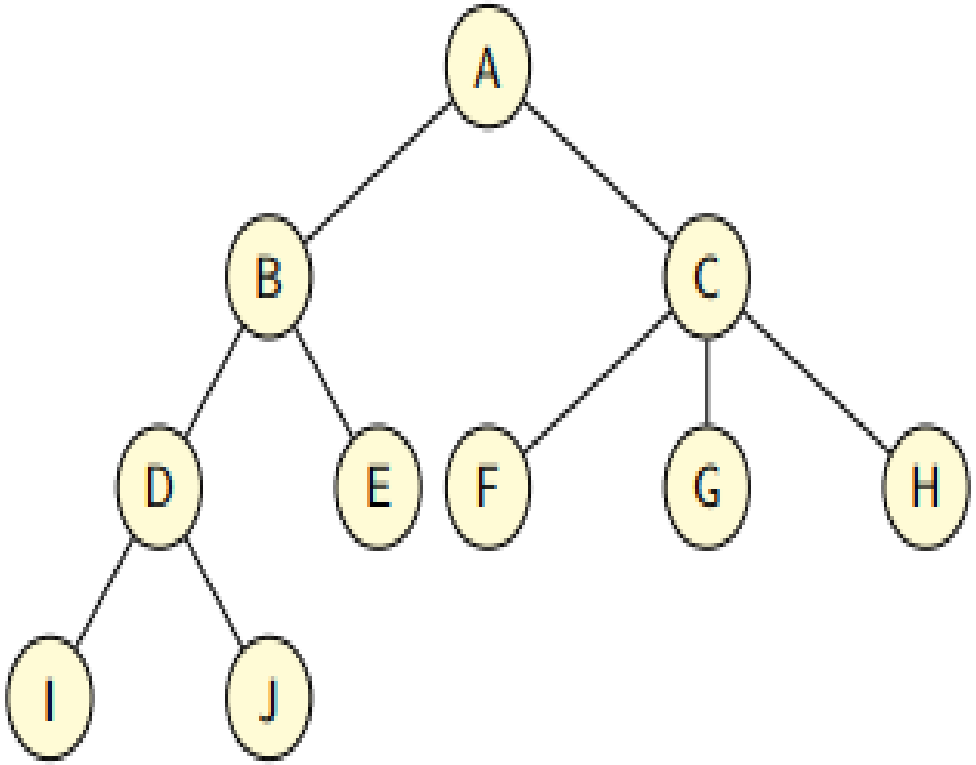


Breadth-first Search

Tìm kiếm theo chiều rộng mở rộng các nút theo thứ tự tạo (FIFO).
ví dụ, danh sách mở dưới dạng danh sách liên kết hoặc deque

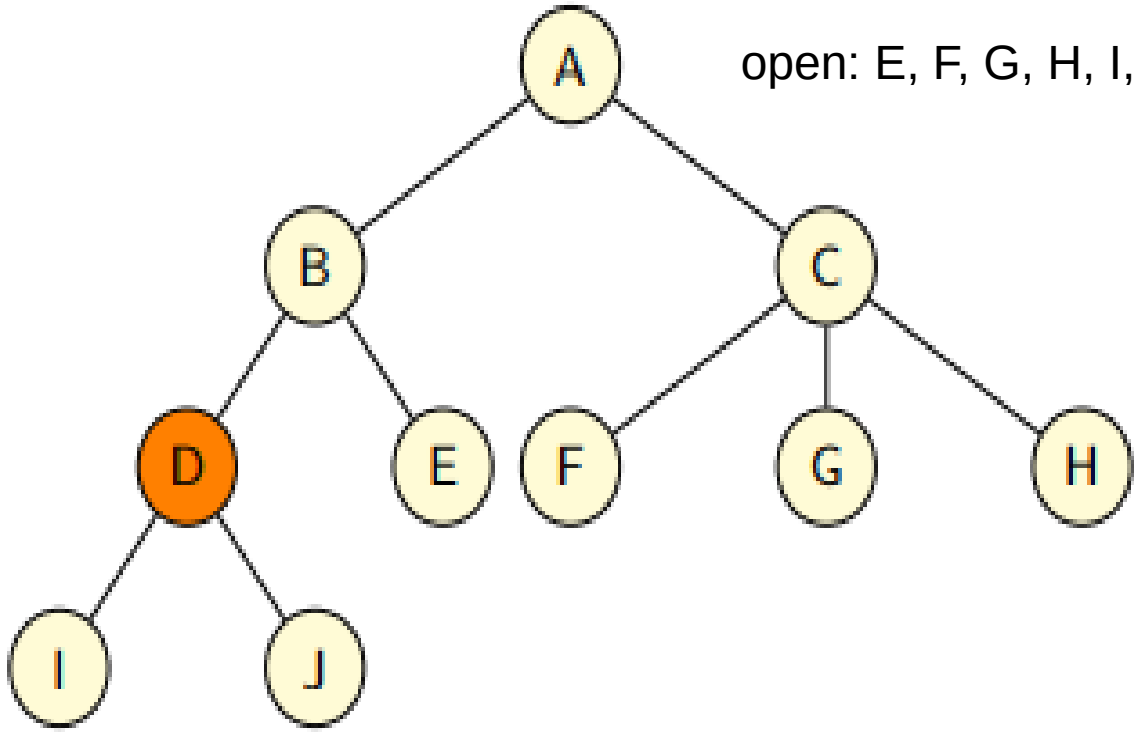


open: D, E, F, G, H

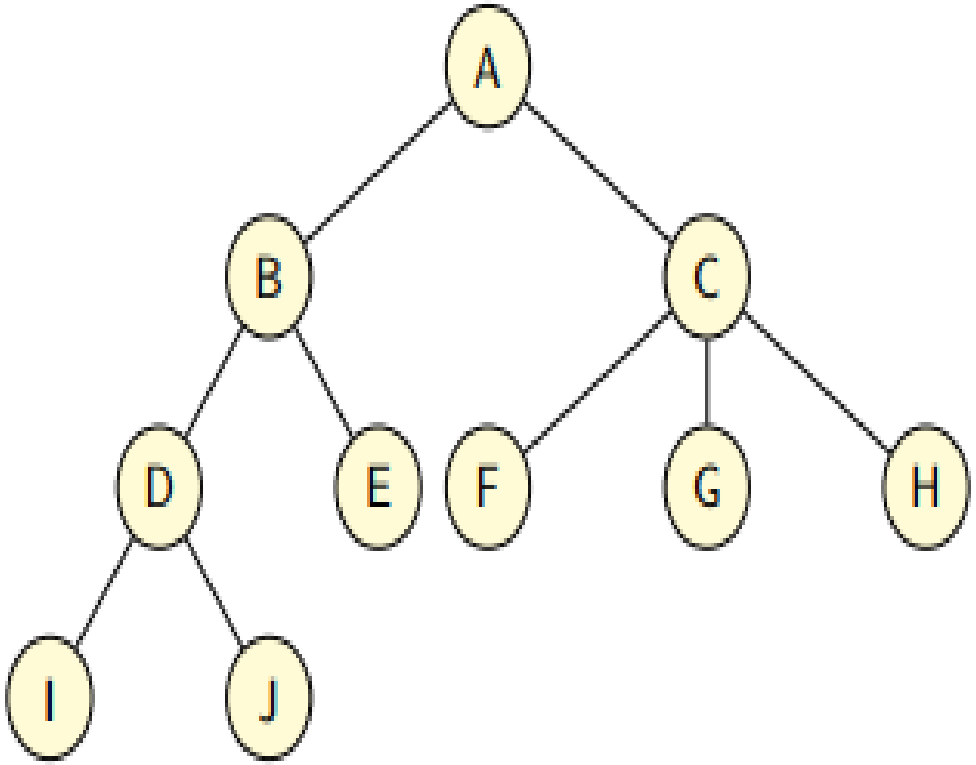


Breadth-first Search

Tìm kiếm theo chiều rộng mở rộng các nút theo thứ tự tạo (FIFO).
ví dụ, danh sách mở dưới dạng danh sách liên kết hoặc deque

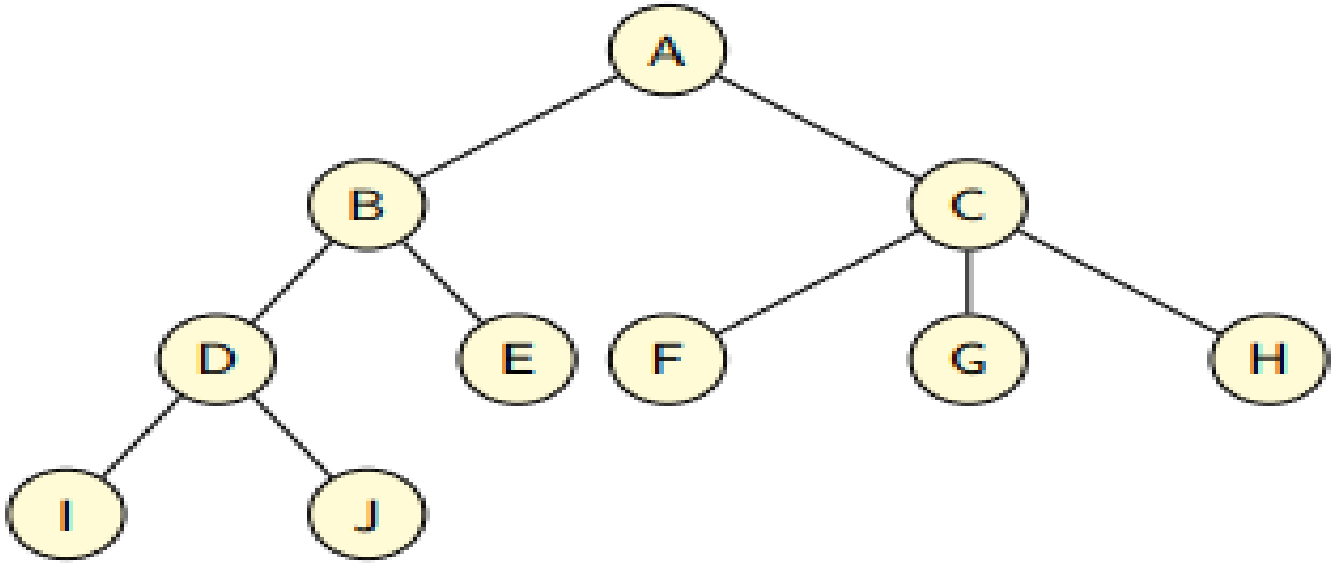


open: E, F, G, H, I, J



Breadth-first Search

Tìm kiếm theo chiều rộng mở rộng các nút theo thứ tự tạo (FIFO).
ví dụ, danh sách mở dưới dạng danh sách liên kết hoặc queue



tìm kiếm không gian trạng thái từng lớp
luôn tìm thấy trạng thái mục tiêu nông nhất đầu tiên

Breadth-first Search

Tìm kiếm theo chiều rộng có thể được thực hiện mà không cần loại bỏ trùng lặp (như tìm kiếm cây)

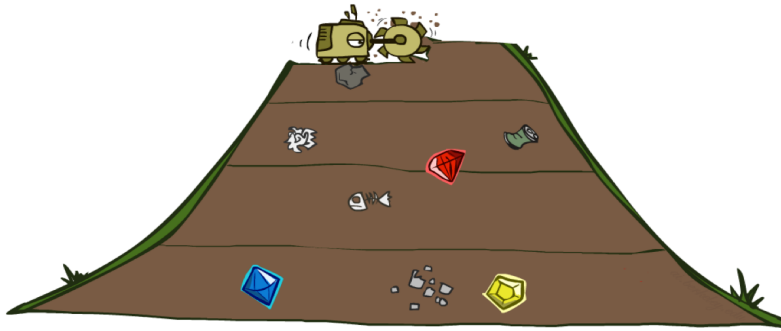
BFS-Tree

hoặc có loại bỏ trùng lặp (như tìm kiếm đồ thị)

BFS-Graph

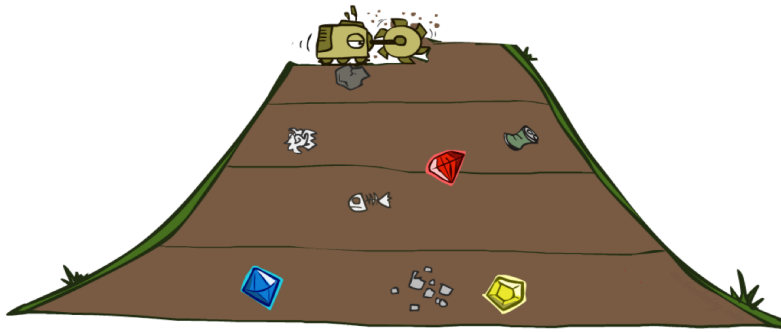
(BFS = breadth-first search).

=>xem xét cả hai biến thể.



BFS-Tree

```
if is goal(init()):
        return <>
open := new queue
open.push_back(make root node())
while not open.is empty():
        n := open.pop_front()
        for each <a,s'>∈ succ(n.state):
                n':= make_node(n, a,s')
                if is goal(s'):
                        return extract path(n')
                open.push_back(n')
return unsolvable
```

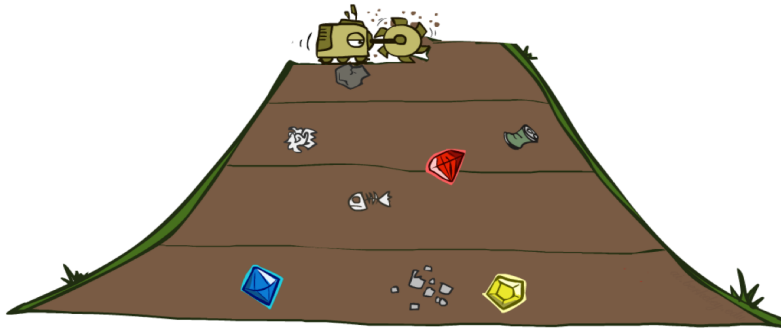


BFS-Graph

```
if is goal(init()):
        return <>
open := new queue
open.push_back(make_root_node())
closed := new HashSet
closed.insert(init())
while not open.is empty():
        n := open.pop_front()
        for each <a,s'>∈ succ(n.state):
                n':= make node(n, a,s')
                if is goal(s'):
                        return extract_path(n')
                if s' ∉ closed:
                        closed.insert(s')
                        open.push_back(n')
return unsolvable
```

Thuộc tính của Breadth-first Search

Thuộc tính của tìm kiếm theo chiều rộng:
BFS-Tree là một nửa hoàn chỉnh. (Tại sao?)
BFS-Graph là hoàn chỉnh. (Tại sao?)
BFS (cả hai biến thể) là tối ưu nếu tất cả các hành động có cùng chi phí (Tại sao?),
độ phức tạp: các trang tiếp theo



Breadth-first Search: Complexity

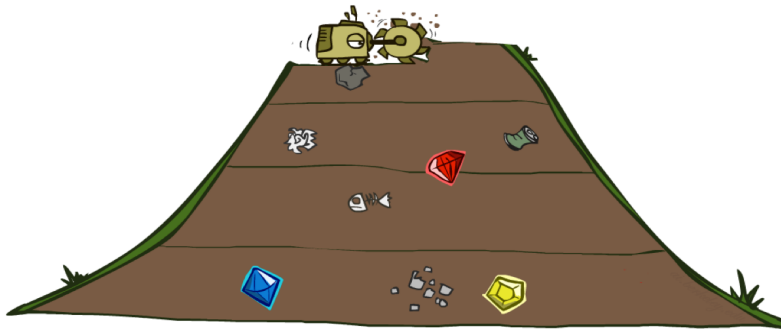
Cho b là số phân nhánh và d là độ dài giải pháp tối thiểu của không gian trạng thái đã cho (độ sâu của cây hoặc graph).

Cho b ≥ 2. Khi đó:

Độ phức tạp thời gian là:

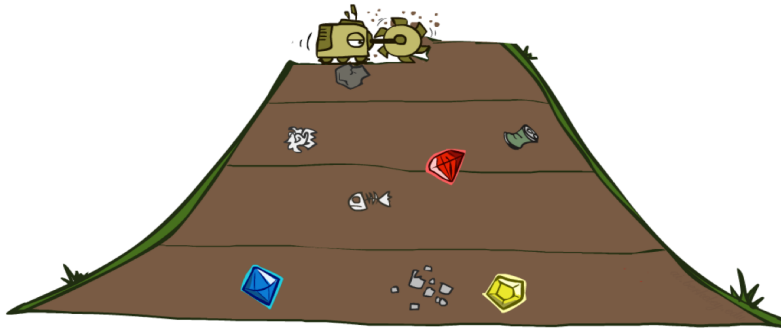
$1 + b + b^2 + b^3 + \cdots + b^d = O(b^d)$

Độ phức tạp không gian là: $O(b^d)$



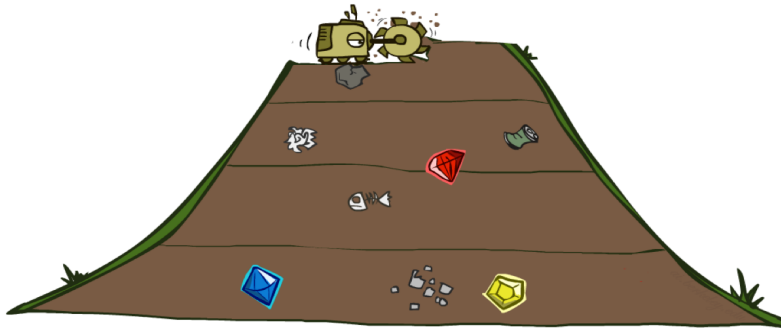
BFS-Tree or BFS-Graph?

- BFS-Tree hay BFS-Graph tốt hơn?
- lợi thế của BFS-Graph:
  - hoàn thành
  - hiệu quả hơn nhiều (!) nếu có nhiều bản sao:



BFS-Tree or BFS-Graph?

- BFS-Tree hay BFS-Graph tốt hơn?
- lợi thế của BFS-Graph:
  - hoàn thành
  - hiệu quả hơn nhiều (!) nếu có nhiều bản sao:
- lợi thế của BFS-Tree:
  - đơn giản hơn
  - ít tốn kém hơn (thời gian/không gian) nếu có ít bản sao

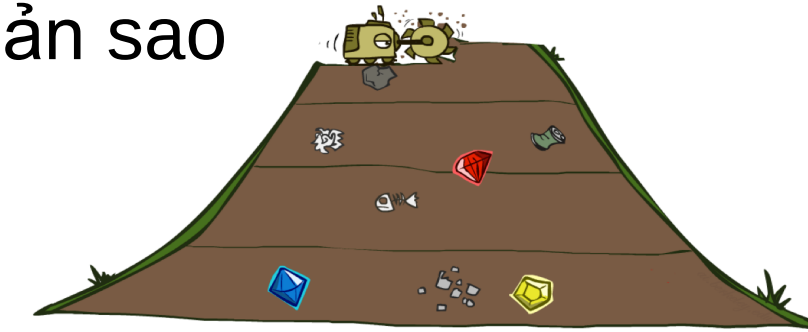


BFS-Tree or BFS-Graph?

- BFS-Tree hay BFS-Graph tốt hơn?
- lợi thế của BFS-Graph:
  - hoàn thành
  - hiệu quả hơn nhiều (!) nếu có nhiều bản sao:
- lợi thế của BFS-Tree:
  - đơn giản hơn
  - ít tốn kém hơn (thời gian/không gian) nếu có ít bản sao

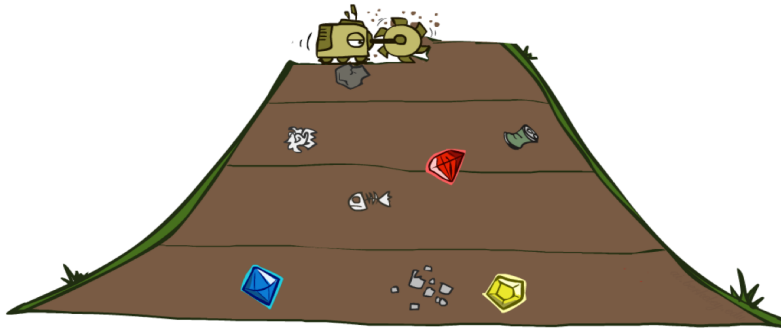
Kết luận

BFS-Graph thường được ưa chuộng hơn, trừ khi chúng ta biết rằng có một số lượng không đáng kể các bản sao trong không gian trạng thái đã cho.



Tóm lại

- thuật toán tìm kiếm mù: không sử dụng thông tin ngoại trừ của không gian trạng thái
- tìm kiếm theo chiều rộng: mở rộng các nút theo thứ tự tạo
  - tìm kiếm không gian trạng thái theo từng lớp
  - có thể là tìm kiếm cây hoặc tìm kiếm đồ thị
  - độ phức tạp O(bd) với số phân nhánh b, chiều dài giải pháp tối thiểu d (nếu $b \geq 2$)
  - hoàn thành như tìm kiếm đồ thị; bán hoàn thành như tìm kiếm cây
  - tối ưu với chi phí hành động thống nhất



Bài tập

Thực hiện tìm lời giải theo phương pháp duyệt BFS cho bài toán 8-puzzle, thể hiện bằng:

- Tree search (vẽ lên giấy)

trạng thái xuất phát

2	6	5
	8	7
4	3	1

trạng thái đích

1	2	3
4	5	6
7	8	

Uninformed search algorithms

Breadth-first search

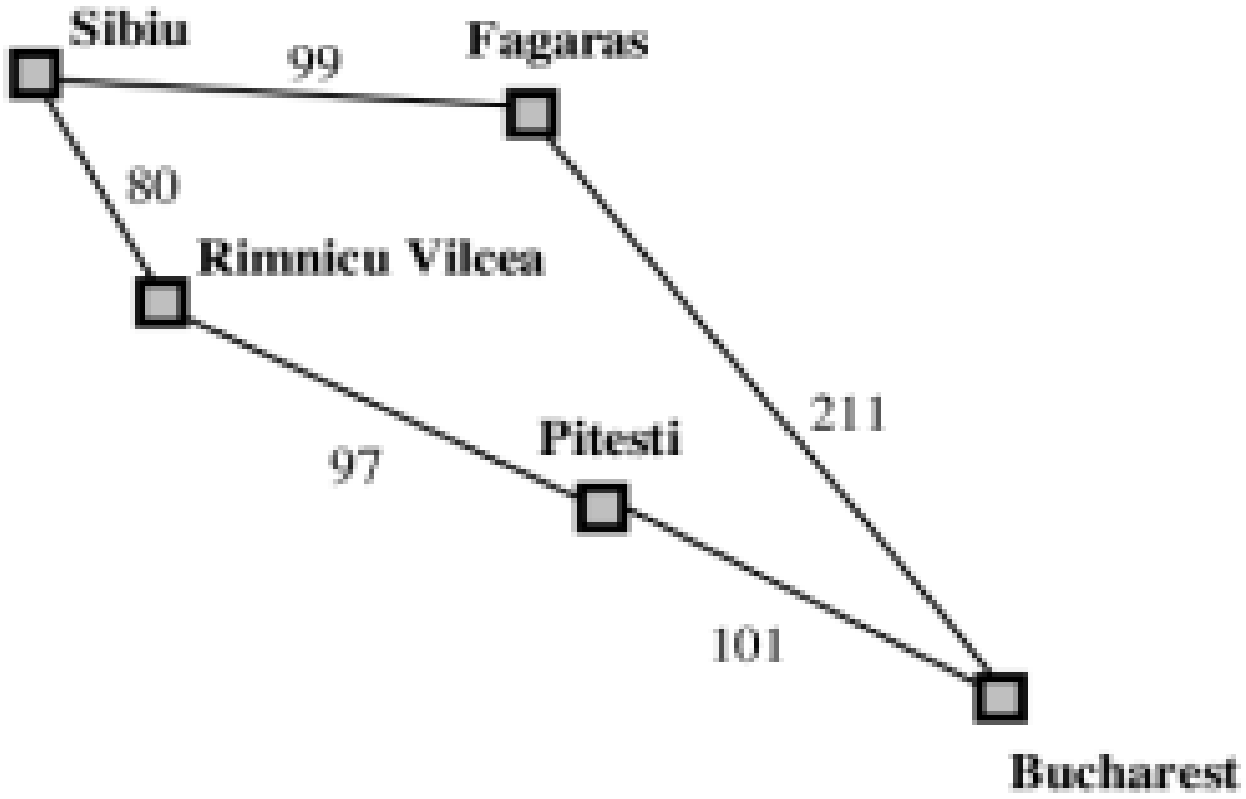
Uniform Cost Search

Depth-first search

Iterative Deepening

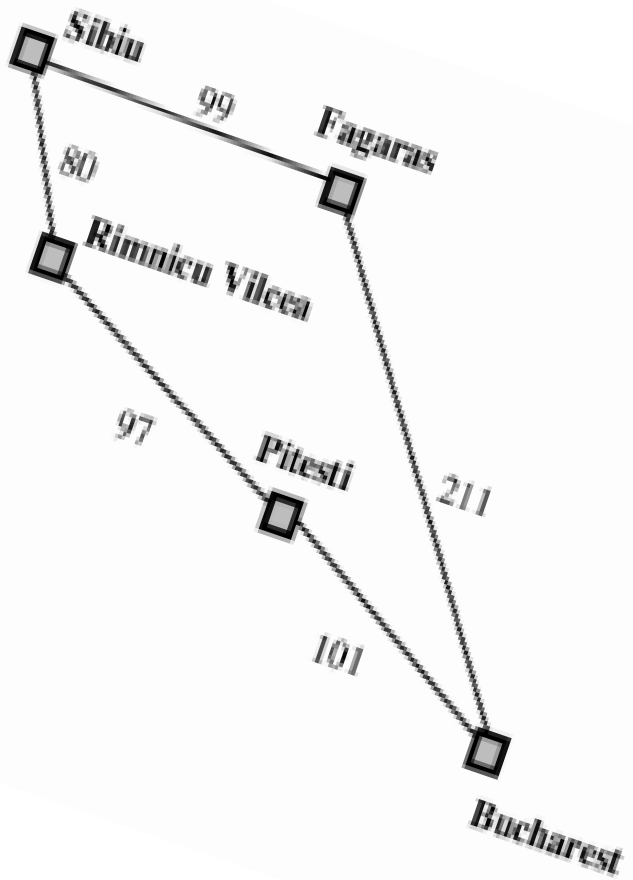
Uniform Cost Search

- tìm kiếm theo chiều rộng tối ưu nếu tất cả chi phí hành động bằng nhau
- nếu không thì không đảm bảo tối ưu
- ví dụ:



Uniform Cost Search

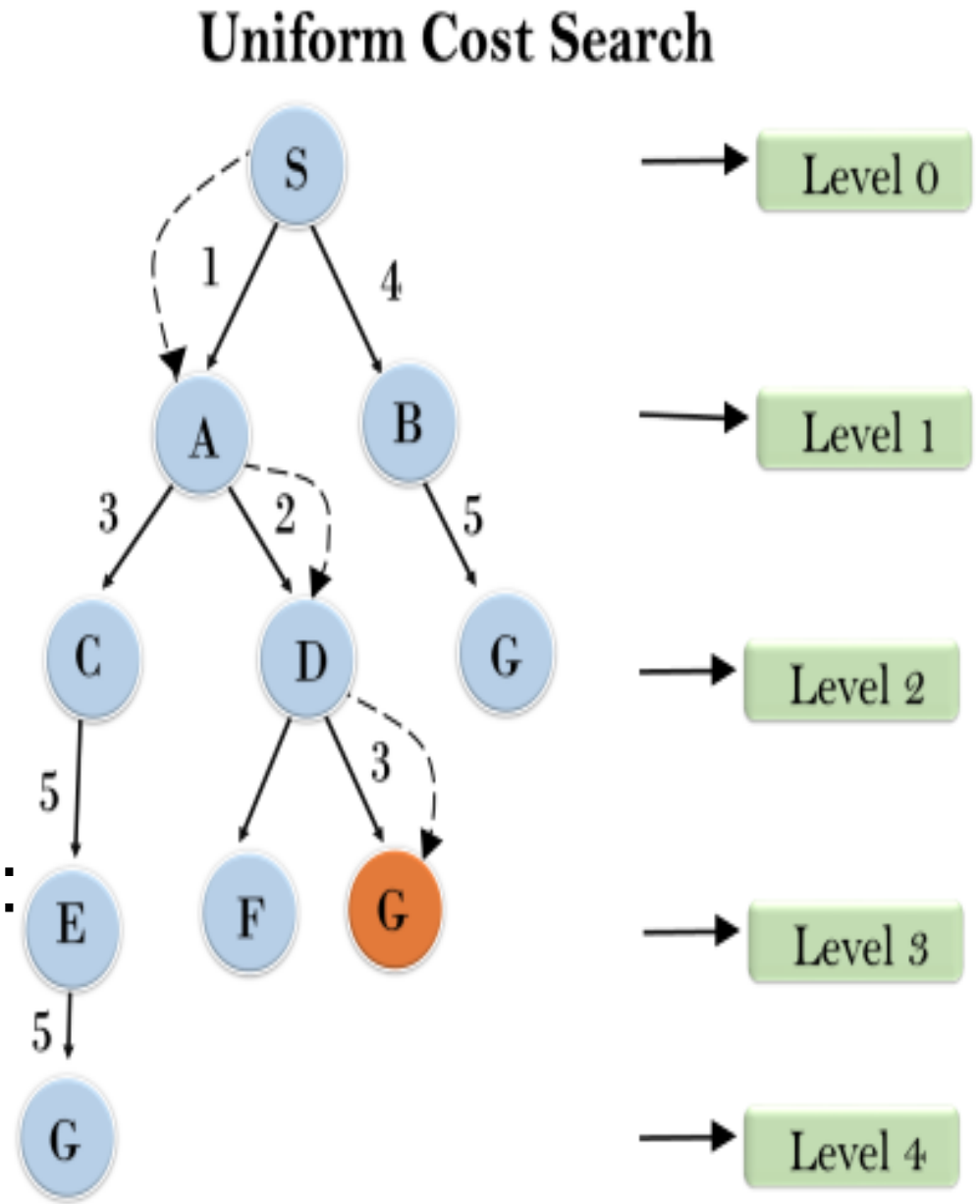
- tìm kiếm theo chiều rộng tối ưu nếu tất cả chi phí hành động bằng nhau
- nếu không thì không đảm bảo tối ưu
- ví dụ:



biện pháp khắc phục:
- uniform cost search luôn mở rộng một nút với chi phí đường dẫn tối thiểu (n.path_cost hay còn gọi là g(n))
- triển khai: hàng đợi ưu tiên (min-heap) cho danh sách mở

Uniform Cost Search

```
open := new MinHeap ordered by g
open.insert(make_root_node())
closed := new HashSet
while not open.is empty():
        n := open.pop min()
        if n.state ∉ closed:
                closed.insert(n.state)
                if is goal(n.state):
                        return extract_path(n)
                for each <a,s'> ∈ succ(n.state):
                        n' := make node(n, a,s')
                        open.insert(n')
return unsolvable
```



Uniform Cost Search

- **Thuận lợi:**
  - Tối ưu vì ở mọi trạng thái, đường dẫn có chi phí thấp nhất sẽ được chọn.
    - hiệu quả khi trọng số cạnh nhỏ vì nó khám phá các đường dẫn theo thứ tự đảm bảo tìm thấy đường dẫn ngắn nhất sớm.
  - Đây là phương pháp tìm kiếm cơ bản, không quá phức tạp, dễ sử dụng đối với nhiều người dùng.
  - Thuật toán hoàn chỉnh, đảm bảo có thể tìm ra giải pháp bất cứ khi nào có giải pháp khả thi.

Uniform Cost Search

- **Nhược điểm:**
  - không quan tâm đến số bước liên quan đến tìm kiếm và chỉ quan tâm đến chi phí đường dẫn.
    - có thể bị kẹt trong vòng lặp vô hạn.
  - Tìm kiếm này giữ nguyên danh sách các nút mà nó đã khám phá trong hàng đợi ưu tiên.
    - nếu bạn có một đồ thị lớn???

Đánh giá Uniform Cost Search

- **Tính đầy đủ:**
  - Tìm kiếm theo chi phí thống nhất hoàn tất, chẳng hạn như nếu có giải pháp, UCS sẽ tìm ra giải pháp đó.

- **Độ phức tạp về thời gian:**
  - Giả sử C* **là Chi phí của giải pháp tối ưu** và **ε** là mỗi bước để tiến gần hơn đến nút đích. Khi đó, số bước là = C*/ε+1. Ở đây chúng ta lấy +1, vì chúng ta bắt đầu từ trạng thái 0 và kết thúc ở C*/ε.
  - Do đó, độ phức tạp thời gian trường hợp xấu nhất của tìm kiếm chi phí thống nhất là $\mathbf{O(b^{1 + [C^*/\varepsilon]})}$.

Uniform Cost Search

- **Độ phức tạp về không gian gian:**
  - Logic tương tự cũng đúng với độ phức tạp của không gian, do đó độ phức tạp của không gian trường hợp xấu nhất của tìm kiếm chi phí thống nhất là $\mathbf{O(b^{1 + [C^*/\varepsilon]})}$.

- **Tối ưu:**
  - Tìm kiếm theo chi phí thống nhất luôn là tối ưu vì nó chỉ chọn đường dẫn có chi phí thấp nhất.

Uninformed search algorithms

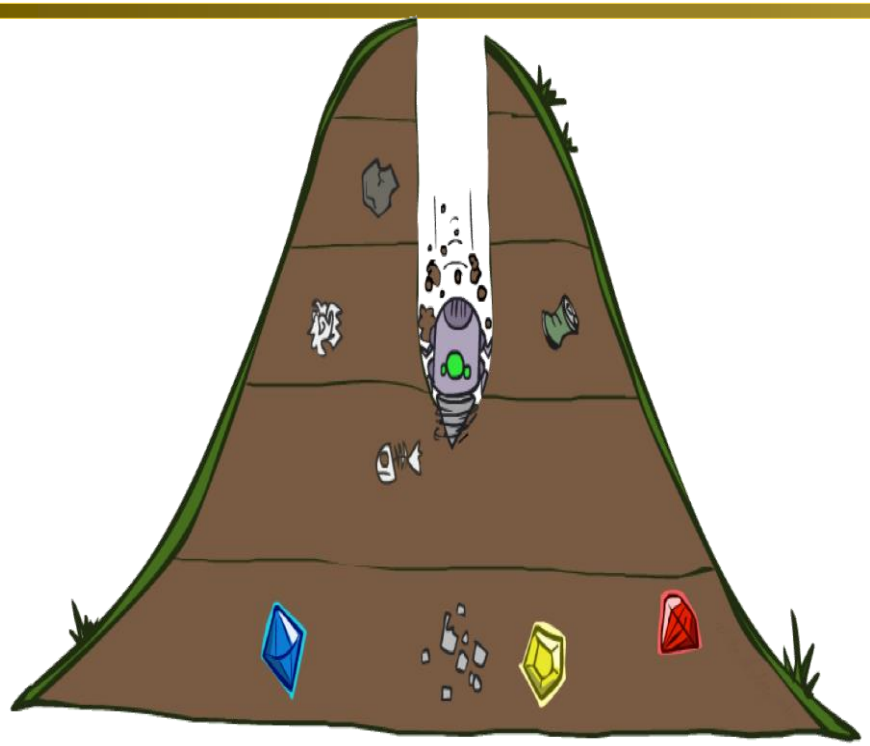
Breadth-first search

Uniform Cost Search

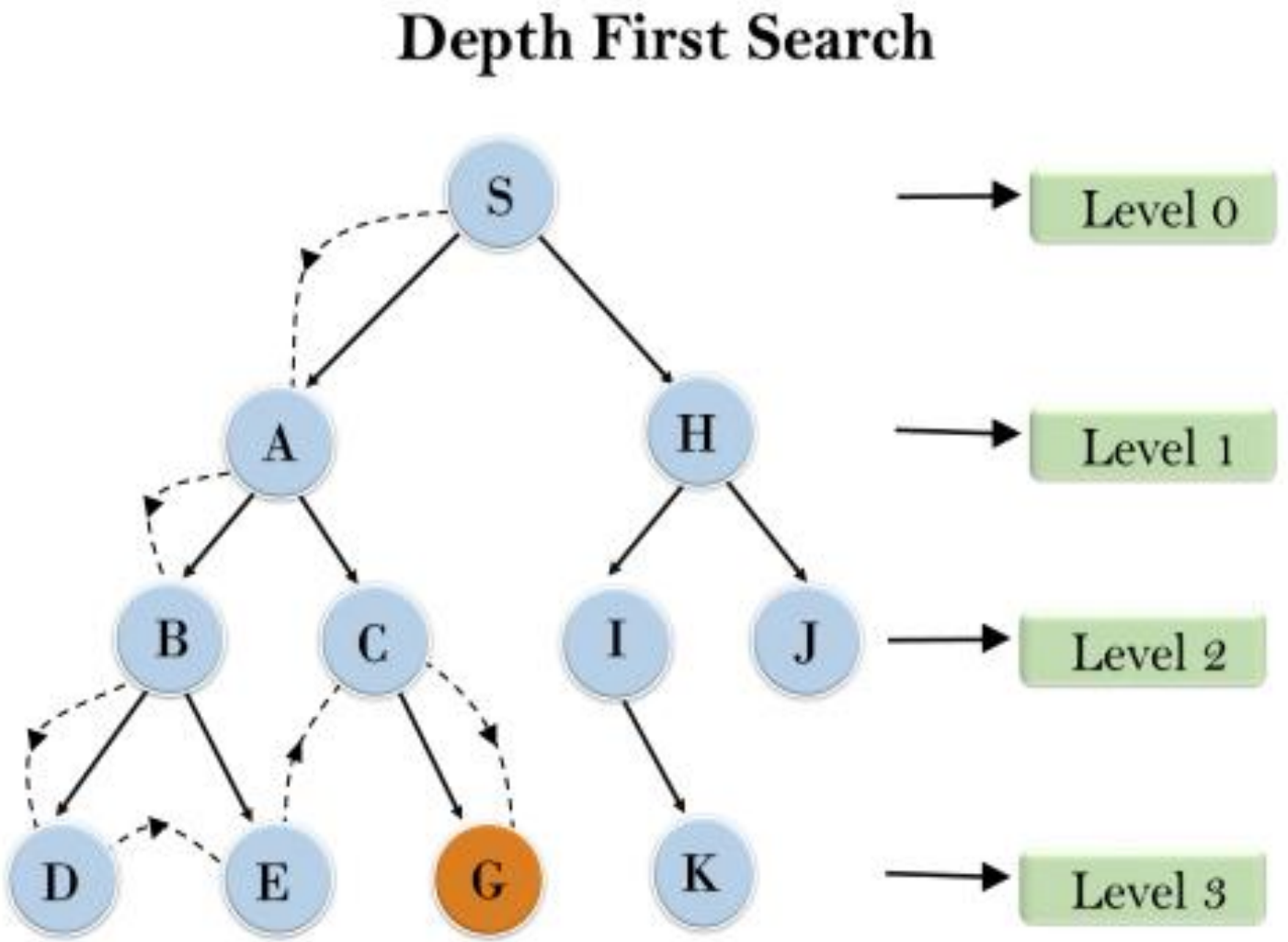
Depth-first search

Iterative Deepening

Depth-First Search



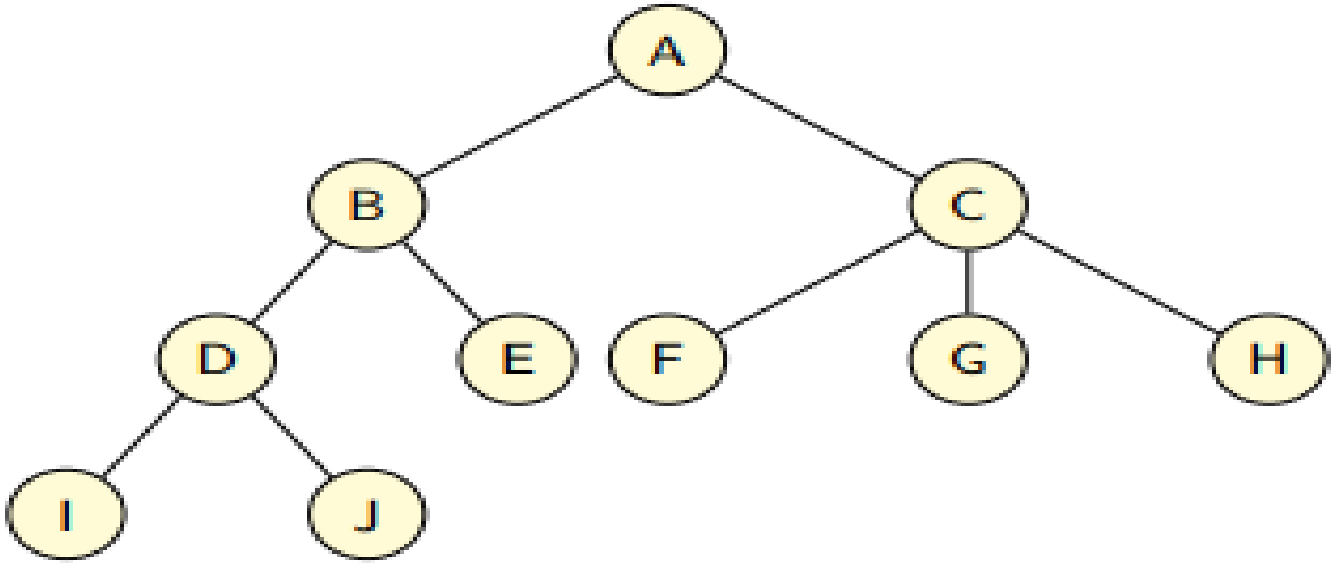
Tìm kiếm theo chiều sâu (DFS) mở rộng các nút theo thứ tự nút sâu nhất mở rộng trước (LIFO).
danh sách mở được triển khai dưới dạng ngăn xếp



Nút gốc ---> Nút trái ----> Nút phải.

Depth-First Search

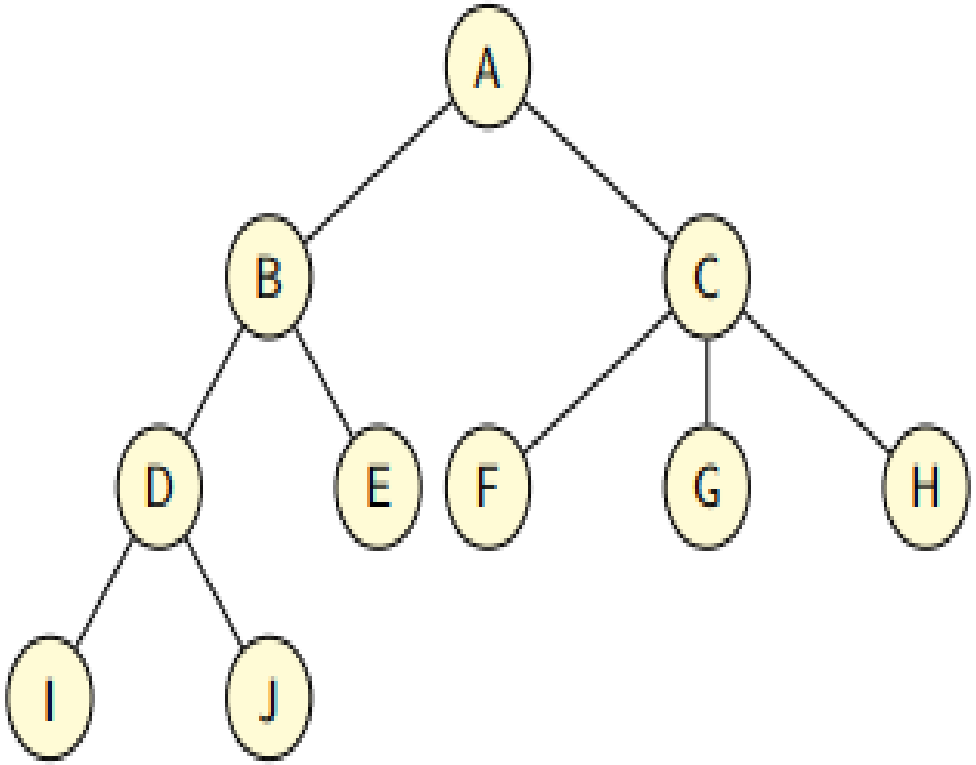
Tìm kiếm theo chiều sâu (DFS) mở rộng các nút theo thứ tự nút sâu nhất mở rộng trước (LIFO).
danh sách mở được triển khai dưới dạng ngăn xếp



Depth-First Search

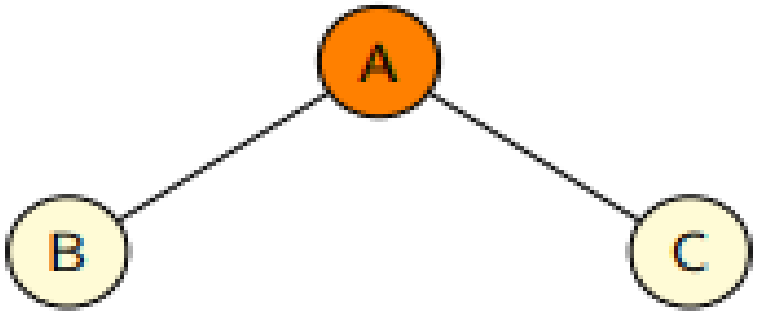
Tìm kiếm theo chiều sâu (DFS) mở rộng các nút theo thứ tự nút sâu nhất mở rộng trước (LIFO).
danh sách mở được triển khai dưới dạng ngăn xếp

open: A

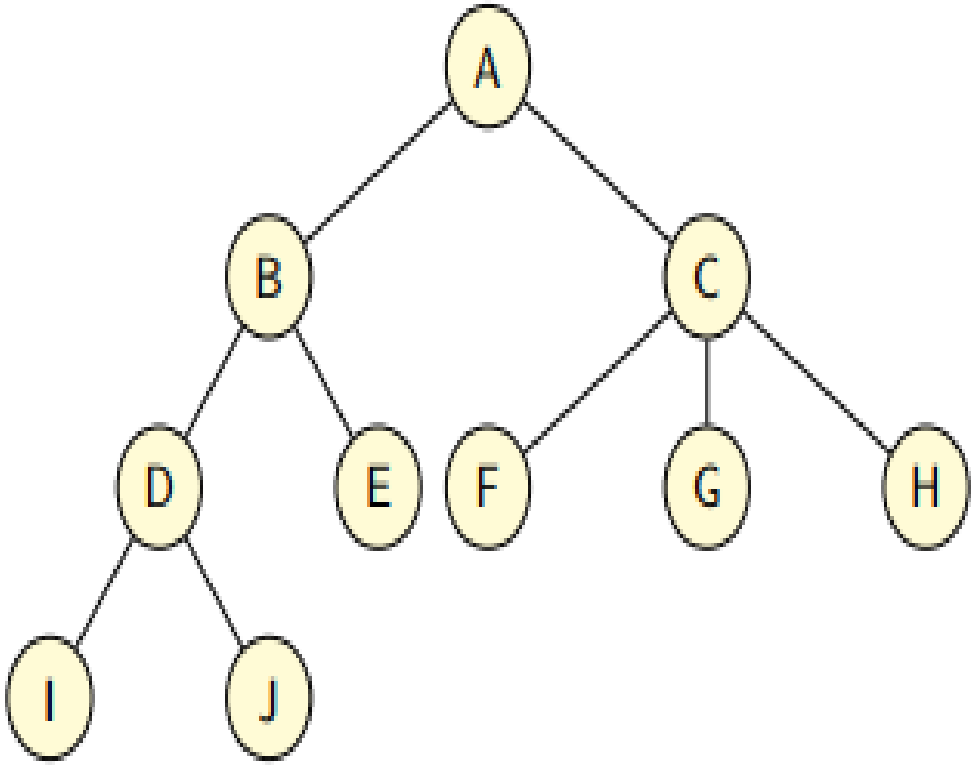


Depth-First Search

Tìm kiếm theo chiều sâu (DFS) mở rộng các nút theo thứ tự nút sâu nhất mở rộng trước (LIFO).
danh sách mở được triển khai dưới dạng ngăn xếp

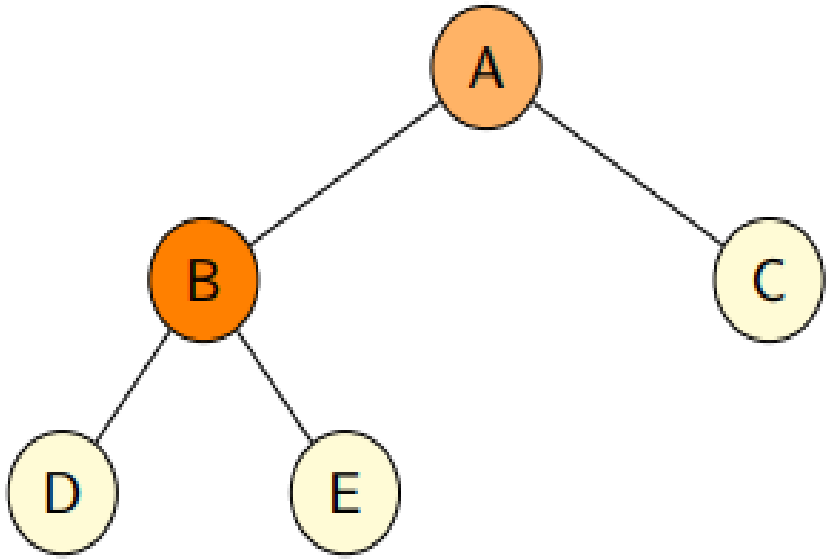


open: C, B

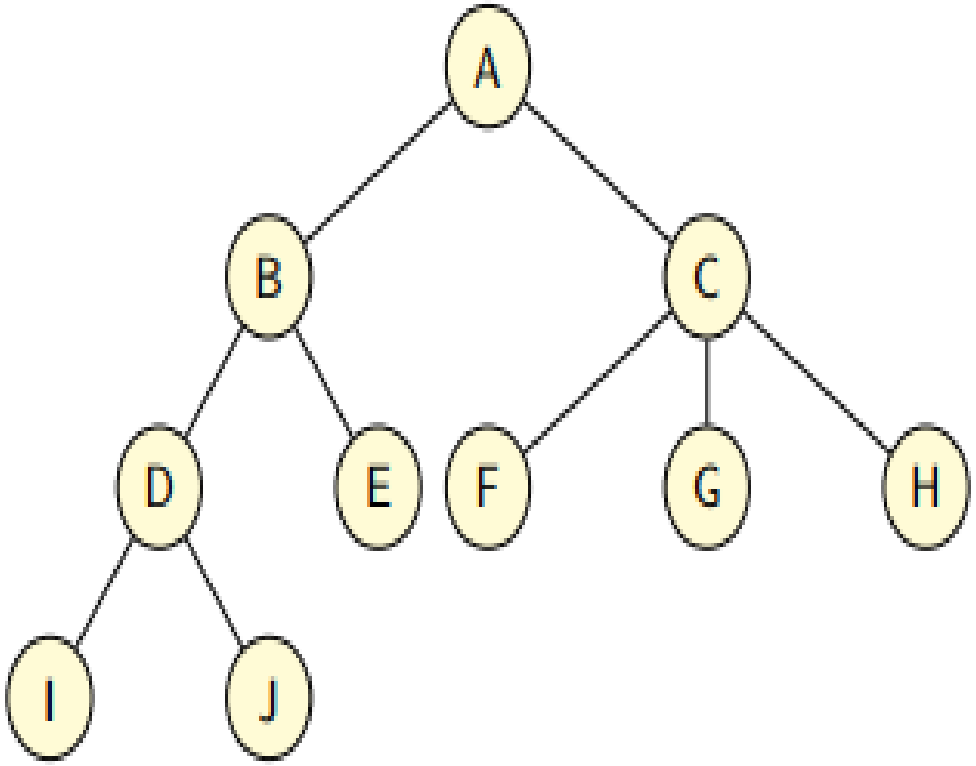


Depth-First Search

Tìm kiếm theo chiều sâu (DFS) mở rộng các nút theo thứ tự nút sâu nhất mở rộng trước (LIFO).
danh sách mở được triển khai dưới dạng ngăn xếp

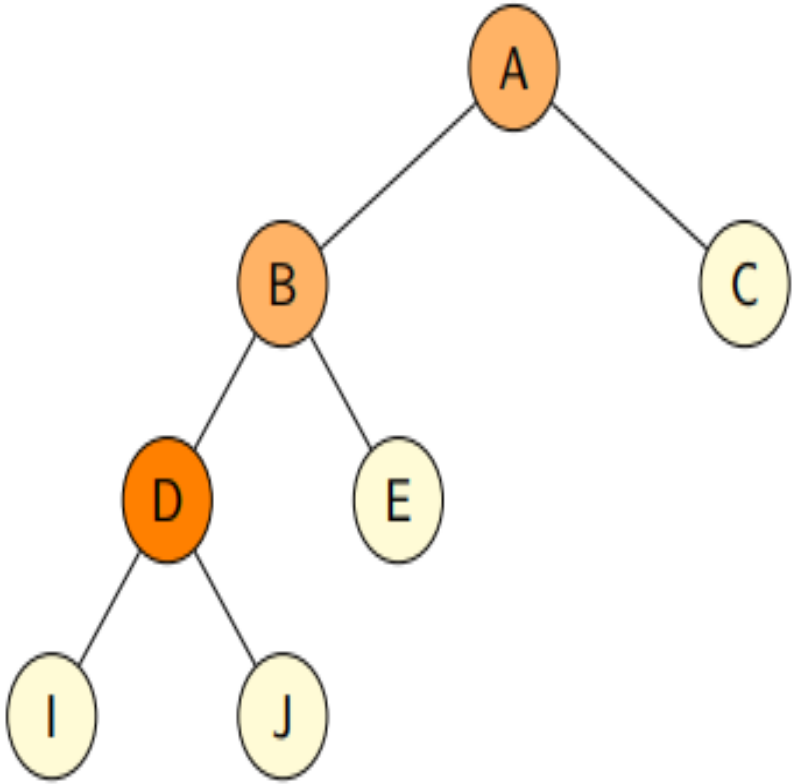


open: C, E, D

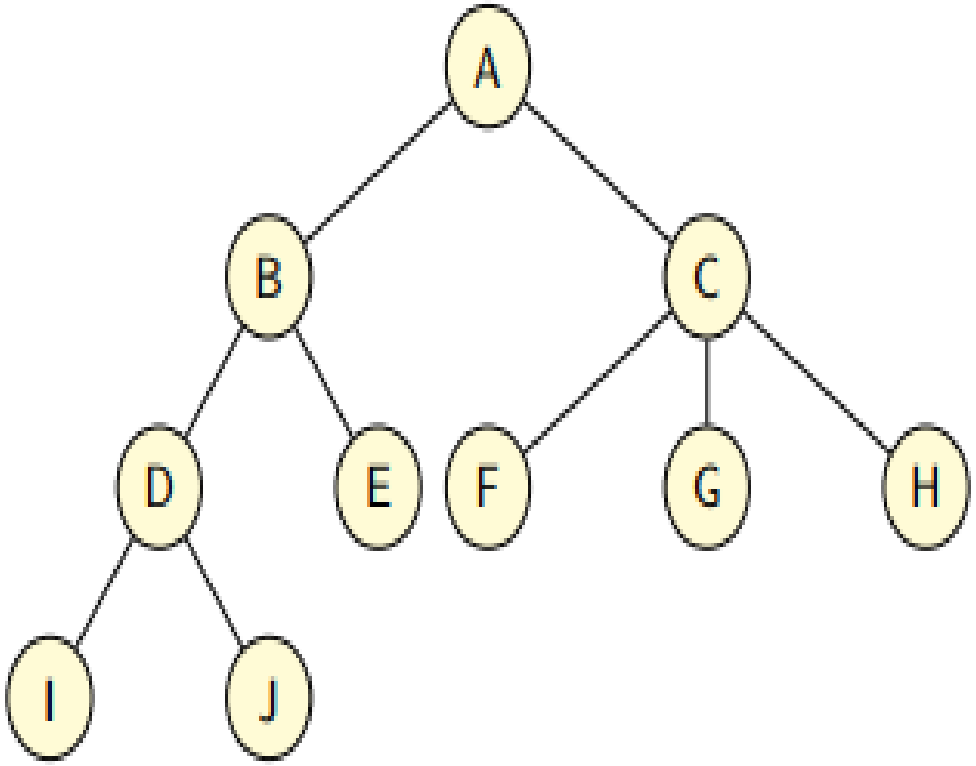


Depth-First Search

Tìm kiếm theo chiều sâu (DFS) mở rộng các nút theo thứ tự nút sâu nhất mở rộng trước (LIFO).
danh sách mở được triển khai dưới dạng ngăn xếp

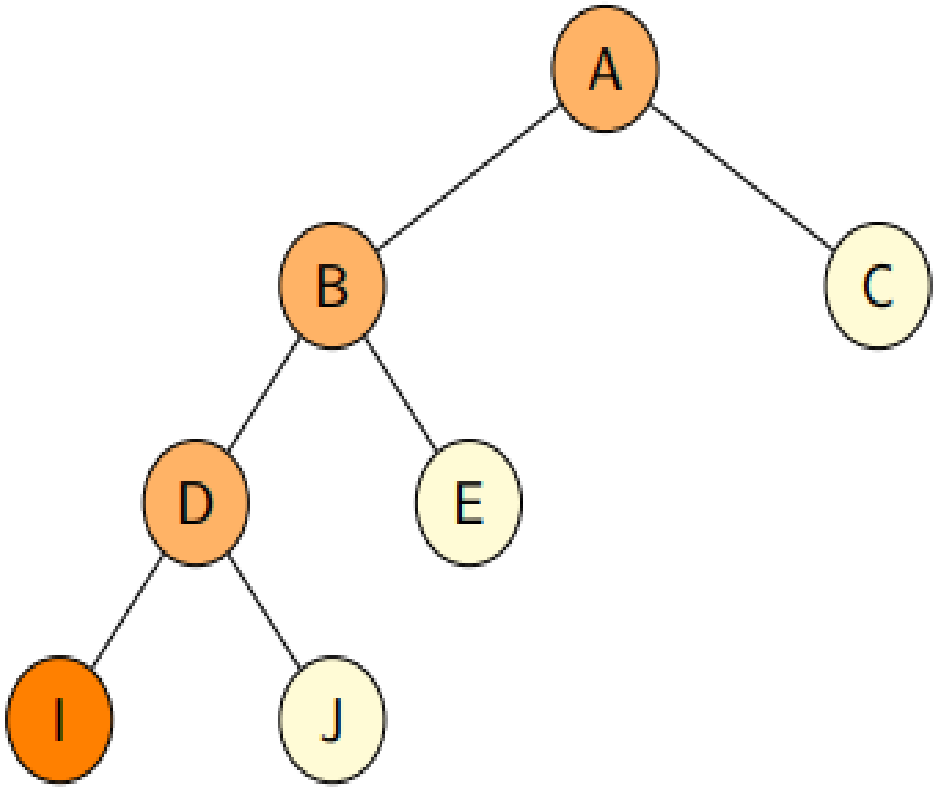


open: C, E, J, I

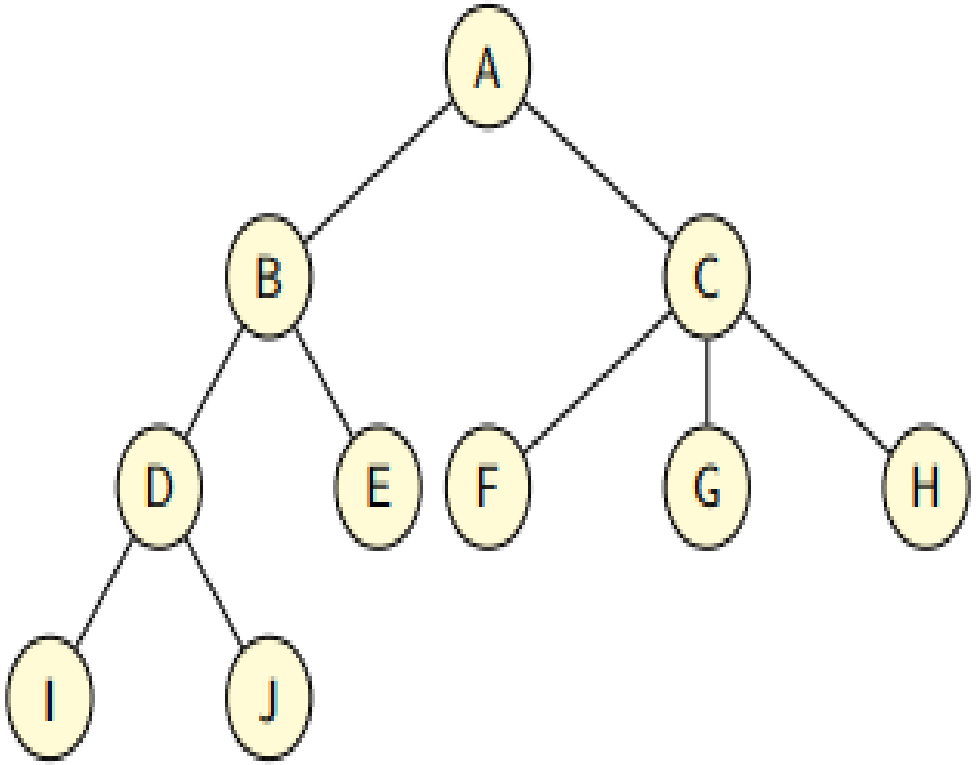


Depth-First Search

Tìm kiếm theo chiều sâu (DFS) mở rộng các nút theo thứ tự nút sâu nhất mở rộng trước (LIFO).
danh sách mở được triển khai dưới dạng ngăn xếp

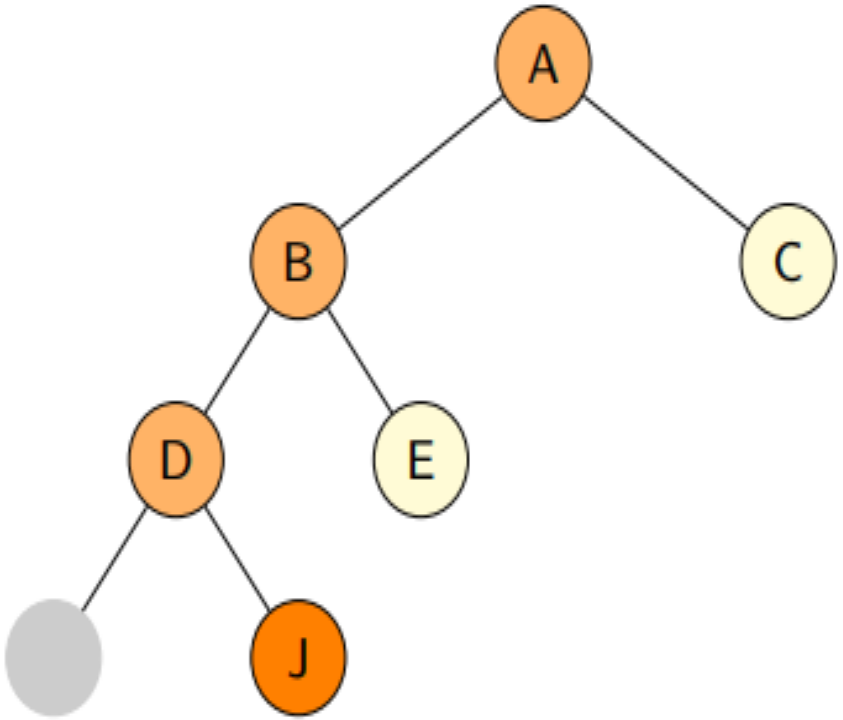


open: C, E, J

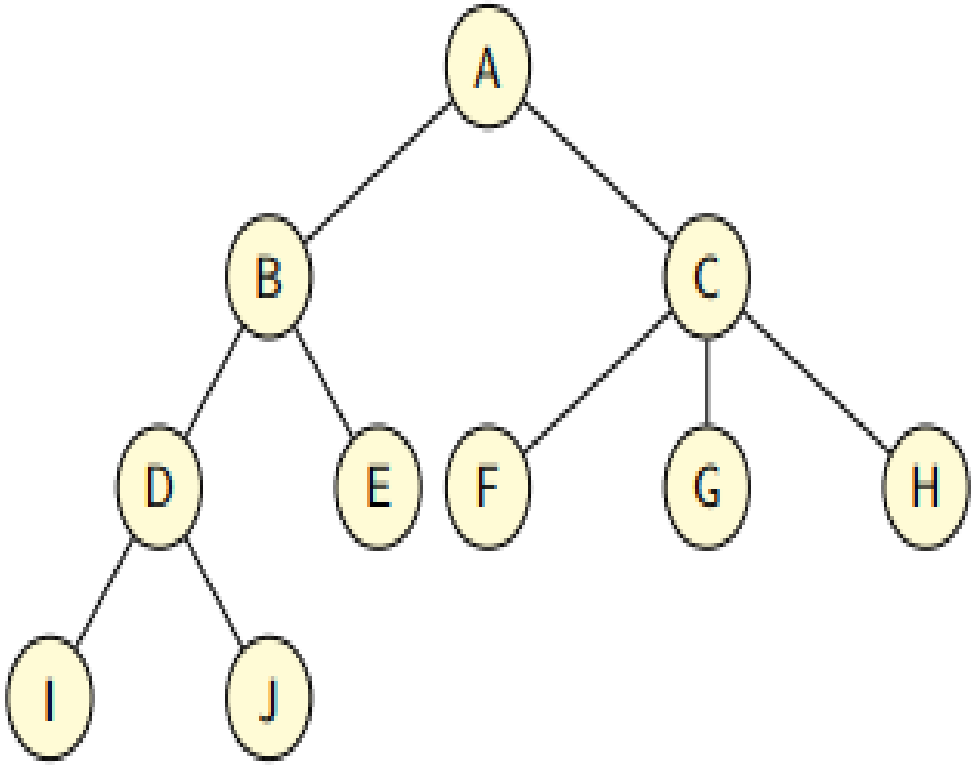


Depth-First Search

Tìm kiếm theo chiều sâu (DFS) mở rộng các nút theo thứ tự nút sâu nhất mở rộng trước (LIFO).
danh sách mở được triển khai dưới dạng ngăn xếp

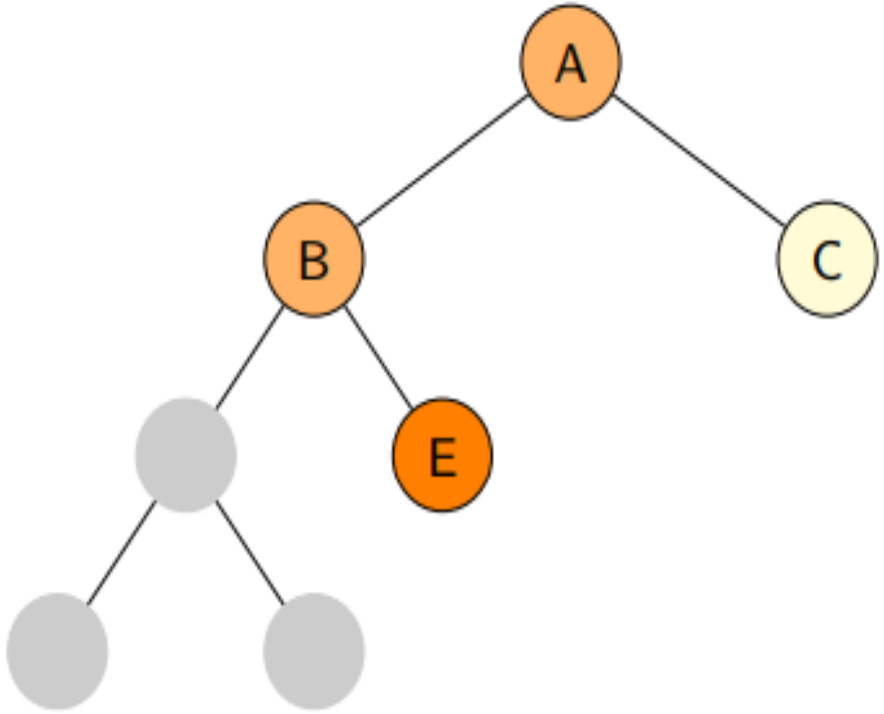


open: C, E

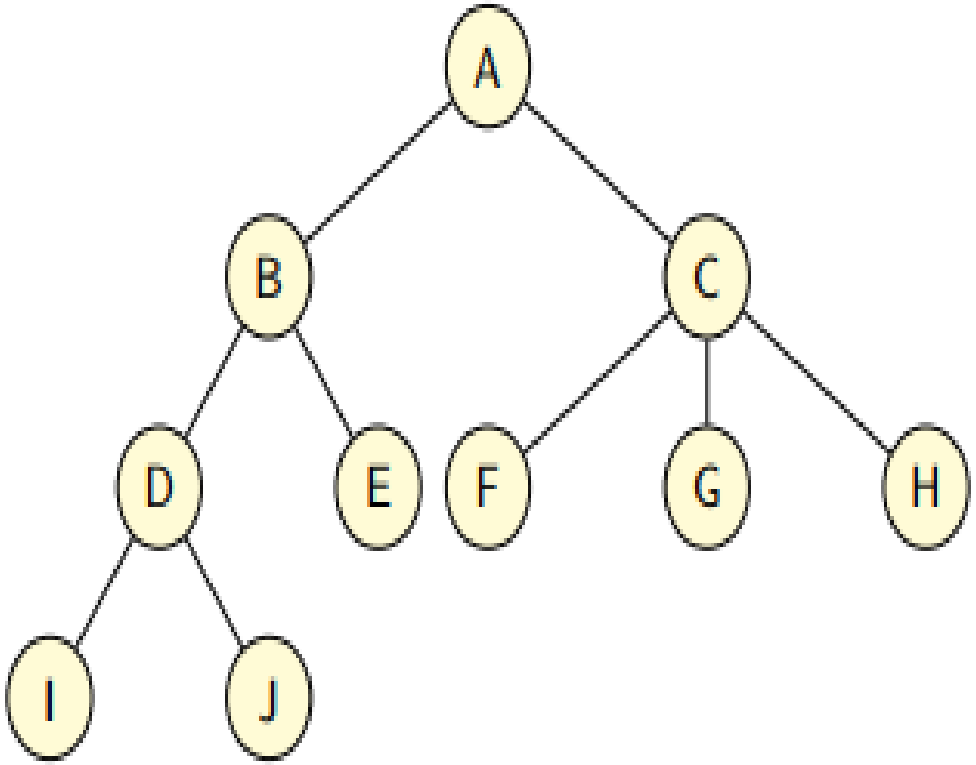


Depth-First Search

Tìm kiếm theo chiều sâu (DFS) mở rộng các nút theo thứ tự nút sâu nhất mở rộng trước (LIFO).
danh sách mở được triển khai dưới dạng ngăn xếp

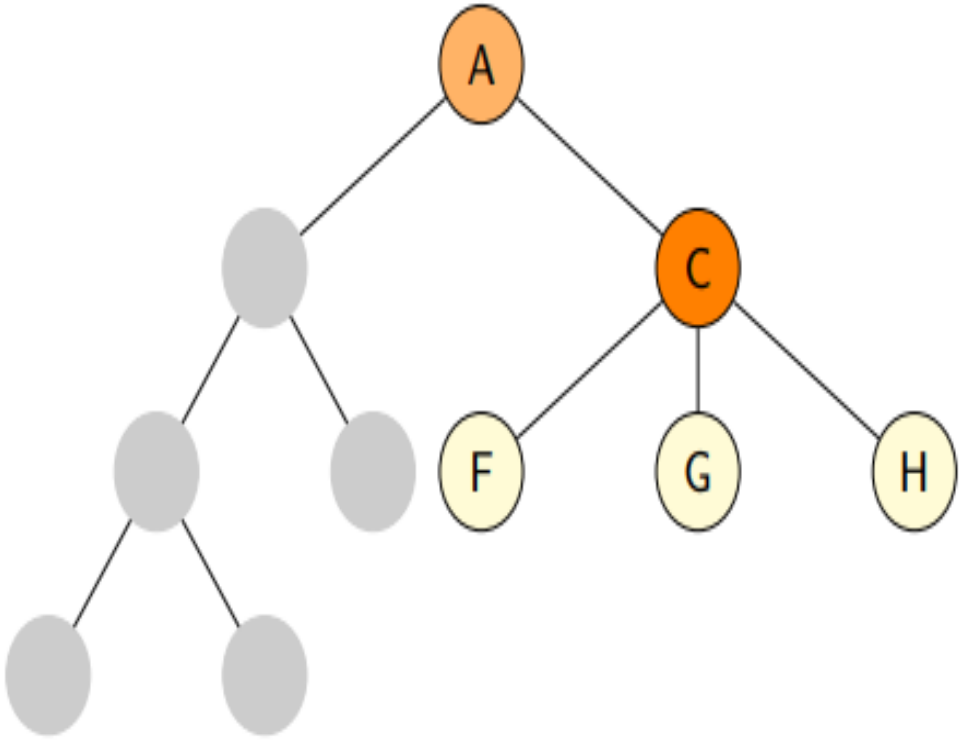


open: C

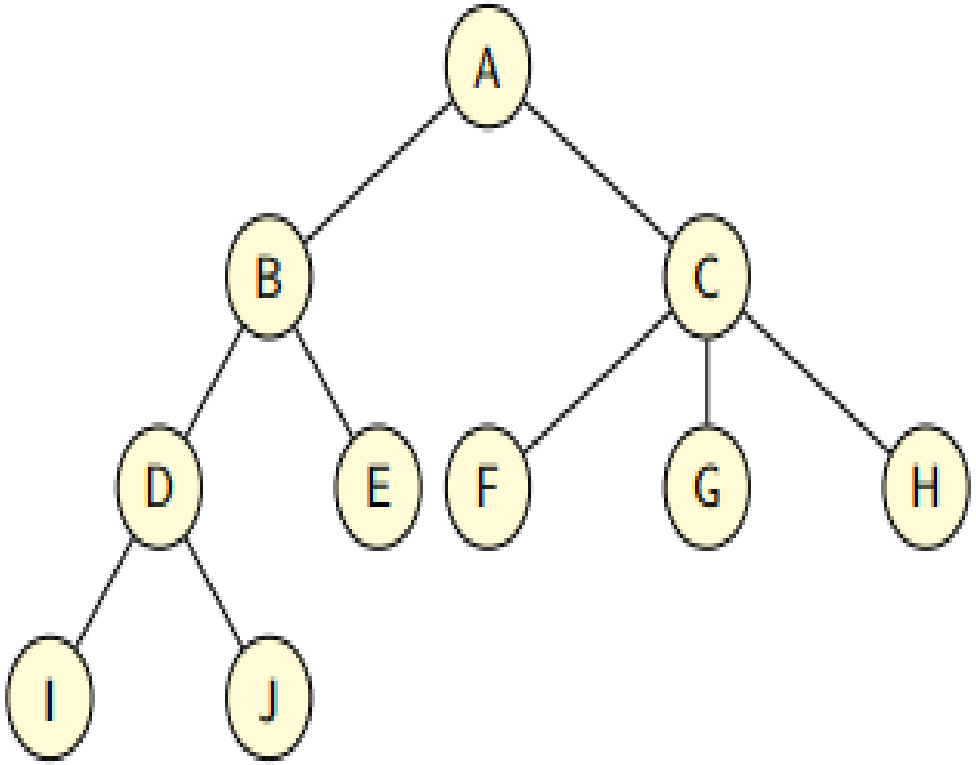


Depth-First Search

Tìm kiếm theo chiều sâu (DFS) mở rộng các nút theo thứ tự nút sâu nhất mở rộng trước (LIFO).
danh sách mở được triển khai dưới dạng ngăn xếp

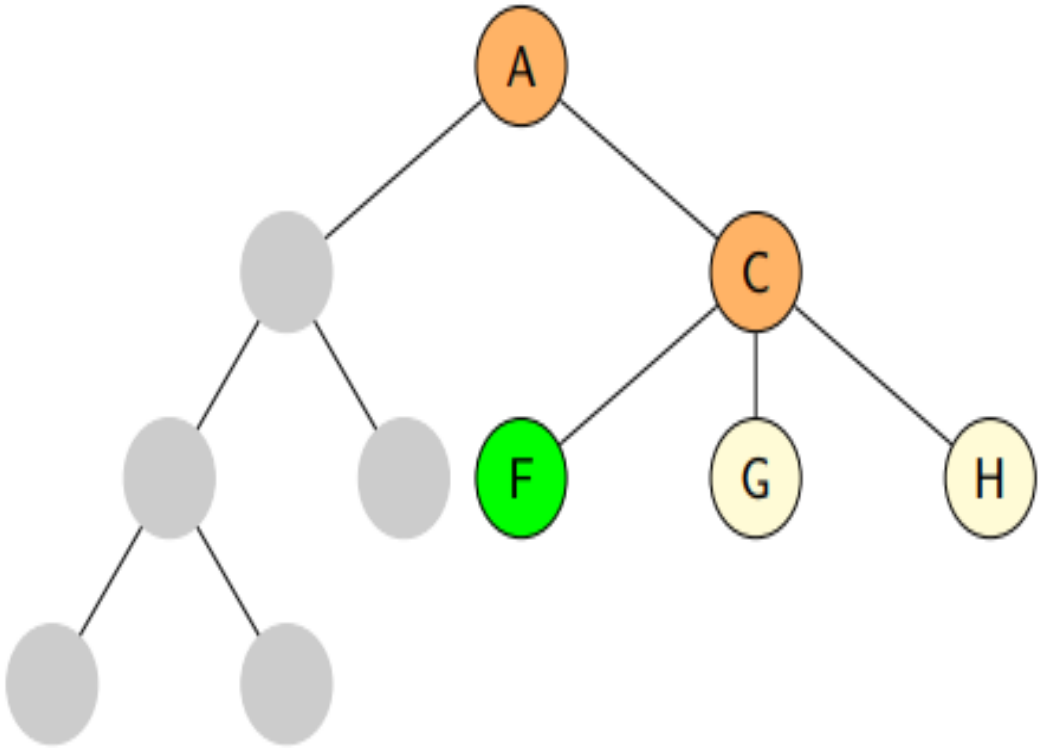


open: H, G, F

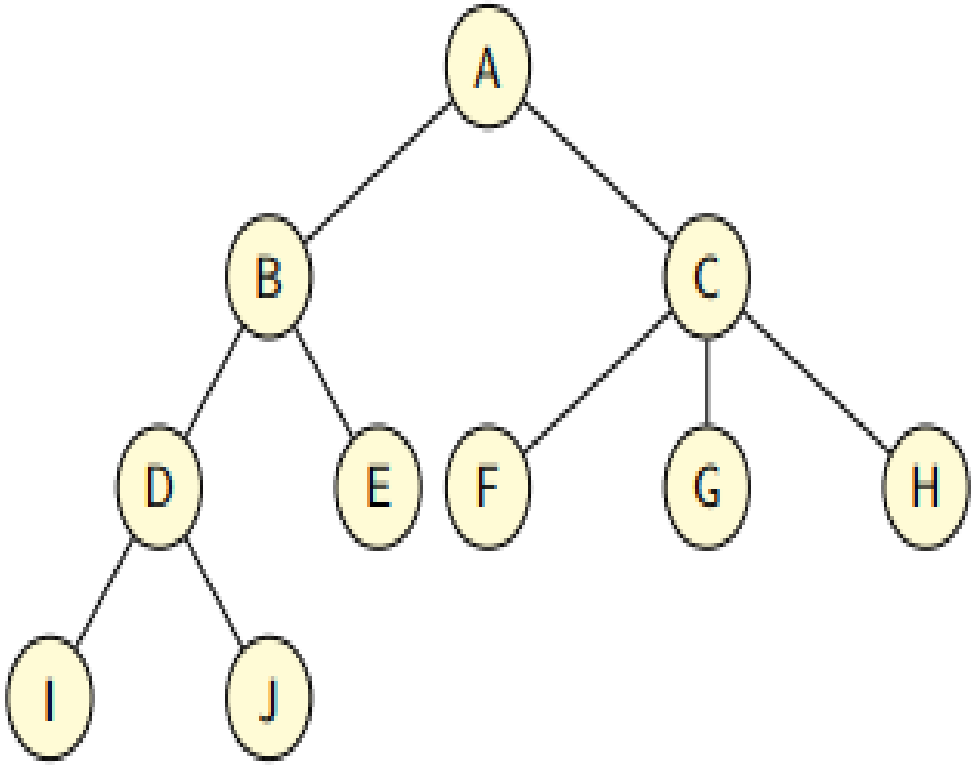


Depth-First Search

Tìm kiếm theo chiều sâu (DFS) mở rộng các nút theo thứ tự nút sâu nhất mở rộng trước (LIFO).
danh sách mở được triển khai dưới dạng ngăn xếp

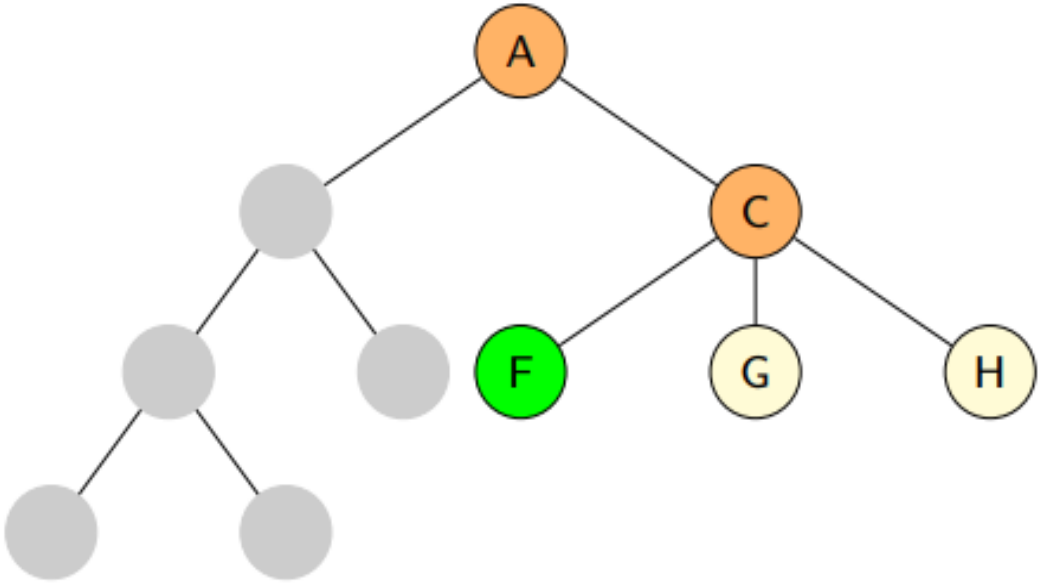


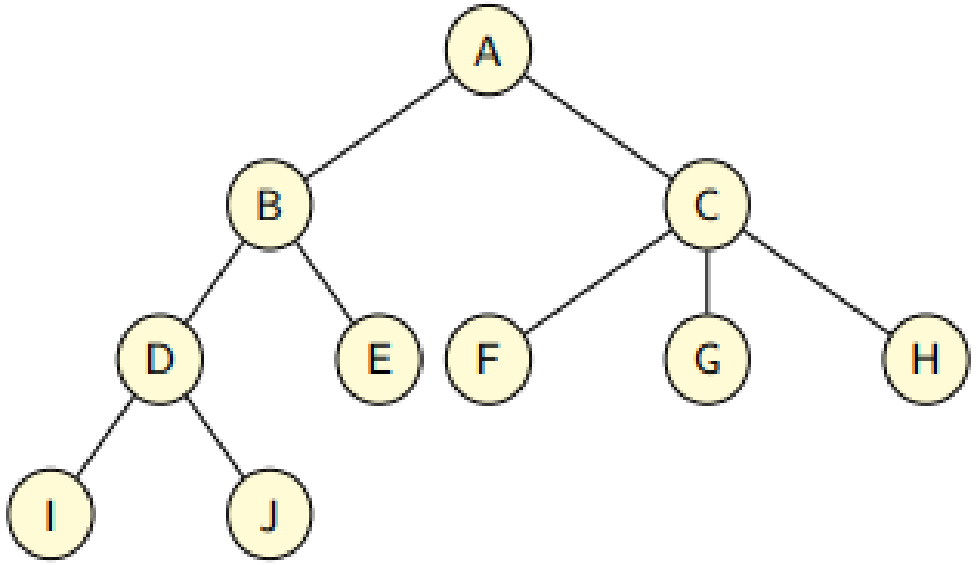
solution found!



Depth-First Search

Tìm kiếm theo chiều sâu (DFS) mở rộng các nút theo thứ tự nút sâu nhất mở rộng trước (LIFO).
danh sách mở được triển khai dưới dạng ngăn xếp





Depth-First Search: Thuộc tính

- **hầu như luôn được triển khai như một tìm kiếm cây (chúng ta sẽ xem tại sao)**
- **không hoàn chỉnh, không hoàn chỉnh một phần, không tối ưu (Tại sao?)**
- **hoàn chỉnh cho các không gian trạng thái phi chu trình,**
- **ví dụ, nếu không gian trạng thái** là **cây có hướng**

Depth-First Search (Non-Recursive Version)

```
open := new Stack
open.push_back(make_root_node())
while not open.is empty():
        n := open.pop_back()
        if is goal(n.state):
                return extract_path(n)
        for each <a,s'>∈ succ(n.state):
                n':= make_node(n, a,s')
                open.push_back(n')
return unsolvable
```

Depth-First Search (Recursive Version)

```
function depth_first_search(s)
if is goal(s):
        return <>
for each <a,s'> ∈ succ(s):
        solution := depth_first_search(s')
        if solution ≠ none:
                solution.push_front(a)
                return solution
return none
```

Depth-first Search (Recursive Version)
return depth_first_search(init())

Đánh giá Depth-First Search

- **Tính hoàn chỉnh:**
  - Thuật toán tìm kiếm DFS hoàn chỉnh trong không gian trạng thái hữu hạn vì nó sẽ mở rộng mọi nút trong một cây tìm kiếm giới hạn.
- **Độ phức tạp thời gian:**
  - Độ phức tạp thời gian của DFS sẽ tương đương với nút mà thuật toán đi qua. Nó được đưa ra bởi: $\mathbf{T(n)= 1+ n^2 + n^3 +.....+ n^m =O(n^m)}$
- **Độ phức tạp về không gian:**
  - DFS chỉ cần lưu trữ một đường dẫn duy nhất từ nút gốc, do đó độ phức tạp về không gian của DFS tương đương với kích thước của tập hợp rìa, là **O(bm)** .
- **Tối ưu:** Thuật toán tìm kiếm DFS không tối ưu vì nó có thể tạo ra số lượng lớn các bước hoặc chi phí cao để tiếp cận nút đích.

Bài tập

Thực hiện tìm lời giải theo phương pháp duyệt DFS cho bài toán 8-puzzle, thể hiện bằng:

- Tree search (vẽ lên giấy)

trạng thái xuất phát

2	6	5
	8	7
4	3	1

trạng thái đích

1	2	3
4	5	6
7	8	

Bài tập

Thực hiện tìm lời giải theo phương pháp duyệt DFS cho bài toán 8-puzzle, thể hiện bằng:

- Tree search (vẽ lên giấy) chụp đưa vào word=> PDF
- Viết chương trình bằng python => py

trạng thái xuất phát

2	6	5
	8	7
4	3	1

trạng thái đích

1	2	3
4	5	6
7	8	

Uninformed search algorithms

Breadth-first search

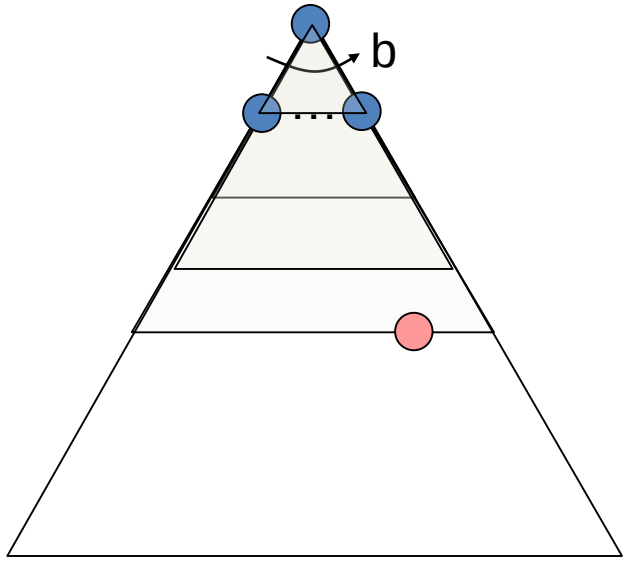
Uniform Cost Search

Depth-first search

Iterative Deepening

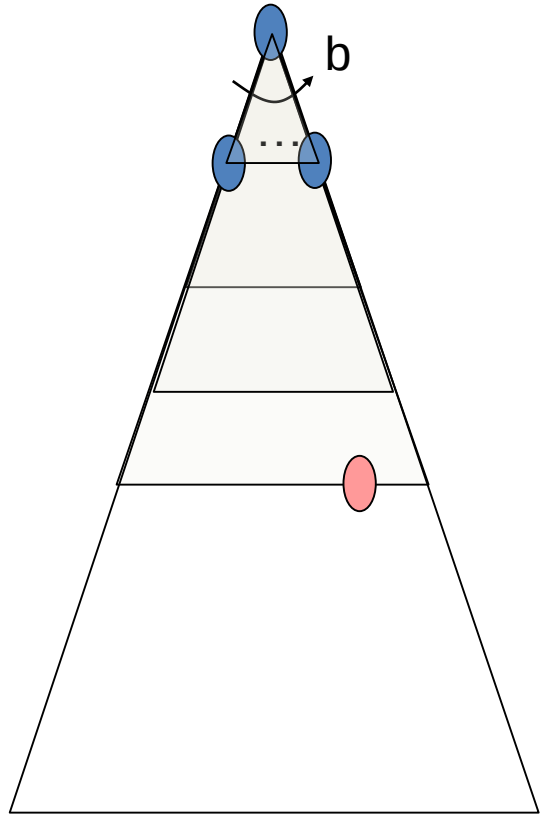
Iterative Deepening

- Là sự kết hợp của thuật toán DFS và BFS.
- Ý tưởng: tìm kiếm theo chiều sâu cho đến một "giới hạn độ sâu" nhất định và tiếp tục tăng giới hạn độ sâu sau mỗi lần lặp cho đến khi tìm thấy nút mục tiêu
- Kết hợp lợi ích của tính năng tìm kiếm nhanh của tìm kiếm theo chiều rộng và tính hiệu quả bộ nhớ của tìm kiếm theo chiều sâu



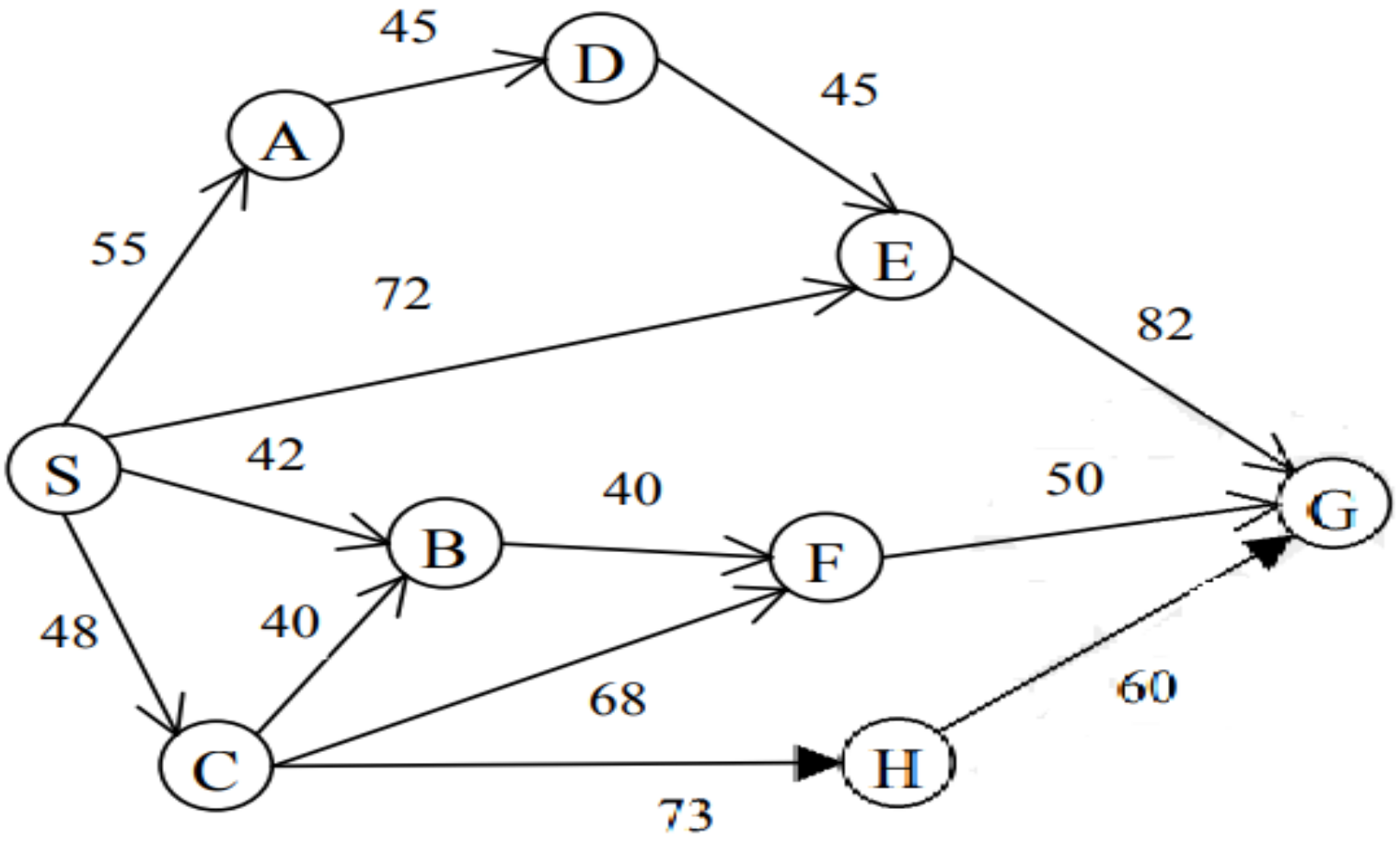
Iterative Deepening

- **Sau đây là các bước thực hiện thuật toán tìm kiếm theo chiều sâu lặp lại:**
  - Đặt giới hạn độ sâu thành 0.
  - Thực hiện DFS đến giới hạn độ sâu.
  - Nếu tìm thấy trạng thái mục tiêu, hãy trả về trạng thái đó.
  - Nếu không tìm thấy trạng thái mục tiêu và chưa đạt đến độ sâu tối đa, hãy tăng giới hạn độ sâu và lặp lại các bước 2-4.
  - Nếu không tìm thấy trạng thái mục tiêu và đã đạt đến độ sâu tối đa, hãy kết thúc tìm kiếm và trả về lỗi.



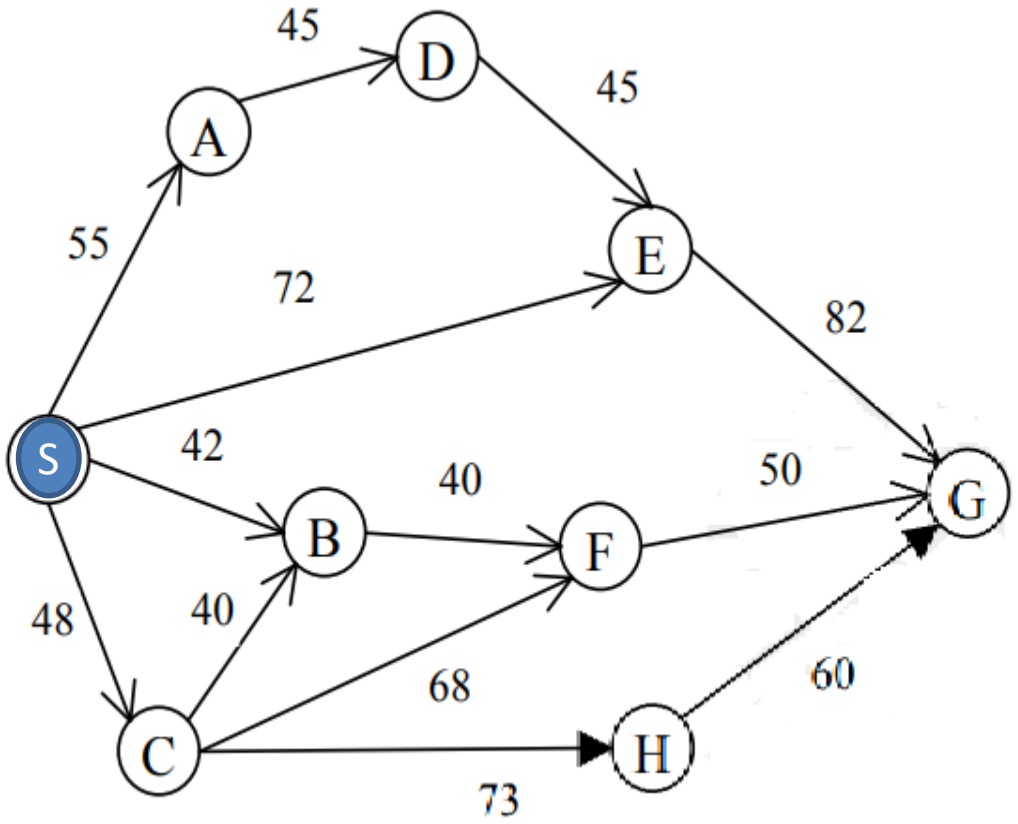
Iterative Deepening

Ví dụ:



Iterative Deepening

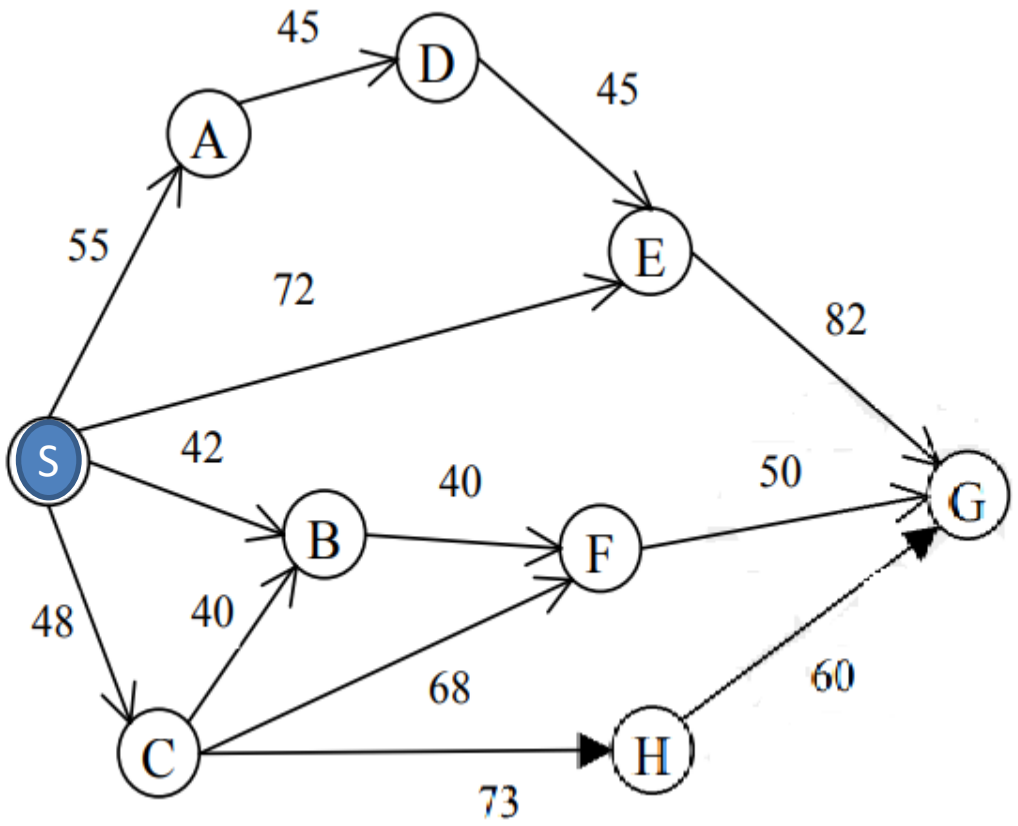
Ví dụ:



Nút được xét	Tập biên O (ngăn xếp LIFO trong trường hợp này)
C = 0	
	S
S	

Iterative Deepening

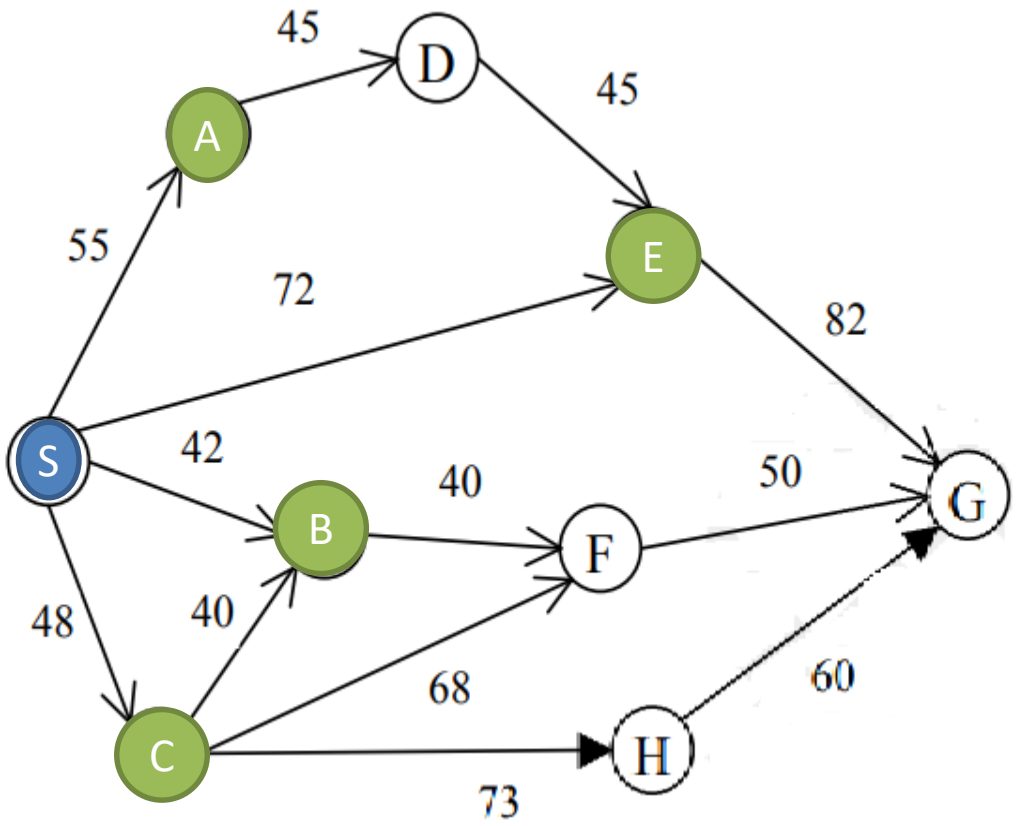
Ví dụ:



Nút được xét	Tập biên O (ngăn xếp LIFO trong trường hợp này)
C = 0	
	S
S	
C = 1	
	S
S	

Iterative Deepening

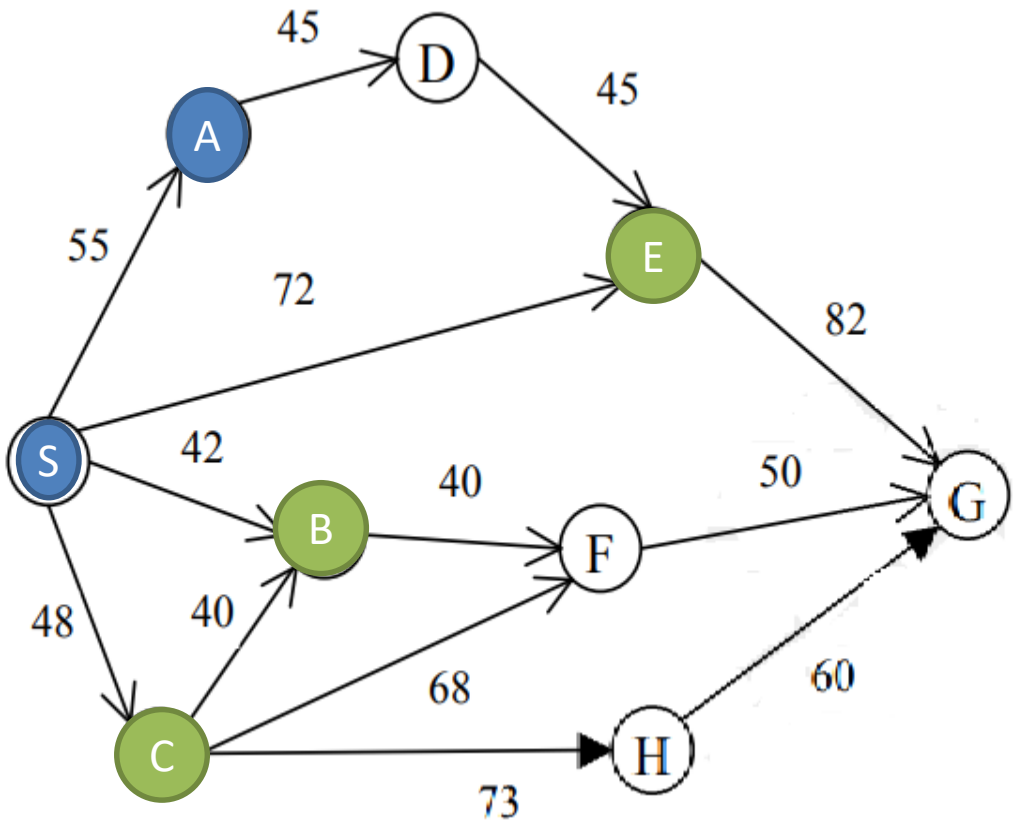
Ví dụ:



Nút được xét	Tập biên O (ngăn xếp LIFO trong trường hợp này)
C = 0	
	S
S	
C = 1	
	S
S	AS, ES, BS, CS

Iterative Deepening

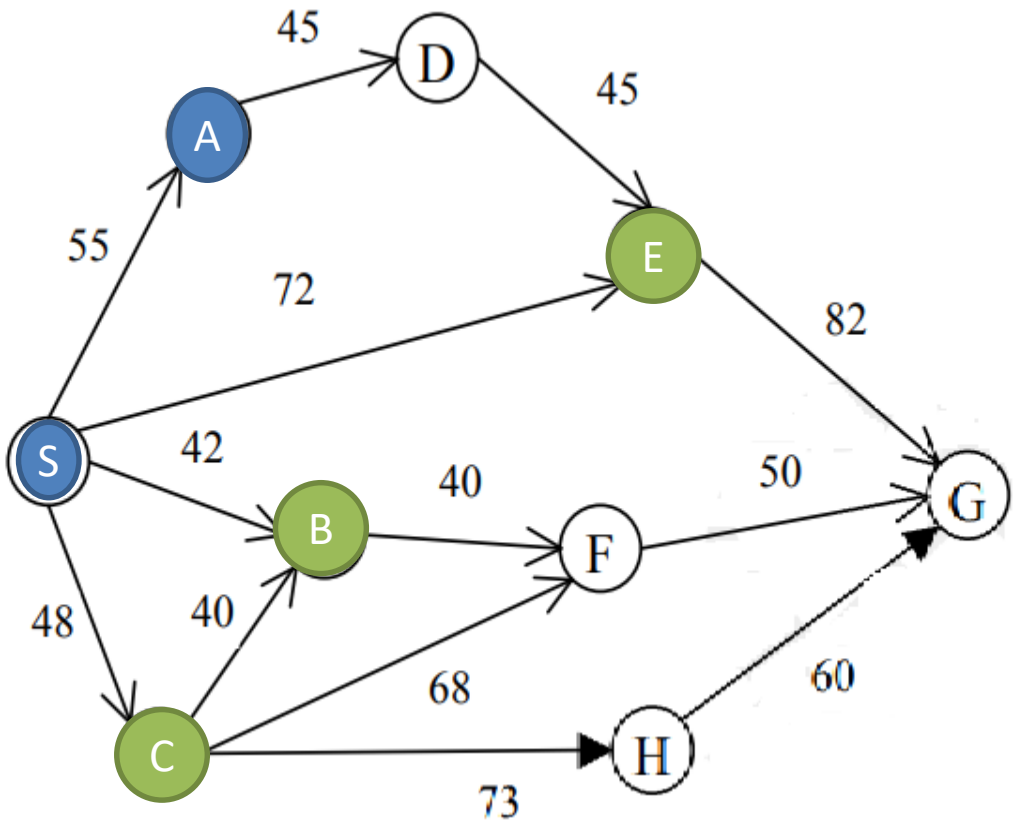
Ví dụ:



Nút được xét	Tập biên O (ngăn xếp LIFO trong trường hợp này)
C = 0	
	S
S	
C = 1	
	S
S	AS, ES, BS, CS
AS	ES, BS, CS

Iterative Deepening

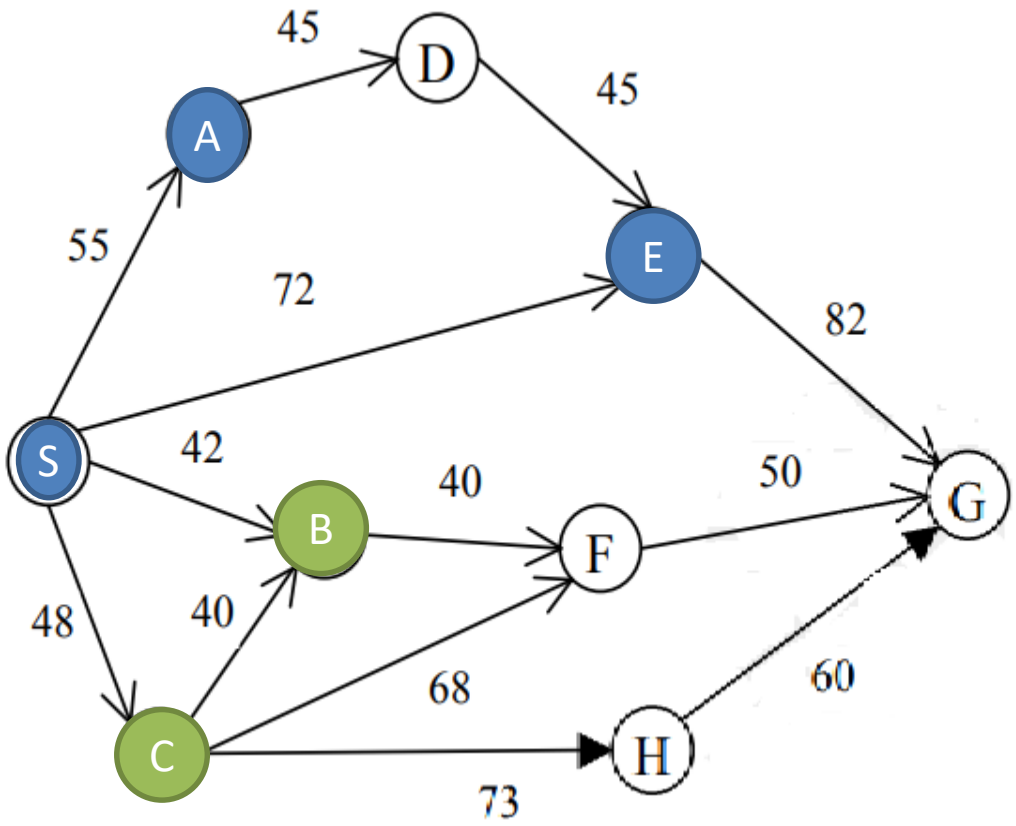
Ví dụ:



Nút được xét	Tập biên O (ngăn xếp LIFO trong trường hợp này)
C = 0	
	S
S	
C = 1	
	S
S	AS, ES, BS, CS
AS	ES, BS, CS

Iterative Deepening

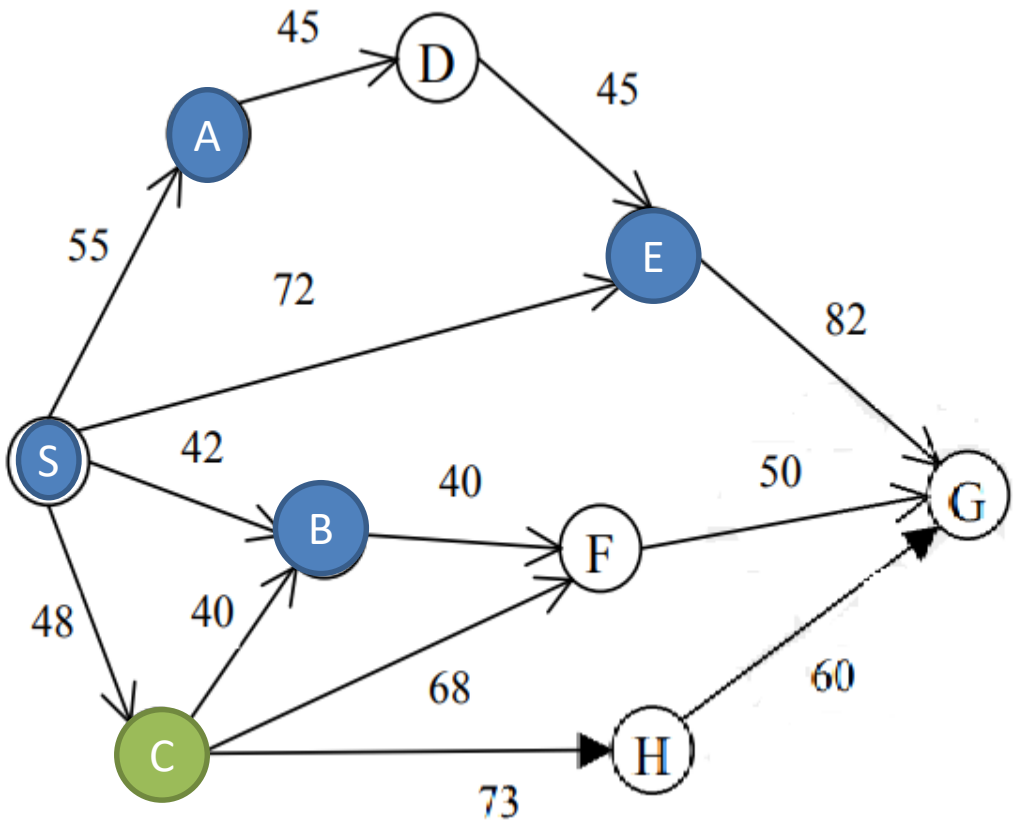
Ví dụ:



Nút được xét	Tập biên O (ngăn xếp LIFO trong trường hợp này)
C = 0	
	S
S	
C = 1	
	S
S	AS, ES, BS, CS
AS	ES, BS, CS
ES	BS, CS

Iterative Deepening

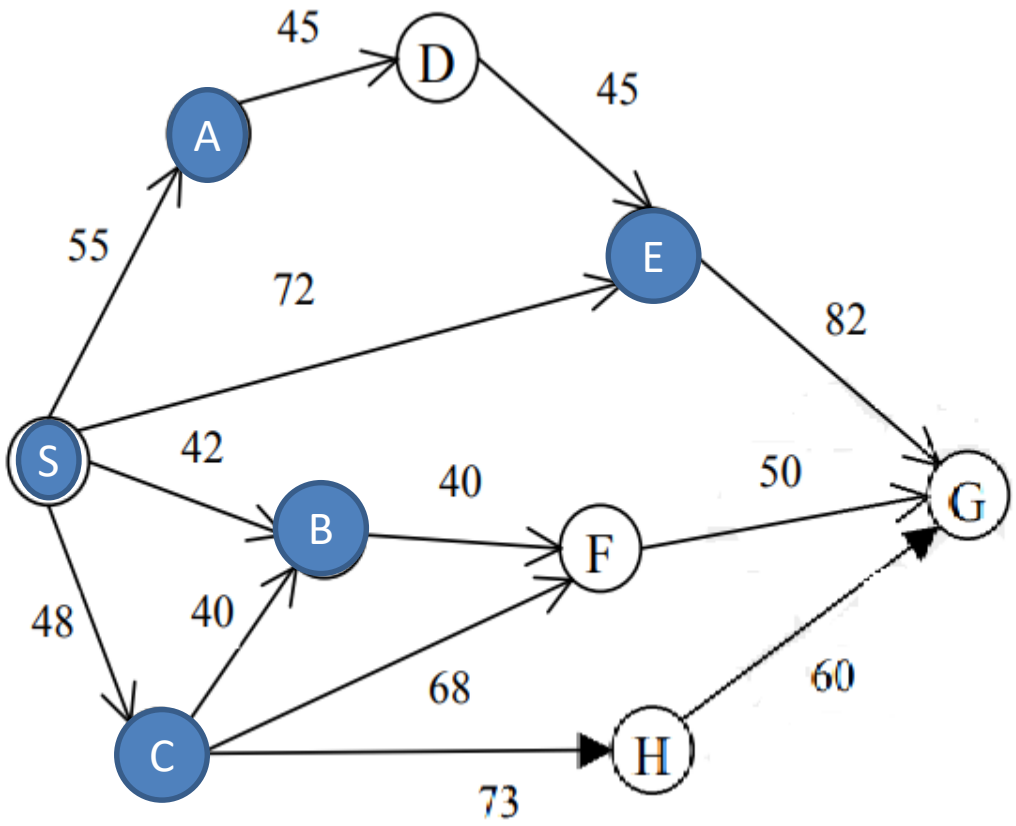
Ví dụ:



Nút được xét	Tập biên O (ngăn xếp LIFO trong trường hợp này)
C = 0	
	S
S	
C = 1	
	S
S	AS, ES, BS, CS
AS	ES, BS, CS
ES	BS, CS
BS	

Iterative Deepening

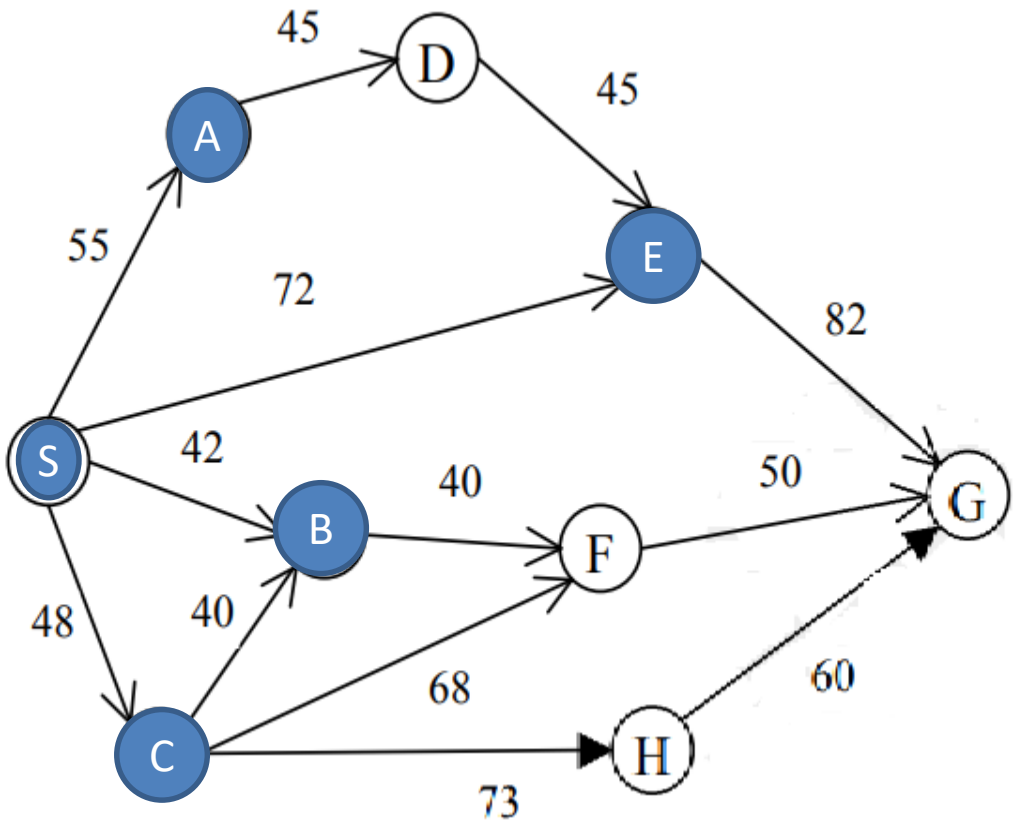
Ví dụ:



Nút được xét	Tập biên O (ngăn xếp LIFO trong trường hợp này)
C = 0	
	S
S	
C = 1	
	S
S	AS, ES, BS, CS
AS	ES, BS, CS
ES	BS, CS
BS	CS

Iterative Deepening

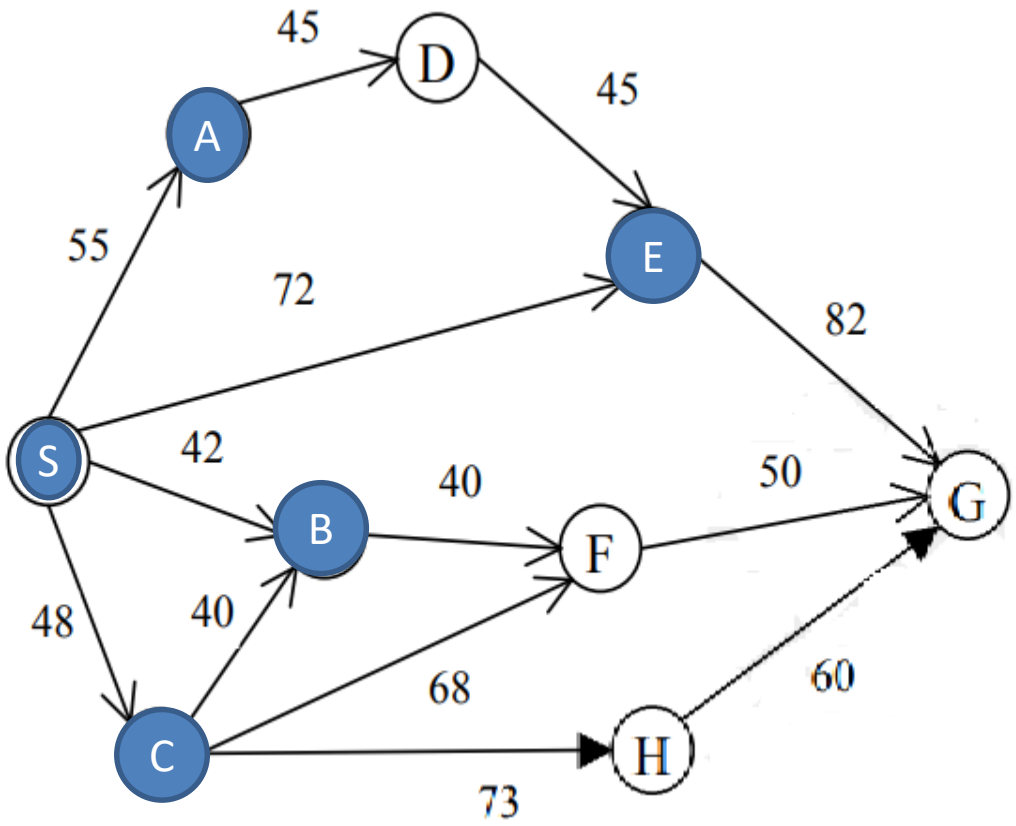
Ví dụ:



Nút được xét	Tập biên O (ngăn xếp LIFO trong trường hợp này)
C = 0	
	S
S	
C = 1	
	S
S	AS, ES, BS, CS
AS	ES, BS, CS
ES	BS, CS
BS	CS
CS	

Iterative Deepening

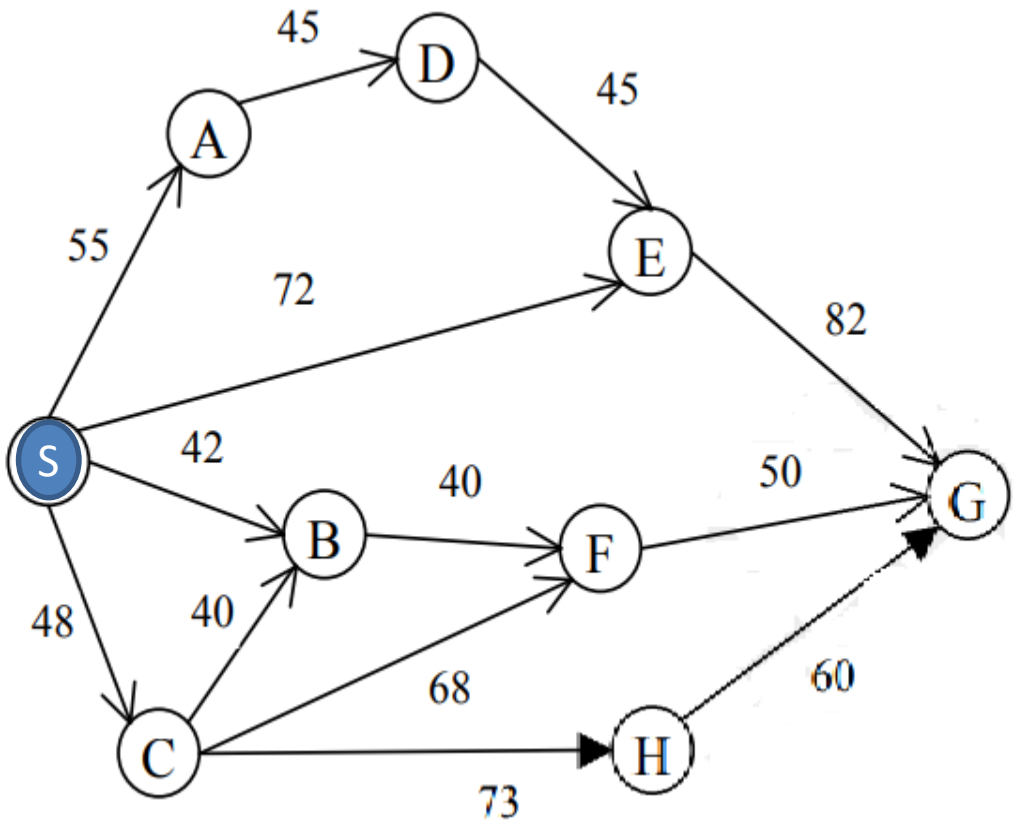
Ví dụ:



Nút được xét	Tập biên O (ngăn xếp LIFO trong trường hợp này)
C = 0	
	S
S	
C = 1	
	S
S	AS, ES, BS, CS
AS	ES, BS, CS
ES	BS, CS
BS	CS
CS	

Iterative Deepening

Ví dụ:

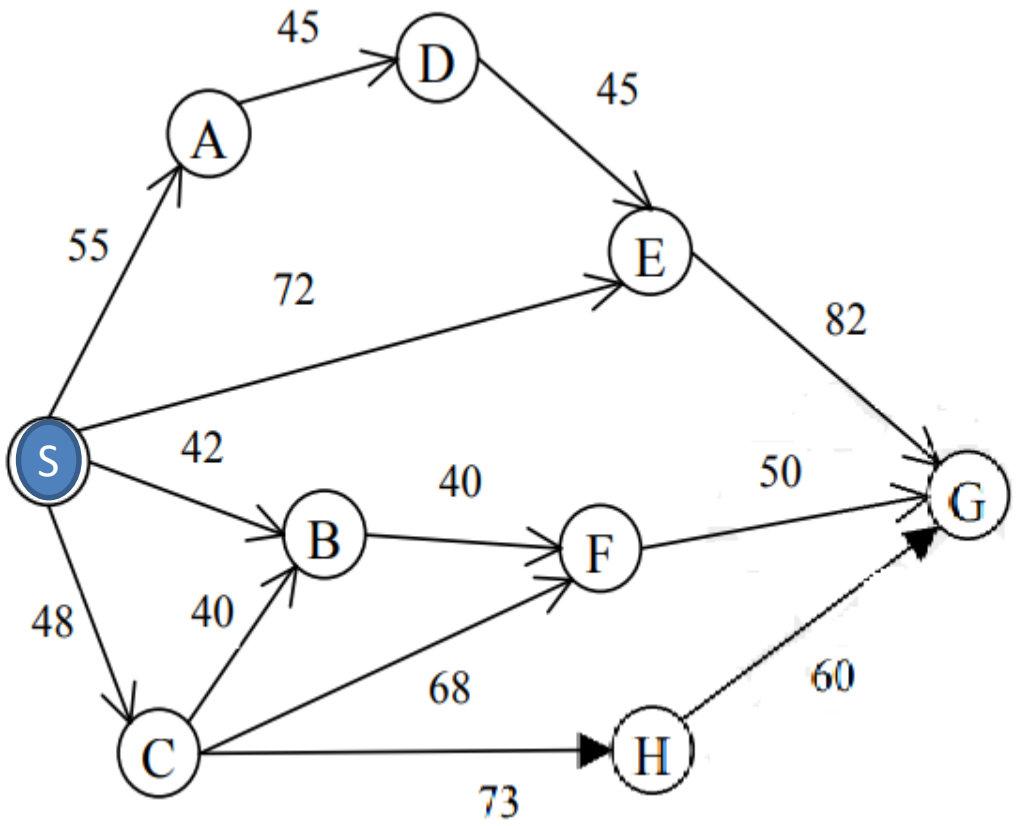


Nút được xét	Tập biên O (ngăn xếp LIFO trong trường hợp này)
C = 0	
	S
S	
C = 1	
	S
S	AS, ES, BS, CS
AS	ES, BS, CS
ES	BS, CS
BS	CS
CS	

Nút được xét	Tập biên O (ngăn xếp LIFO trong trường hợp này)
C = 2	
	S

Iterative Deepening

Ví dụ:

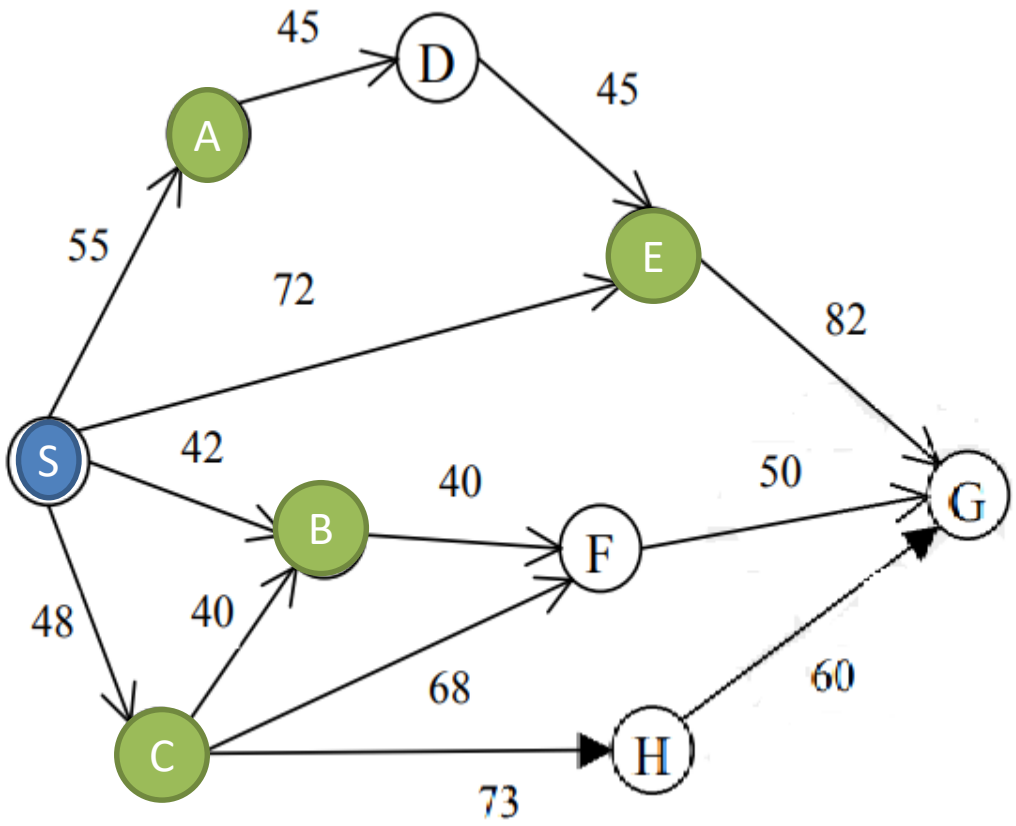


Nút được xét	Tập biên O (ngăn xếp LIFO trong trường hợp này)
C = 0	
	S
S	
C = 1	
	S
S	AS , BS , CS
ES	AS, BS
CS	AS, BS
BS	AS
AS	

Nút được xét	Tập biên O (ngăn xếp LIFO trong trường hợp này)
C = 2	
	S
S	

Iterative Deepening

Ví dụ:

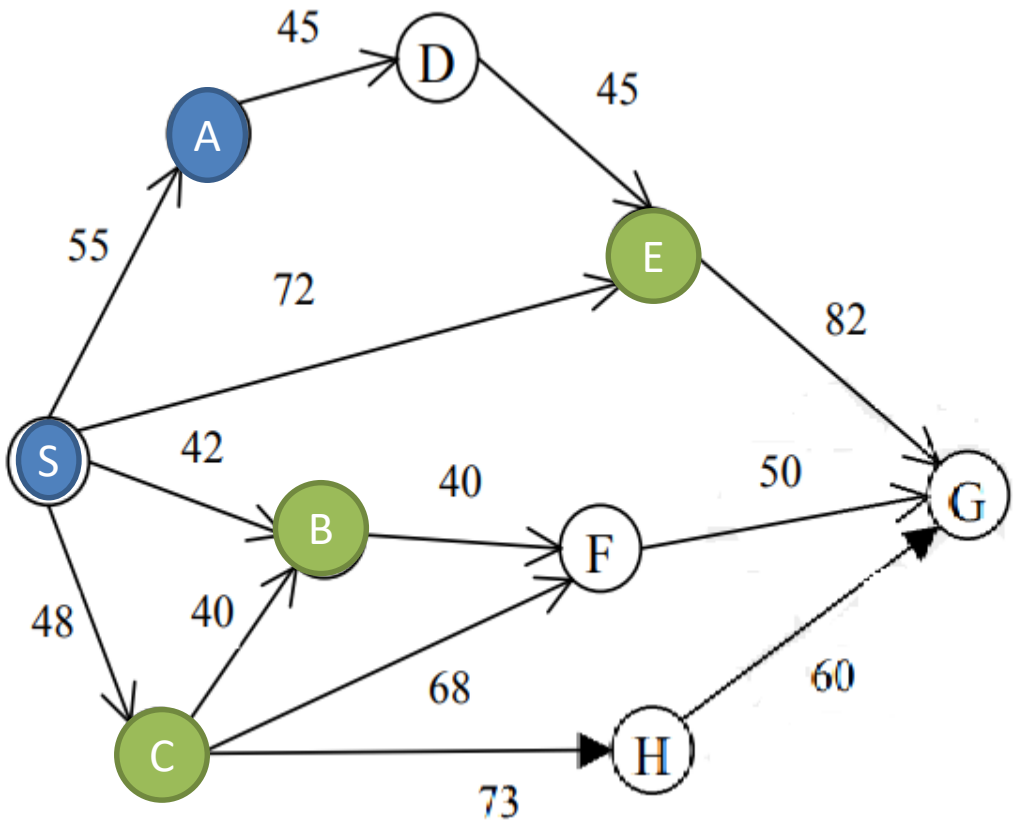


Nút được xét	Tập biên O (ngăn xếp LIFO trong trường hợp này)
C = 0	
	S
S	
C = 1	
	S
S	AS, ES, BS, CS
AS	ES, BS, CS
ES	BS, CS
BS	CS
CS	

Nút được xét	Tập biên O (ngăn xếp LIFO trong trường hợp này)
C = 2	
	S
S	AS, ES, BS, CS

Iterative Deepening

Ví dụ:

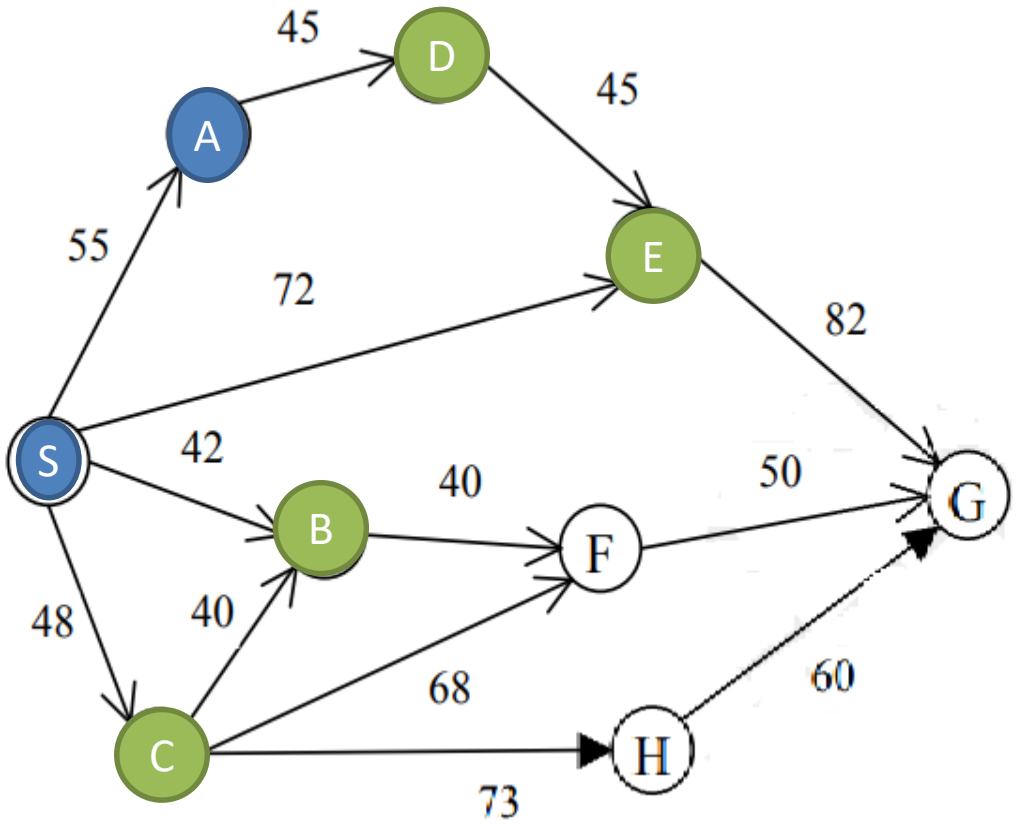


Nút được xét	Tập biên O (ngăn xếp LIFO trong trường hợp này)
C = 0	
	S
S	
C = 1	
	S
S	AS, ES, BS, CS
AS	ES, BS, CS
ES	BS, CS
BS	CS
CS	

Nút được xét	Tập biên O (ngăn xếp LIFO trong trường hợp này)
C = 2	
	S
S	AS, ES, BS, CS
AS	

Iterative Deepening

Ví dụ:

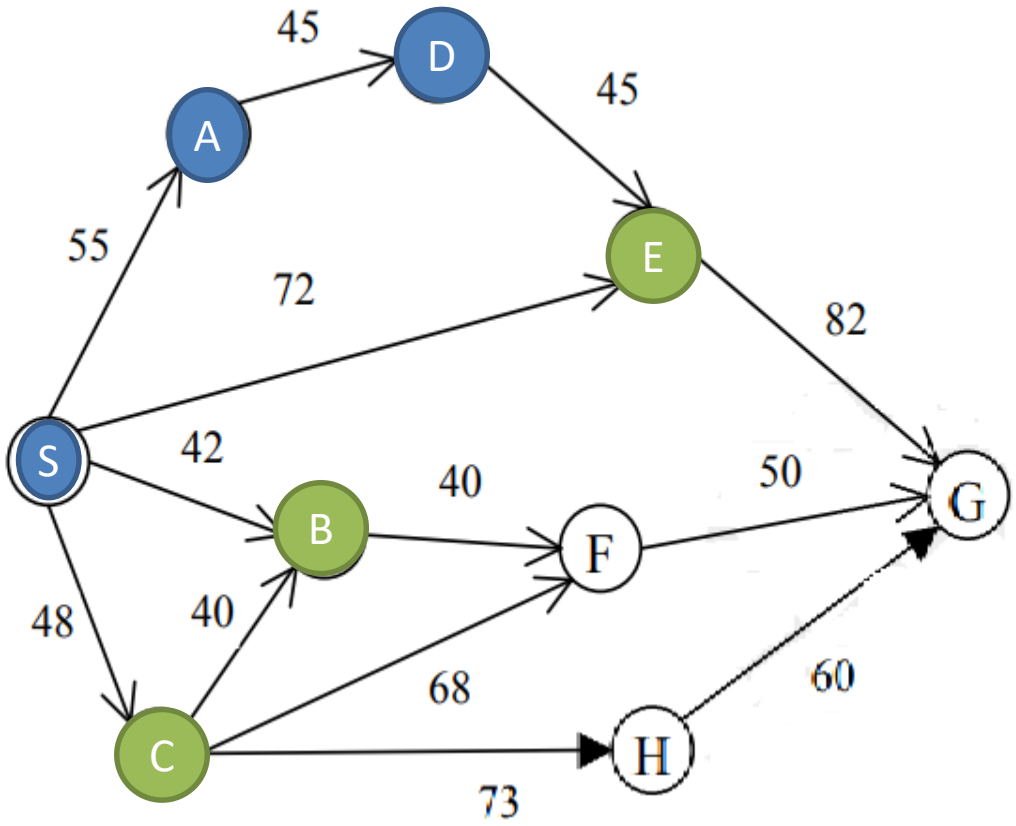


Nút được xét	Tập biên O (ngăn xếp LIFO trong trường hợp này)
C = 0	
	S
S	
C = 1	
	S
S	AS, ES, BS, CS
AS	ES, BS, CS
ES	BS, CS
BS	CS
CS	

Nút được xét	Tập biên O (ngăn xếp LIFO trong trường hợp này)
C = 2	
	S
S	AS, ES, BS, CS
AS	DA, ES, BS, CS

Iterative Deepening

Ví dụ:

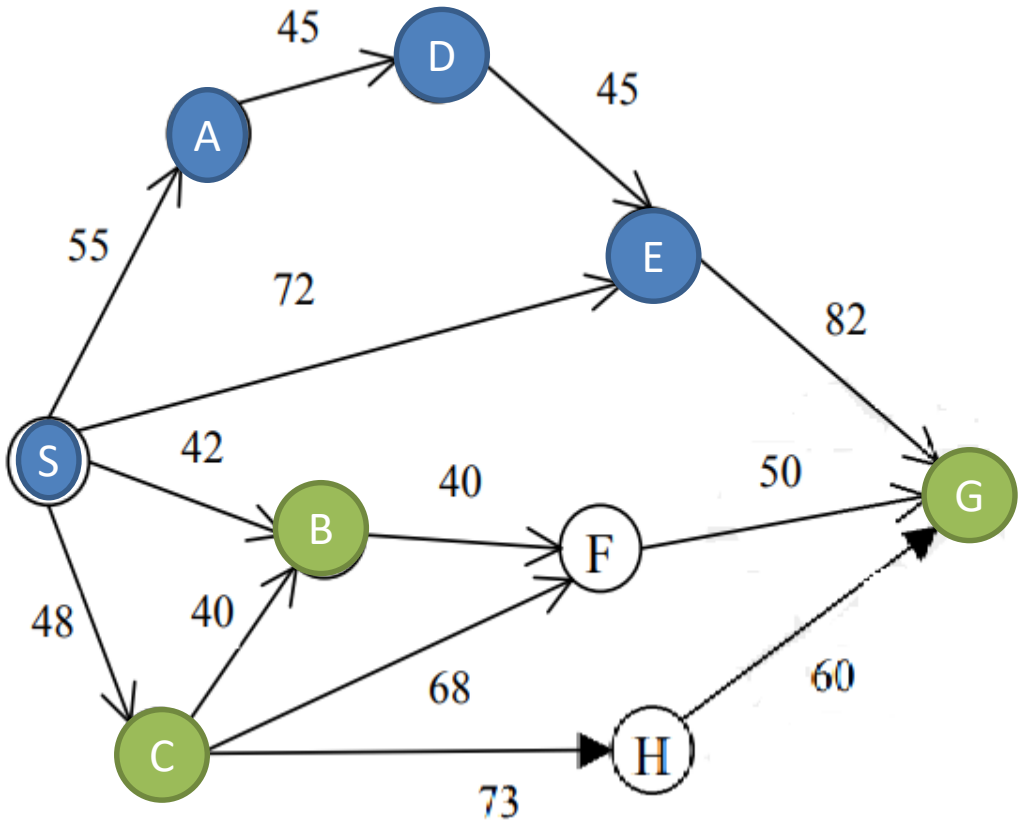


Nút được xét	Tập biên O (ngăn xếp LIFO trong trường hợp này)
C = 0	
	S
S	
C = 1	
	S
S	AS, ES, BS, CS
AS	ES, BS, CS
ES	BS, CS
BS	CS
CS	

Nút được xét	Tập biên O (ngăn xếp LIFO trong trường hợp này)
C = 2	
	S
S	AS, ES, BS, CS
AS	DA, ES, BS, CS
DA	ES, BS, CS

Iterative Deepening

Ví dụ:

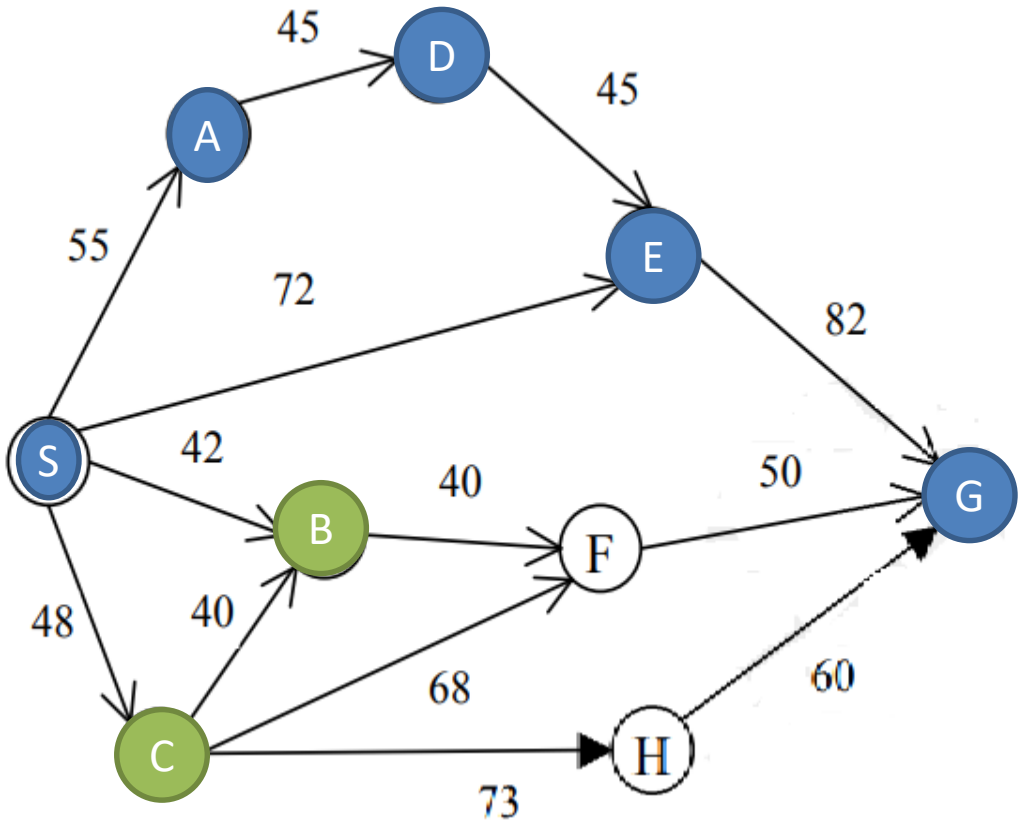


Nút được xét	Tập biên O (ngăn xếp LIFO trong trường hợp này)
C = 0	
	S
S	
C = 1	
	S
S	AS, ES, BS, CS
AS	ES, BS, CS
ES	BS, CS
BS	CS
CS	

Nút được xét	Tập biên O (ngăn xếp LIFO trong trường hợp này)
C = 2	
	S
S	AS, ES, BS, CS
AS	DA, ES, BS, CS
DA	ES, BS, CS
ES	GE, BS, CS

Iterative Deepening

Ví dụ:

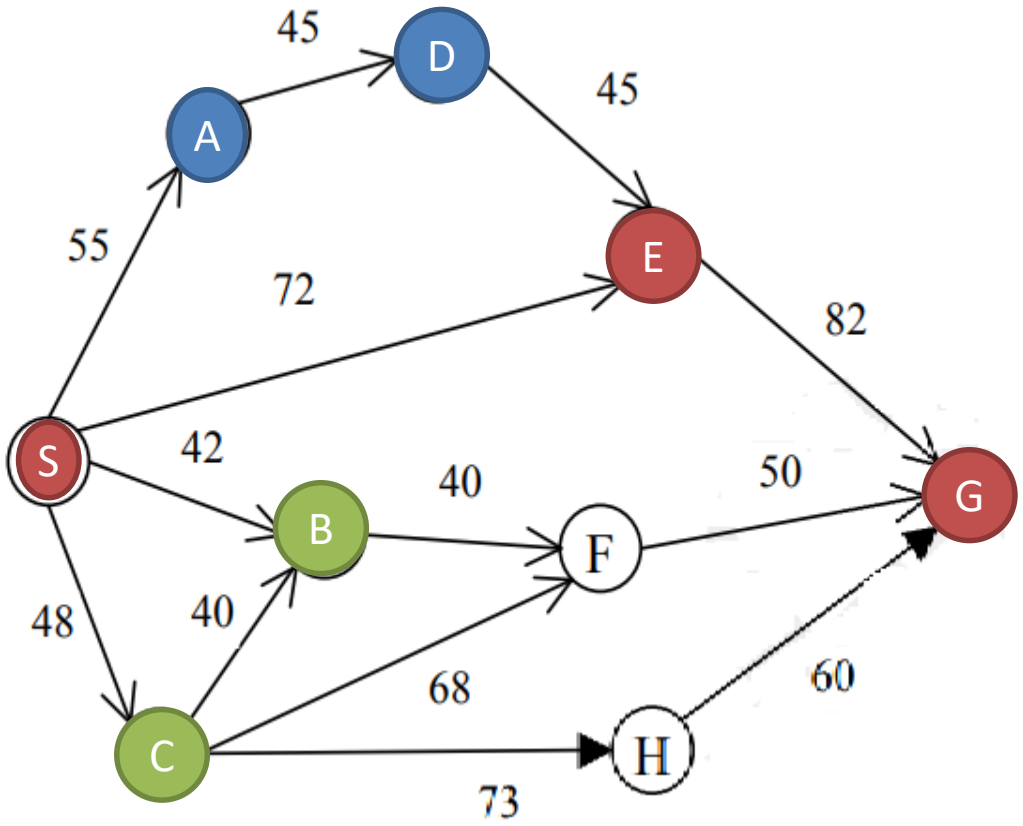


Nút được xét	Tập biên O (ngăn xếp LIFO trong trường hợp này)
C = 0	
	S
S	
C = 1	
	S
S	AS, ES, BS, CS
AS	ES, BS, CS
ES	BS, CS
BS	CS
CS	

Nút được xét	Tập biên O (ngăn xếp LIFO trong trường hợp này)
C = 2	
	S
S	AS, ES, BS, CS
AS	DA, ES, BS, CS
DA	ES, BS, CS
ES	GE, BS, CS
GE	

Iterative Deepening

Ví dụ:



Nút được xét	Tập biên O (ngăn xếp LIFO trong trường hợp này)
C = 0	
	S
S	
C = 1	
	S
S	AS, ES, BS, CS
AS	ES, BS, CS
ES	BS, CS
BS	CS
CS	

Nút được xét	Tập biên O (ngăn xếp LIFO trong trường hợp này)
C = 2	
	S
S	AS, ES, BS, CS
AS	DA, ES, BS, CS
DA	ES, BS, CS
ES	GE, BS, CS
GE	ĐÍCH

Iterative Deepening

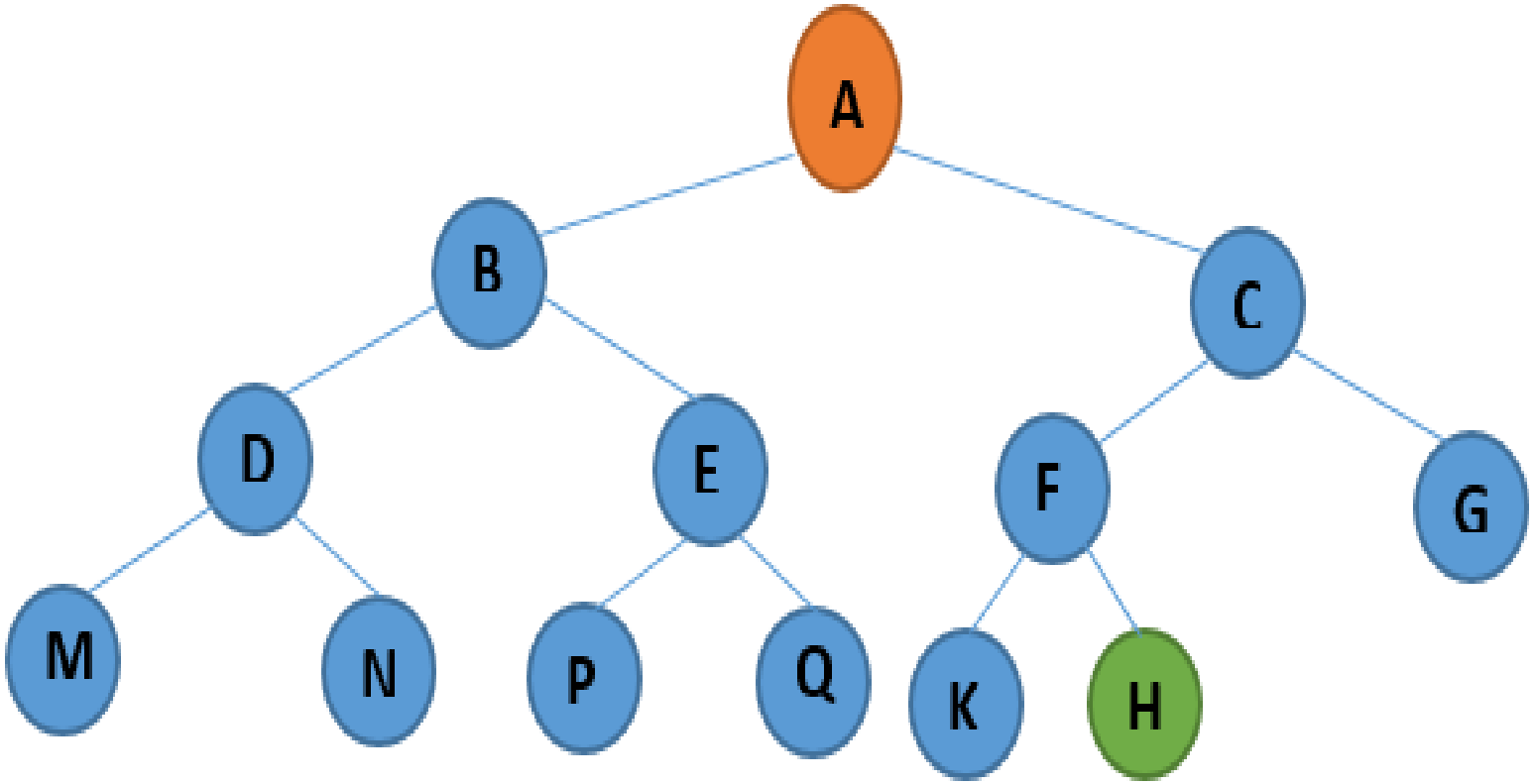
```
function depth_bounded_search(s, depth bound):
        if is goal(s):
                return <>
        if depth_bound > 0:
                for each <a,s'> ∈ succ(s):
                        solution := depth_bounded_search(s', depth_bound − 1)
                        if solution ≠ none:
                                solution.push_front(a)
                                return solution
return none
```

Iterative Deepening

Iterative Deepening DFS

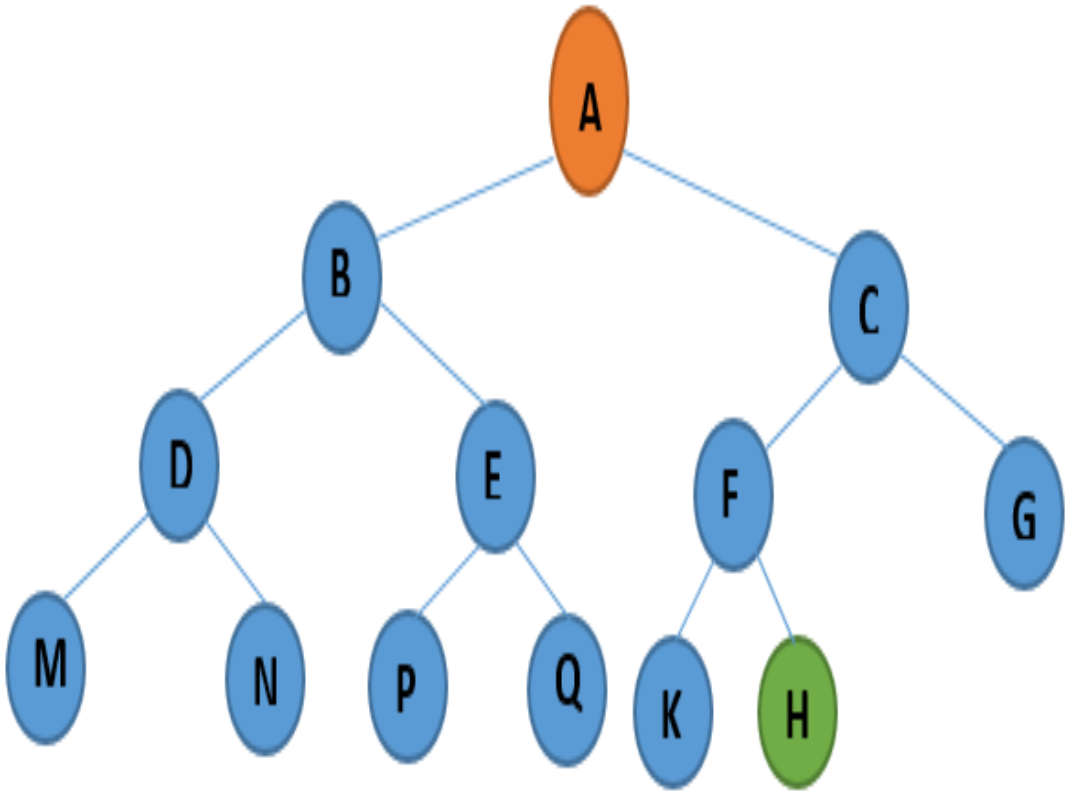
```
for depth bound ∈ {0, 1, 2, . . . }:
        solution := depth_bounded_search(init(), depth_bound)
        if solution ≠ none:
                return solution
```

Iterative Deepening



Iterative Deepening

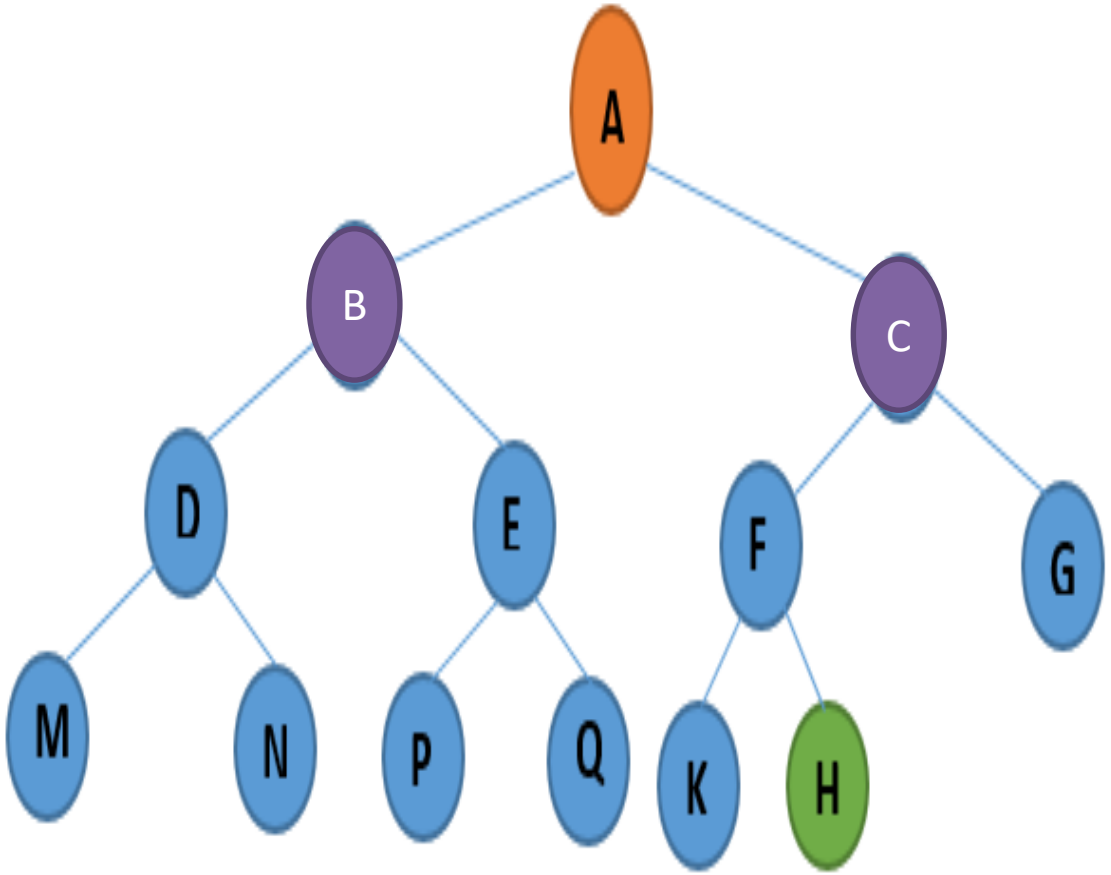
- depth bound: 0



Nút được xét	Tập biên O (ngăn xếp LIFO trong trường hợp này)
C = 0	
	A
A	

Iterative Deepening

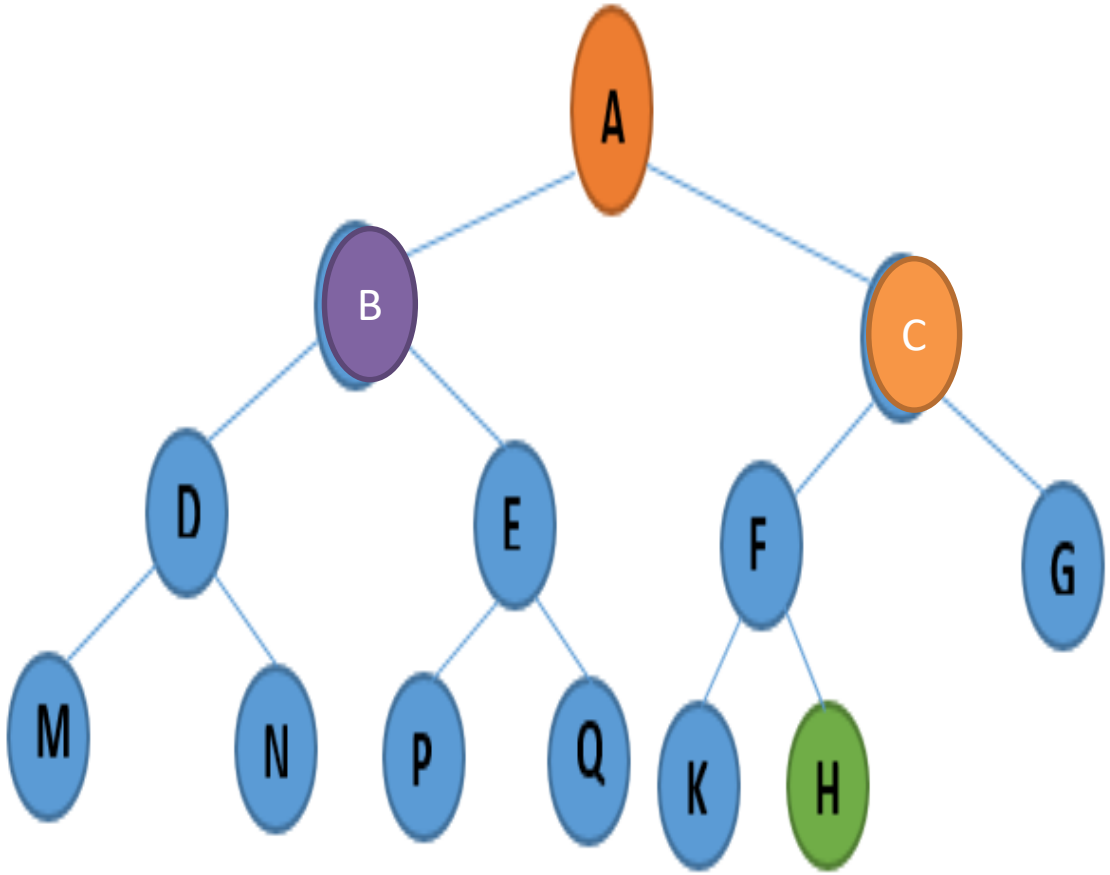
Ví dụ:



Nút được xét	Tập biên O (ngăn xếp LIFO trong trường hợp này)
C = 0	
	A
A	
C = 1	
	A
A	CA, BA

Iterative Deepening

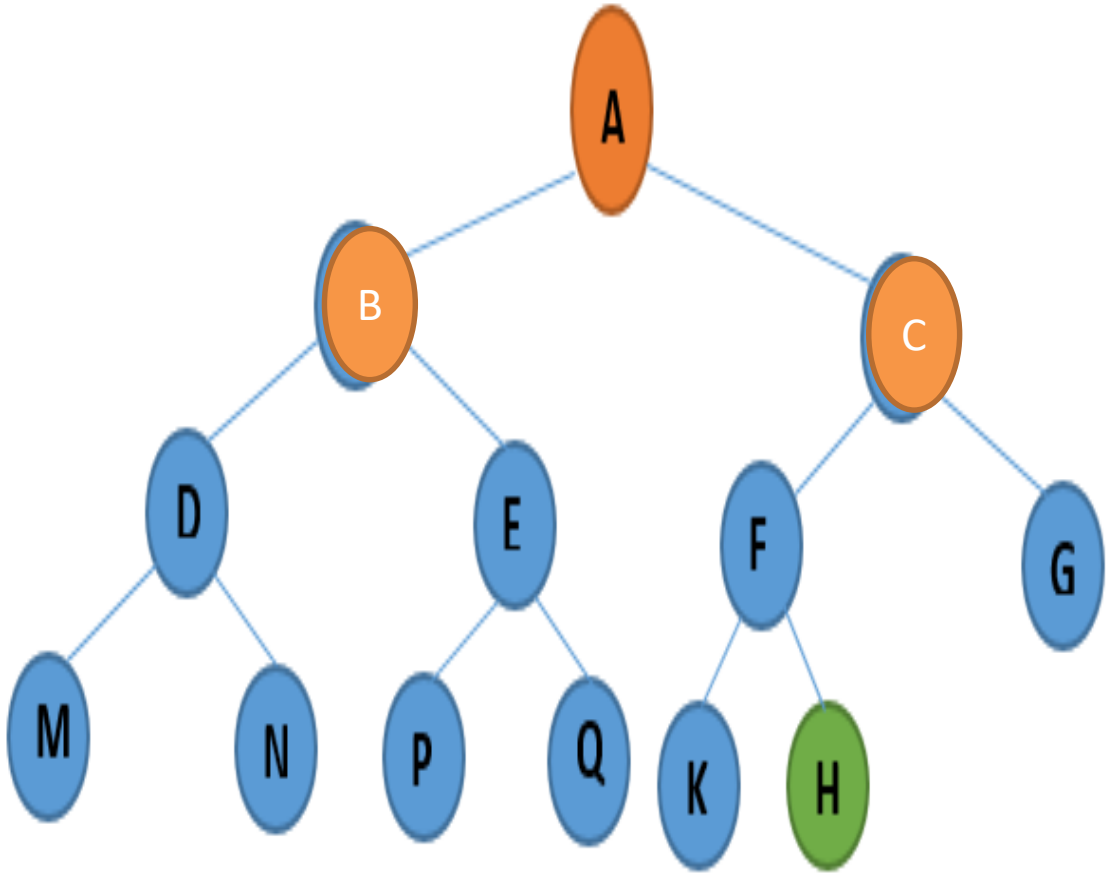
Ví dụ:



Nút được xét	Tập biên O (ngăn xếp LIFO trong trường hợp này)
C = 0	
	A
A	
C = 1	
	A
A	CA, BA
CA	BA

Iterative Deepening

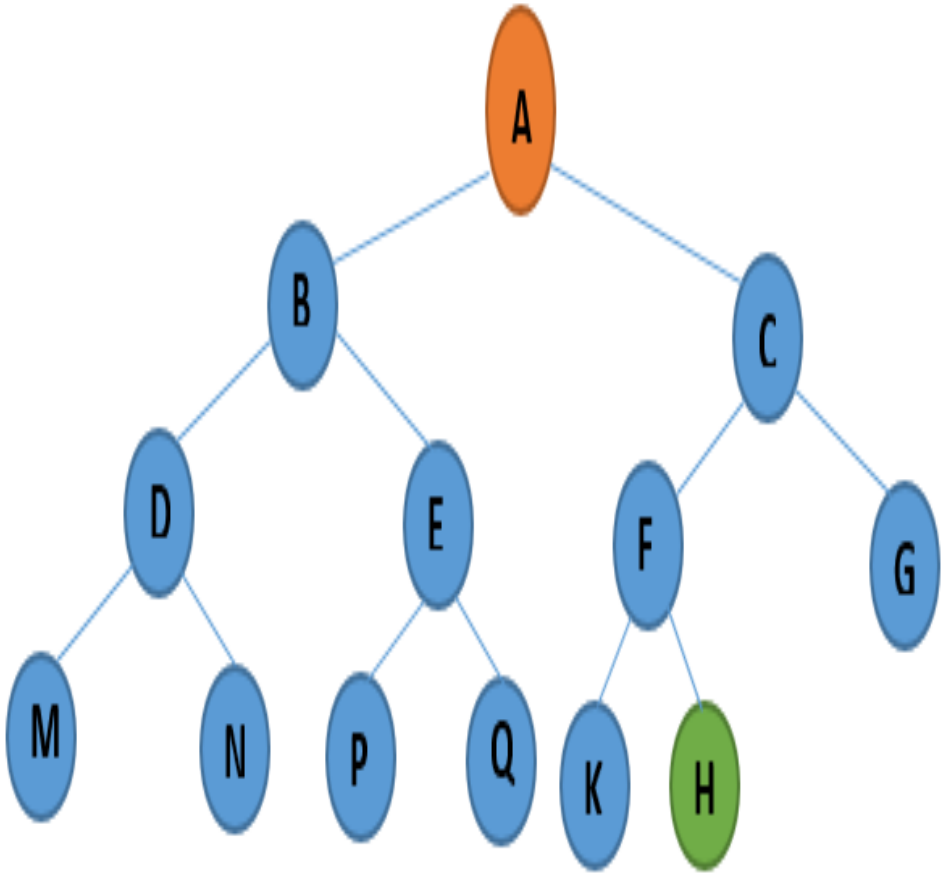
Ví dụ:



Nút được xét	Tập biên O (ngăn xếp LIFO trong trường hợp này)
C = 0	
	A
A	
C = 1	
	A
A	CA, BA
CA	BA
BA	

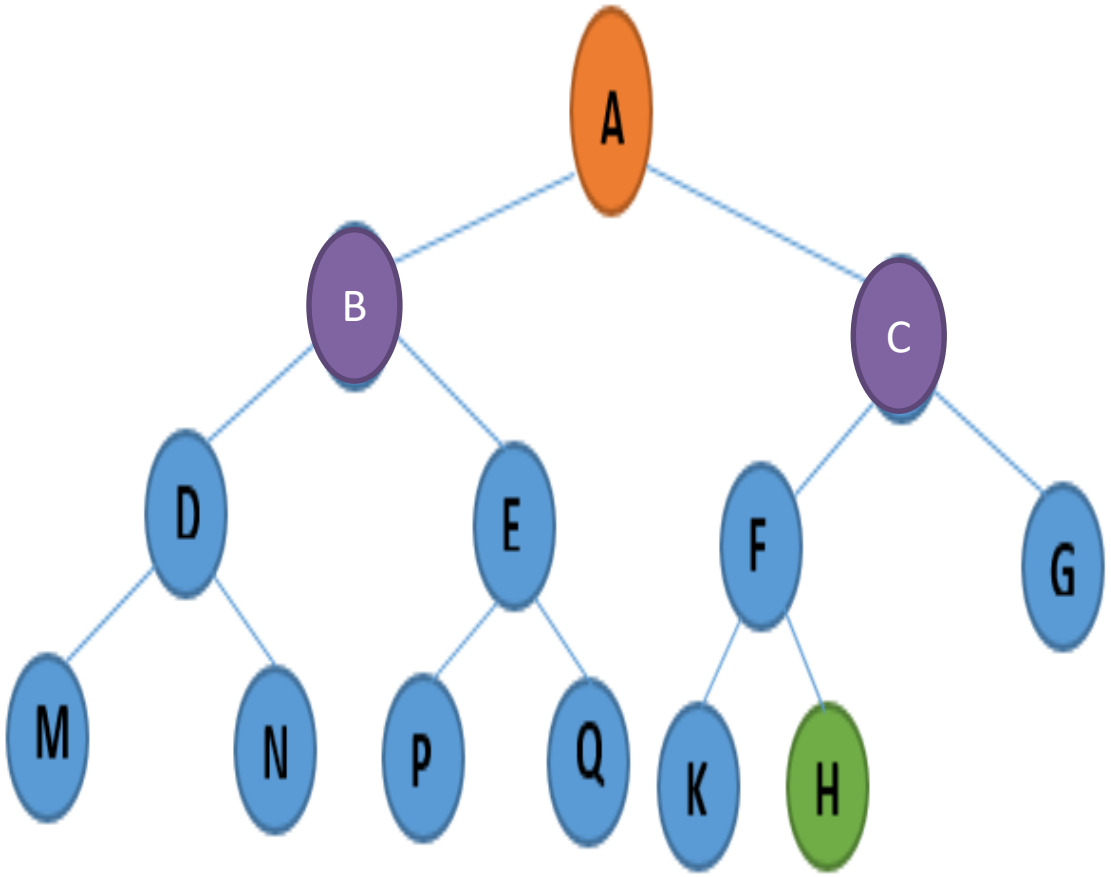
Iterative Deepening

Ví dụ:



Nút được xét	Tập biên O (ngăn xếp LIFO trong trường hợp này)
C = 0	
	A
A	
C = 1	
	A
A	CA, BA
CA	BA
BA	
C=2	
	A
A	

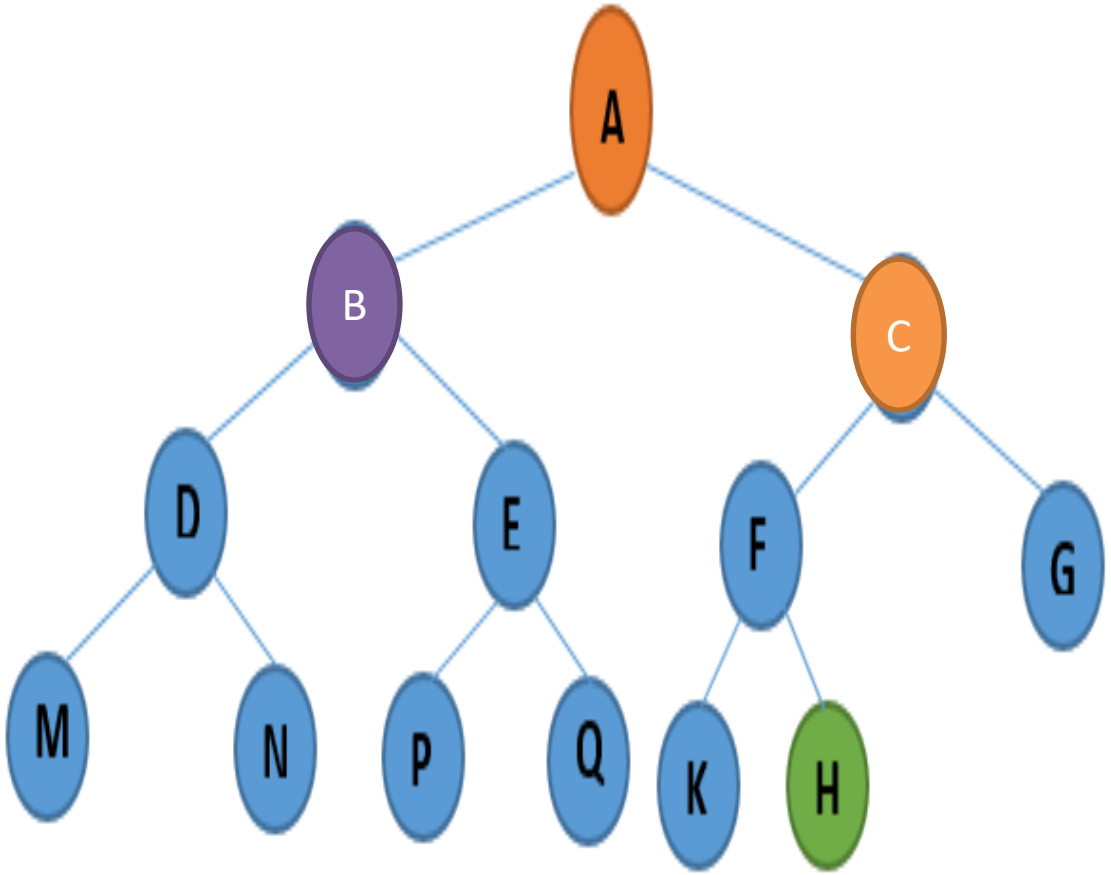
Iterative Deepening



Nút được xét	Tập biên O (ngăn xếp LIFO trong trường hợp này)
C = 0	
	A
A	
C = 1	
	A
A	CA, BA
CA	BA
BA	
C=2	
	A
A	CA, BA

Iterative Deepening

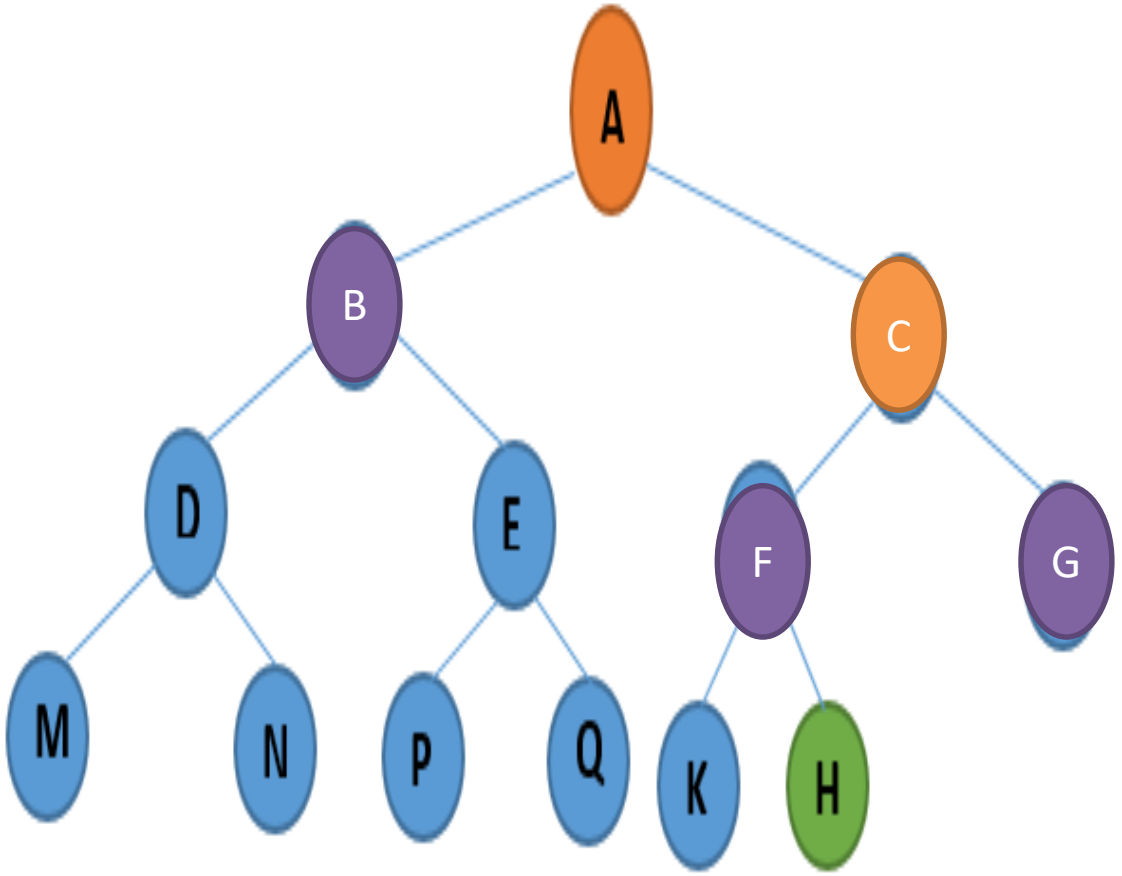
Ví dụ:



Nút được xét	Tập biên O (ngăn xếp LIFO trong trường hợp này)
C = 0	
	A
A	
C = 1	
	A
A	CA, BA
CA	BA
BA	
C=2	
	A
A	CA, BA
CA	

Iterative Deepening

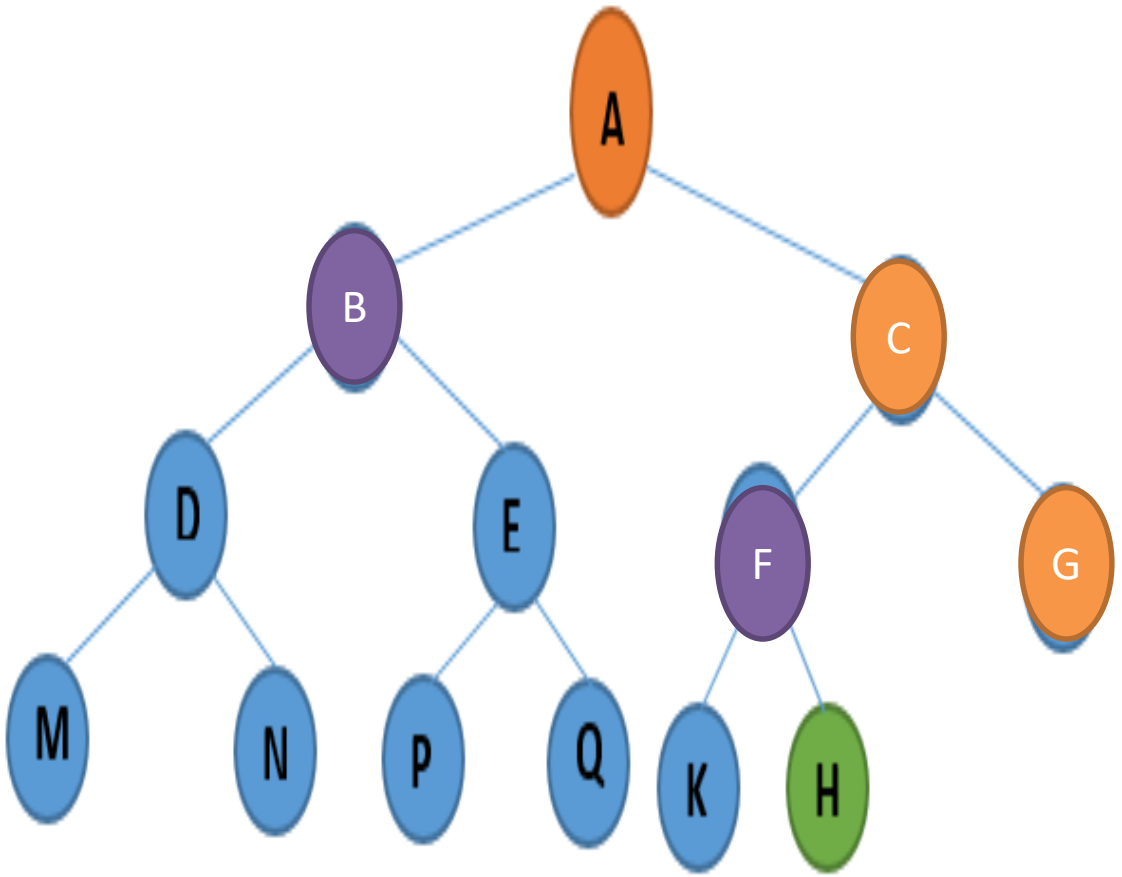
Ví dụ:



Nút được xét	Tập biên O (ngăn xếp LIFO trong trường hợp này)
C = 0	
	A
A	
C = 1	
	A
A	CA, BA
CA	BA
BA	
C=2	
	A
A	CA, BA
CA	GC,FC,BA

Iterative Deepening

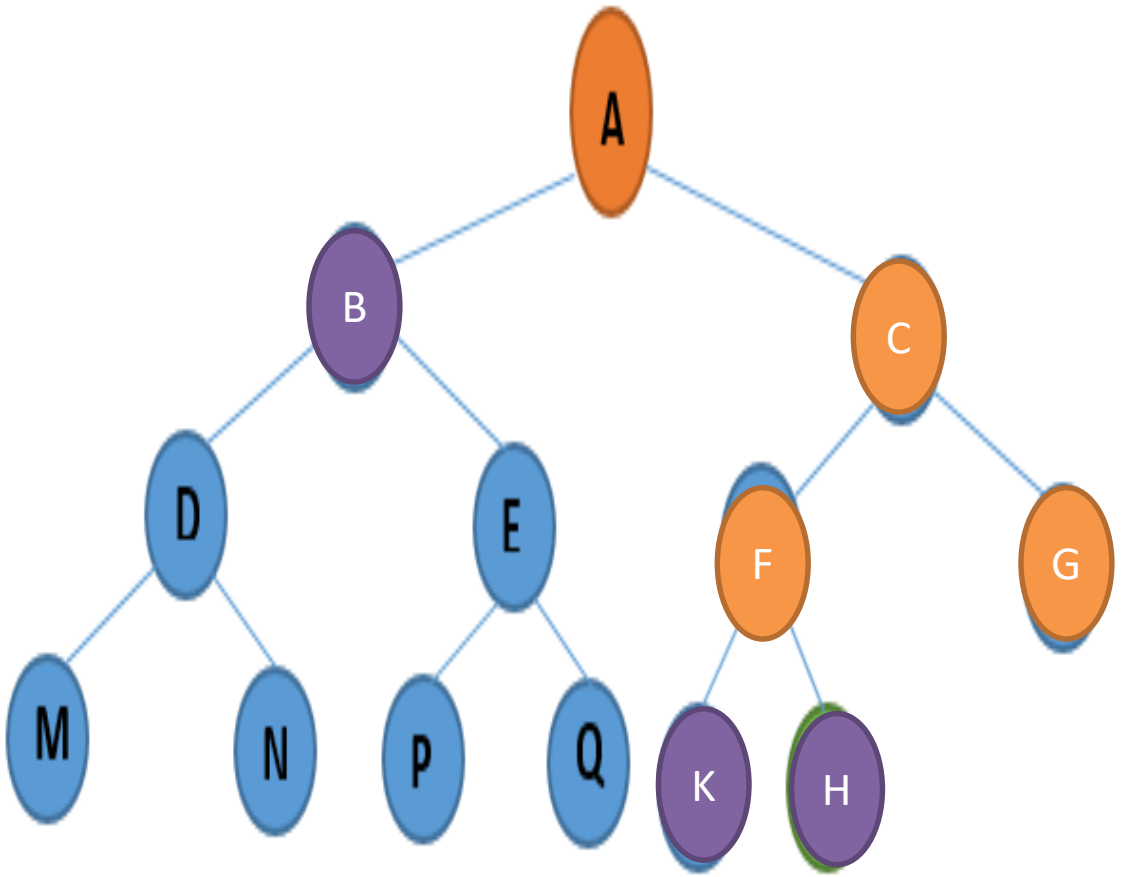
Ví dụ:



Nút được xét	Tập biên O (ngăn xếp LIFO trong trường hợp này)
C = 0	
	A
A	
C = 1	
	A
A	CA, BA
CA	BA
BA	
C=2	
	A
A	CA, BA
CA	GC,FC,BA
GC	FC,BA

Iterative Deepening

Ví dụ:

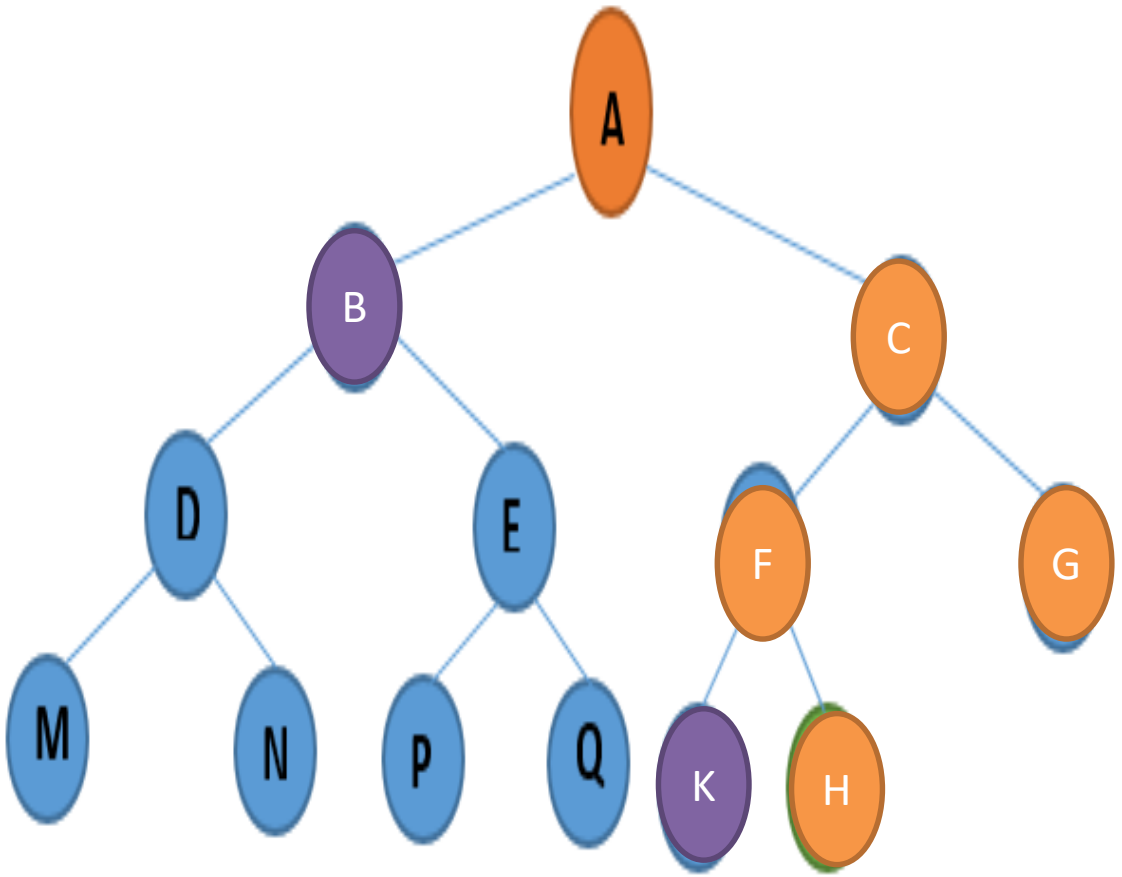


Nút được xét	Tập biên O (ngăn xếp LIFO trong trường hợp này)
C = 0	
	A
A	
C = 1	
	A
A	CA, BA
CA	BA
BA	
C=2	
	A
A	CA, BA
CA	GC,FC,BA
GC	FC,BA

Nút được xét	Tập biên O (ngăn xếp LIFO trong trường hợp này)
C = 2	
FC	HF,KF,BA

Iterative Deepening

Ví dụ:

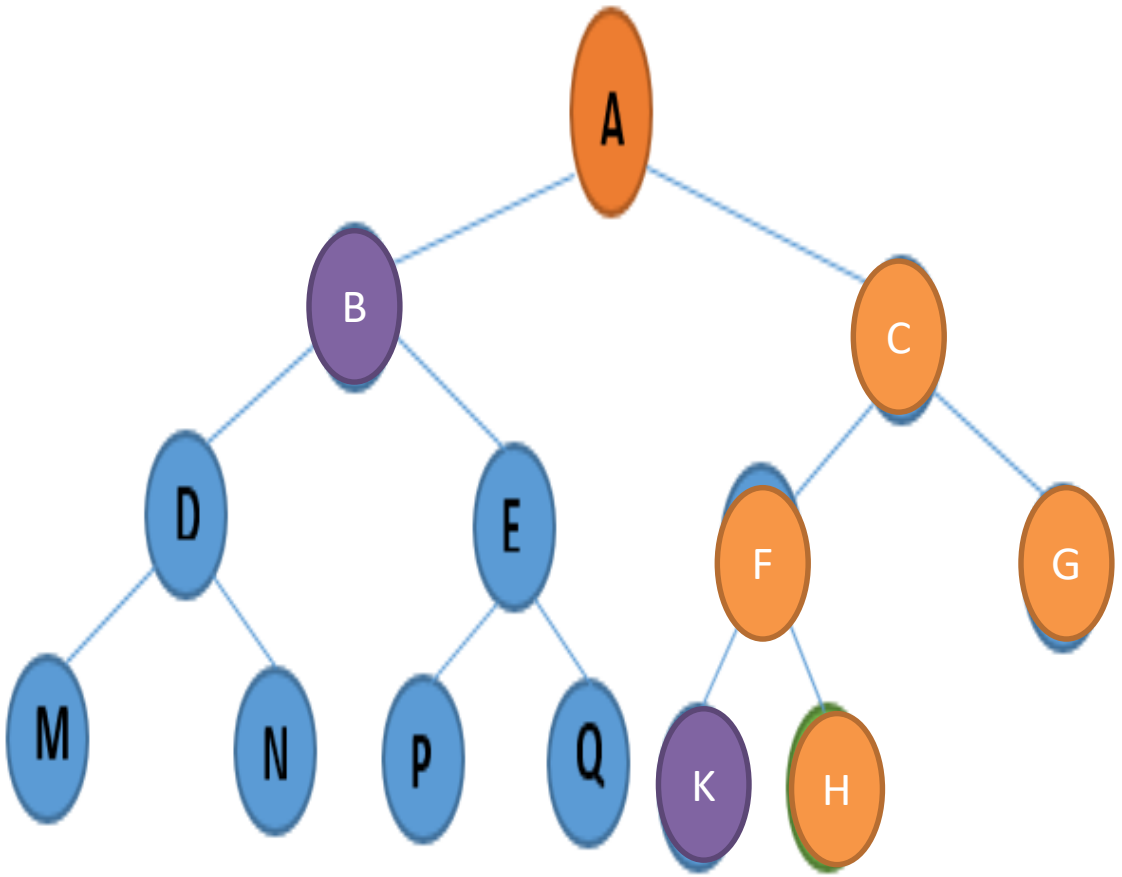


Nút được xét	Tập biên O (ngăn xếp LIFO trong trường hợp này)
C = 0	
	A
A	
C = 1	
	A
A	CA, BA
CA	BA
BA	
C=2	
	A
A	CA, BA
CA	GC,FC,BA
GC	FC,BA

Nút được xét	Tập biên O (ngăn xếp LIFO trong trường hợp này)
C = 2	
FC	HF,KF,BA
HF	

Iterative Deepening

Ví dụ:

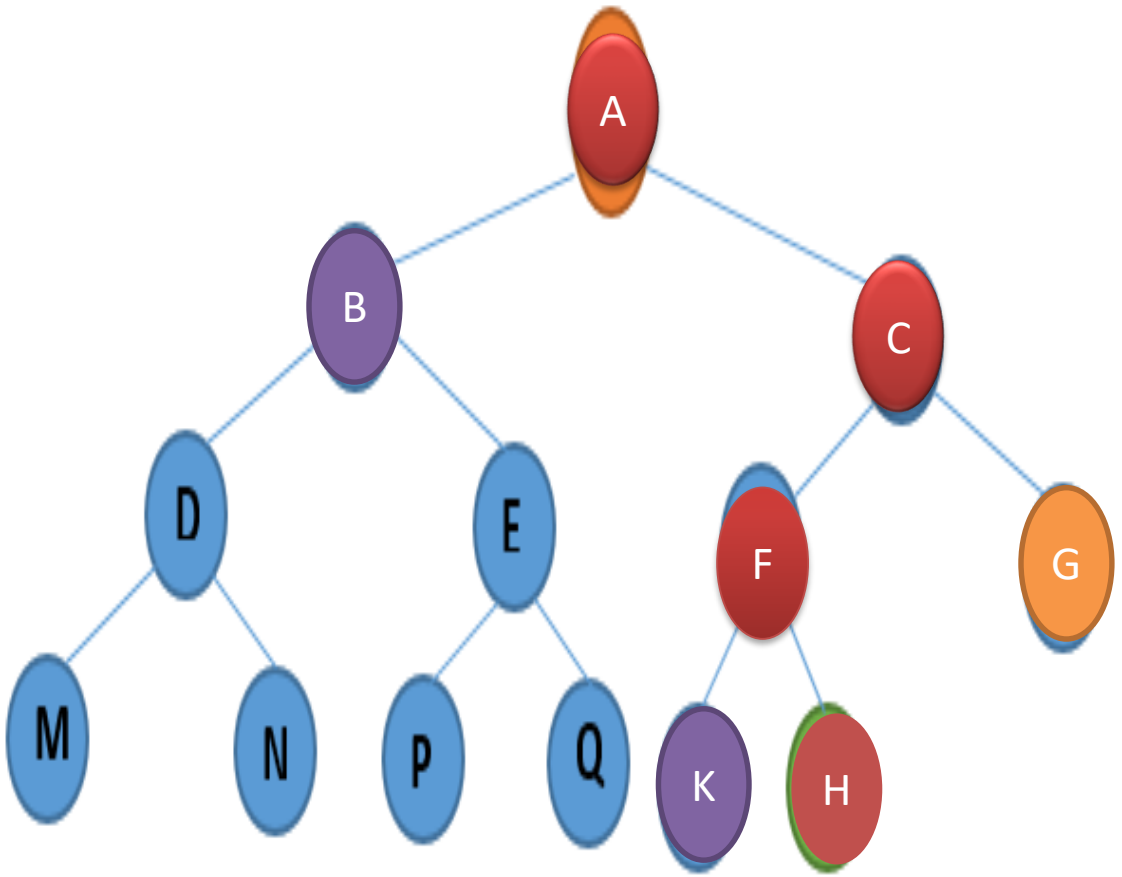


Nút được xét	Tập biên O (ngăn xếp LIFO trong trường hợp này)
C = 0	
	A
A	
C = 1	
	A
A	CA, BA
CA	BA
BA	
C=2	
	A
A	CA, BA
CA	GC,FC,BA
GC	FC,BA

Nút được xét	Tập biên O (ngăn xếp LIFO trong trường hợp này)
C = 2	
FC	HF,KF,BA
HF	ĐÍCH

Iterative Deepening

Ví dụ:



Nút được xét	Tập biên O (ngăn xếp LIFO trong trường hợp này)
C = 0	
	A
A	
C = 1	
	A
A	CA, BA
CA	BA
BA	
C=2	
	A
A	CA, BA
CA	GC,FC,BA
GC	FC,BA

Nút được xét	Tập biên O (ngăn xếp LIFO trong trường hợp này)
C = 2	
FC	HF,KF,BA
HF	ĐÍCH

Iterative Deepening

- Tính chất của thuật toán Tìm kiếm sâu dần:
  - Thuật toán đầy đủ, tức là đảm bảo tìm ra lời giải nếu có.
  - Thuật toán tối ưu nếu giá thành đường đi giữa hai nút giống nhau
  - Yêu cầu bộ nhớ nhỏ. Cụ thể, yêu cầu bộ nhớ là O(bd)
  - Độ phức tạp tính toán $O(b^d)=(d+1)b^0 + d\,b^1 + (d-1)b^2 + \ldots + b^d$

Comparison of Blind Search Algorithms

	search algorithm				
criterion	breadth-first	uniform cost	depth-first	depth-bounded	iterative deepening
complete?	yes*	yes	no	no	semi
optimal?	yes**	yes	no	no	yes**
time	$O(b^d)$	$O(b^{\lfloor c^*/\varepsilon \rfloor + 1})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$
space	$O(b^d)$	$O(b^{\lfloor c^*/\varepsilon \rfloor + 1})$	$O(bm)$	$O(b\ell)$	$O(bd)$

Bài tập

Thực hiện tìm lời giải theo phương pháp duyệt BFS cho bài toán 8-puzzle, thể hiện bằng:

- Tree search (vẽ lên giấy) chụp đưa vào word=> PDF
- Viết chương trình bằng python => py

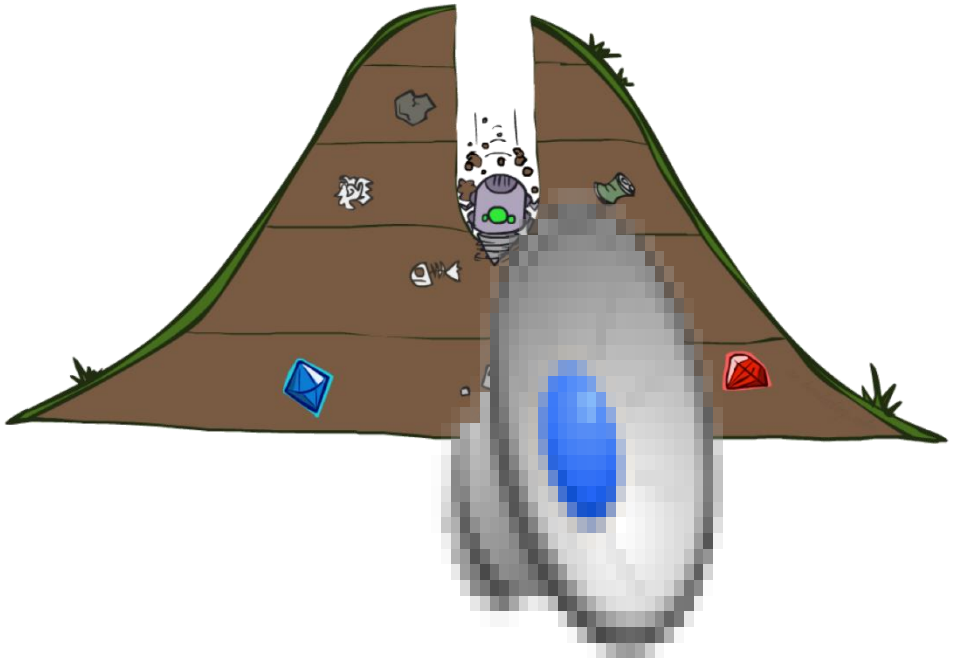
trạng thái xuất phát

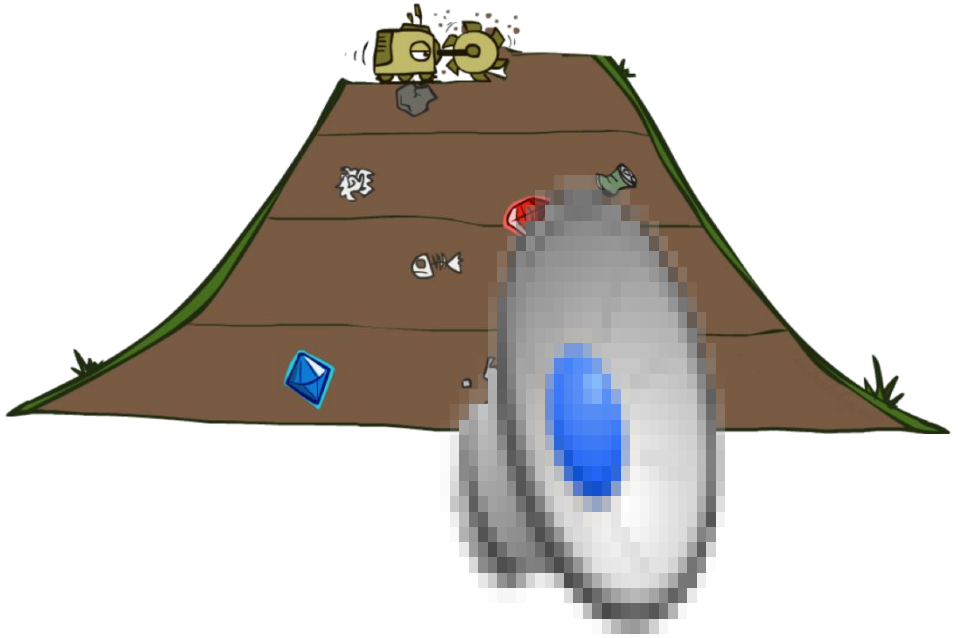
2	6	5
	8	7
4	3	1

trạng thái đích

1	2	3
4	5	6
7	8	

DFS vs BFS





Video of
Demo Maze
Water
DFS/BFS

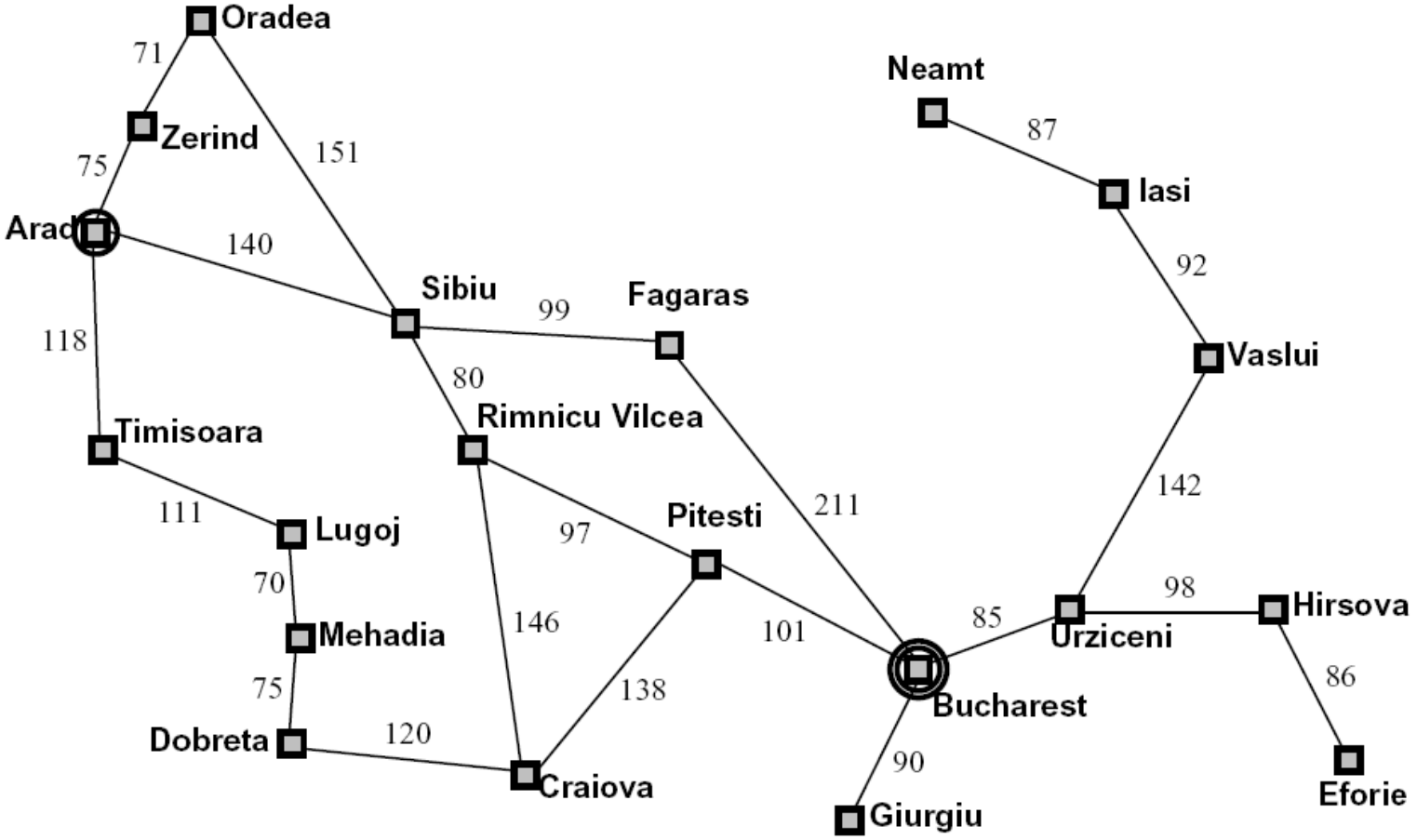
- When will BFS outperform DFS?
- When will DFS outperform BFS?

Bài tập

- Thể hiện quá trình tìm đường đi bằng phương pháp DFS, BFS, với

➢ Trạng thái đầu: **Arad**

➢ Trạng thái đích: **Bucharest**



(*) Viết chương trình python

Thanks for your attention!

Q&A