Search for Solutions

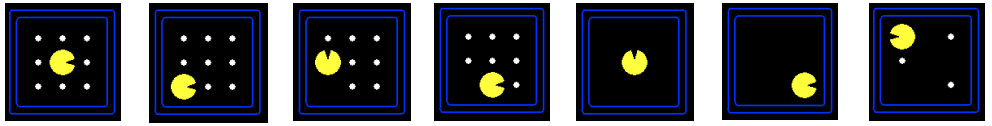
(ARTIFICIAL INTELLIGENCE)

(slides adapted from :

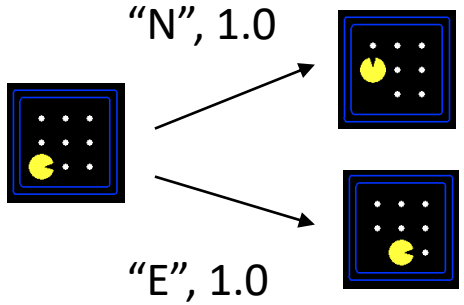
1. Dan Klein, Pieter Abbeel, Anca Dragan, et al
2. Artificial Intelligence: A modern approach (Stuart J. Russell and Peter Norvig))

Search Problems

- Trong AI, các kỹ thuật tìm kiếm là phương pháp giải quyết vấn đề phổ biến



- Một search problem bao gồm:
  - Một không gian trạng thái
  - Một tập các hành động và chi phí
  - Một trạng thái bắt đầu và một trạng thái kết thúc (mục tiêu)



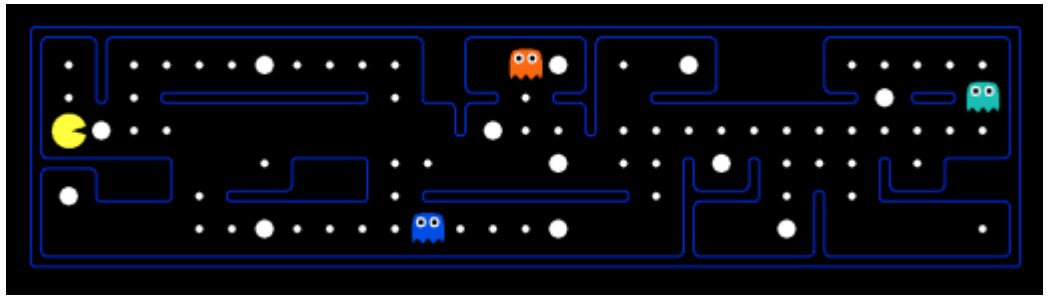
- Một solution là một chuỗi hành động (một kế hoạch) để chuyển đổi trạng thái từ trạng thái bắt đầu thành trạng thái mục tiêu

Search Problems

- Các thuật ngữ cần quan tâm:
  - Search: là một quy trình từng bước để giải quyết vấn đề.
  - 3 main factors: Search Space; Start State; Goal test
  - Search tree: represent a search problem.
  - Actions.
  - Transition model.
  - Path Cost.
  - Solution.
  - Optimal Solution.

Không gian trạng thái có gì?

- Toàn bộ world state là bao gồm mọi chi tiết cuối cùng của môi trường



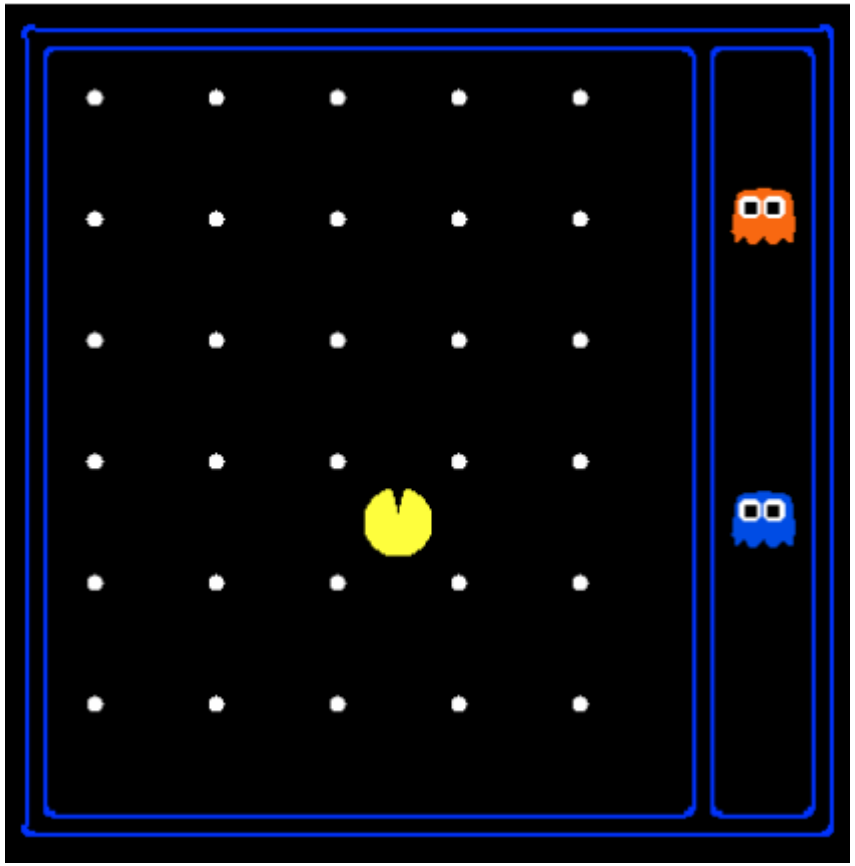
- Một search state chỉ giữ lại những chi tiết cần thiết cho solution!

Bài toán: Pathing
o Các trạng thái: (x,y) location
o Các hành động: NSE W
o Trạng thái tiếp theo: Chỉ cập nhật thông tin vị trí
o Trạng thái mục tiêu: is (x,y)=E ND

Bài toán: ăn tất cả các dấu chấm (dot)
Trạng thái: {(x, y); các dot boolean}
Hành động: N, S, E, W
Trạng thái kế: cập nhật một thông tin vị trí và (có thể) cập nhật dot (khi ăn)
Hàm kiểm tra đích: Tất cả các dot boolean đều false

Không gian trạng thái có kích thước?

- World state
  - Số vị trí của agent: 120 (do 1 bảng=>10 cột, 12 hàng)
  - Số dấu chấm: 30 (5 cột và 6 hàng có dot)
  - Số vị trí của một con ma: 12 (có 2 con ma)
  - Agent có 4 lựa chọn: N, S, E, W
- Bao nhiêu:
  - World states?
    - 120x(2^30)x(12^2)x4
  - Các trạng thái cho bài toán tìm đường:
    - 120
  - Các trạng thái cho bài toán ăn tất cả dấu chấm
    - 120x(2^30)



Search Problems

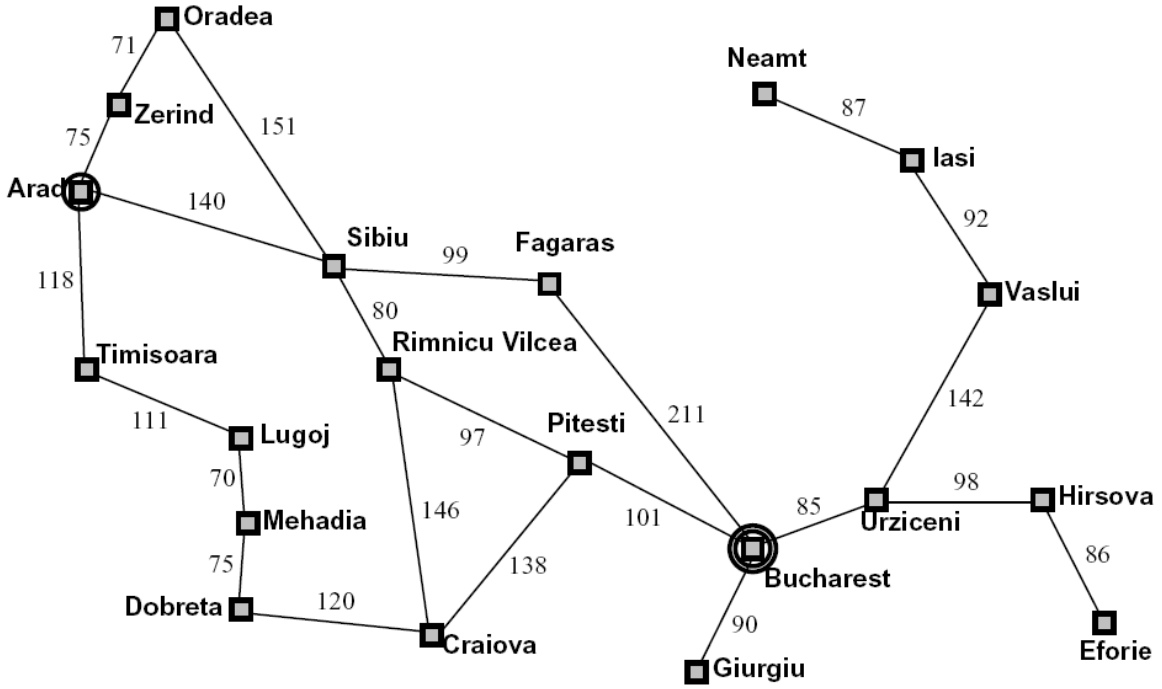
Các thành phần của một search problem

- Không gian trạng thái:
  - Các thành phố
- Tập các hành động và chi phí:
  - Hành động: các con đường
  - Chi phí

  Đi đến thành phố lân cận với chi phí = khoảng cách
- Trạng thái bắt đầu:
  - Arad
- Trạng thái mục tiêu:
  - Is state == Bucharest?

- Giải pháp là gì?

Ví dụ: Traveling in Romania

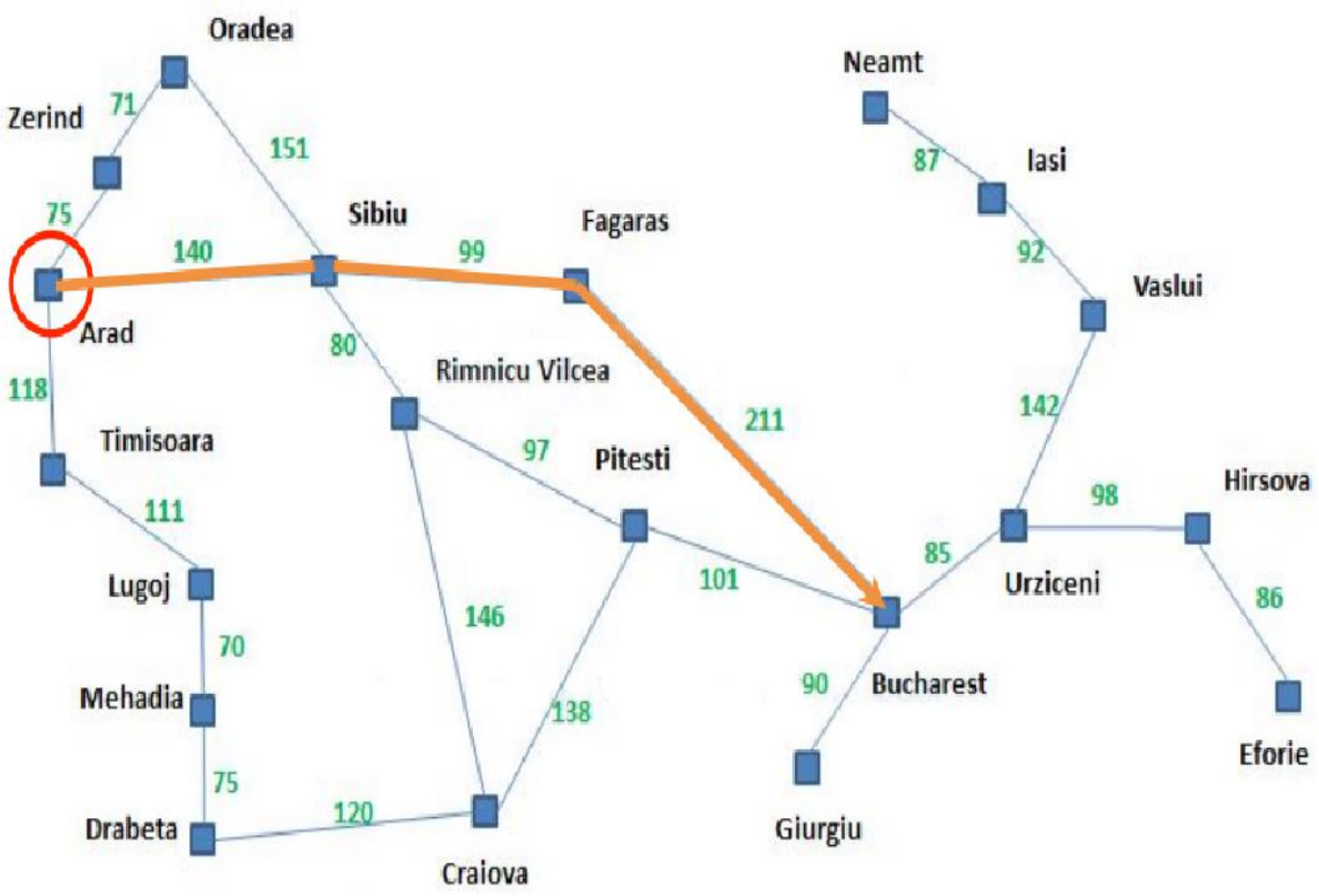


Search Problems

Các thành phần của một search problem

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

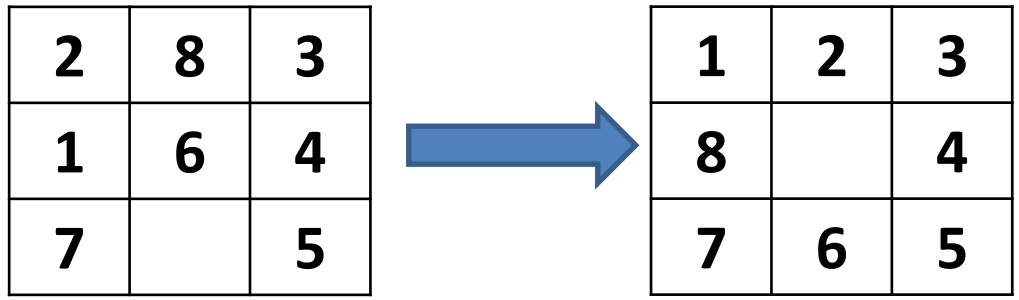
Ví dụ: Traveling in Romania



Search Problems

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

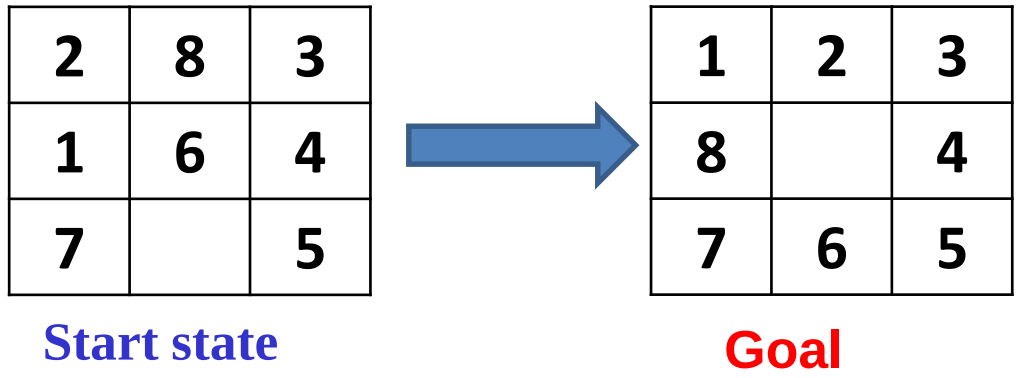
- Không gian trạng thái?
- Các hành động?
- Trạng thái xuất phát?
- Trạng thái mục tiêu?
- Chi phí?



Start state **Goal**

Search Problems

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

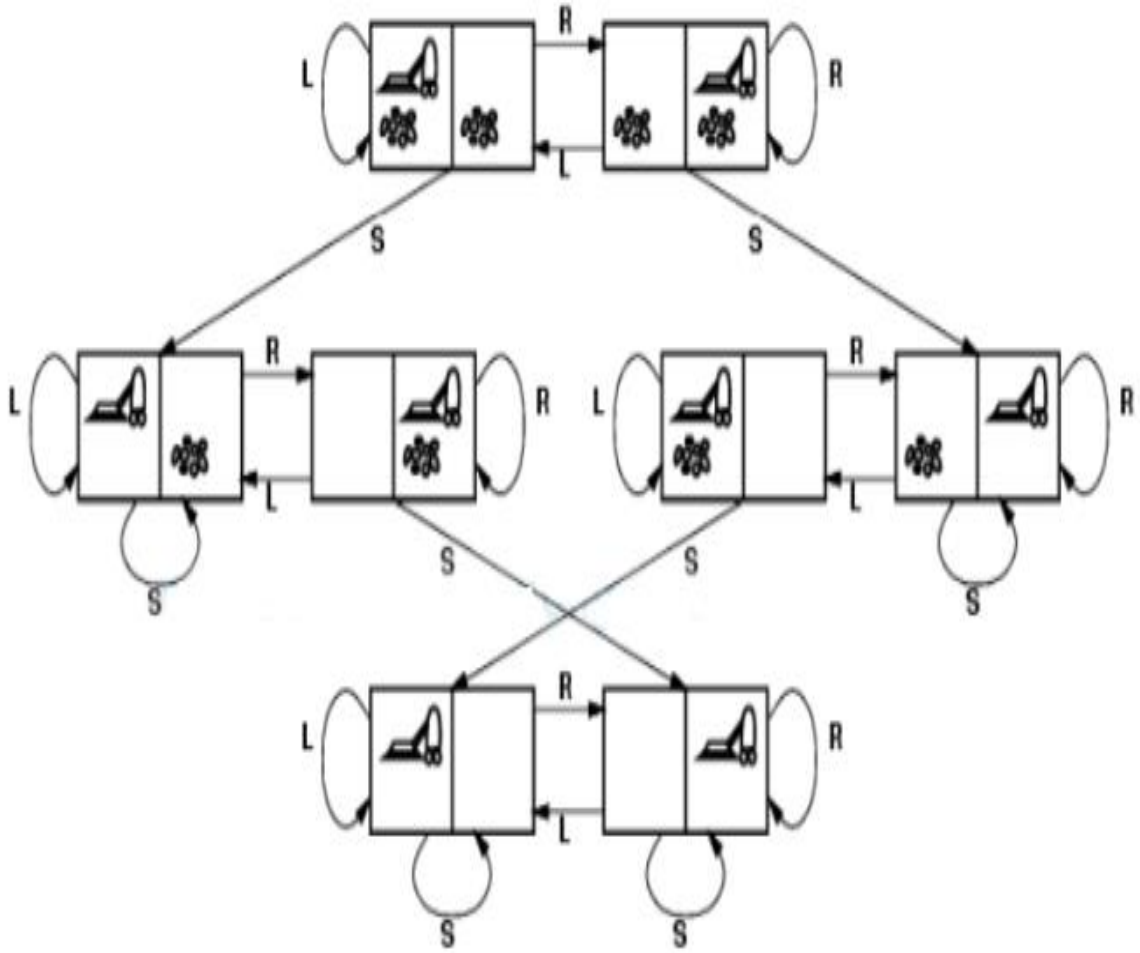


- Không gian trạng thái?
  - Các sắp xếp cụ thể vị trí các ô
- Các hành động?
  - di chuyển ô trống trái, phải, lên, xuống
- Trạng thái xuất phát?
  - Trạng thái bên trái (ở hình trên)
- Trạng thái mục tiêu?
  - Trạng thái bên phải (ở hình trên)
- Chi phí?
  - Mỗi chuyển động có giá thành bằng 1

Search Problems

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

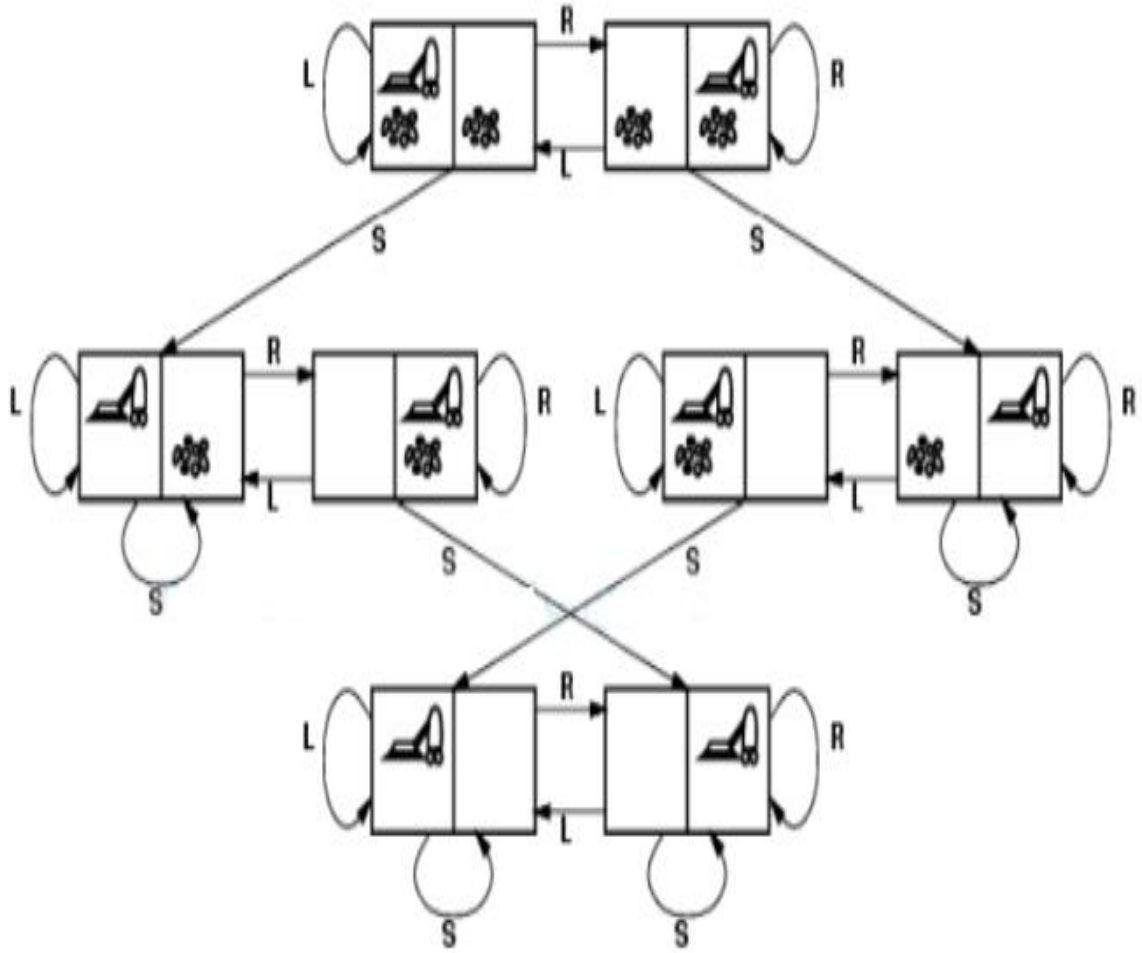
- Không gian trạng thái?
- Các hành động?
- Trạng thái xuất phát?
- Trạng thái mục tiêu?
- Chi phí?



Search Problems

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

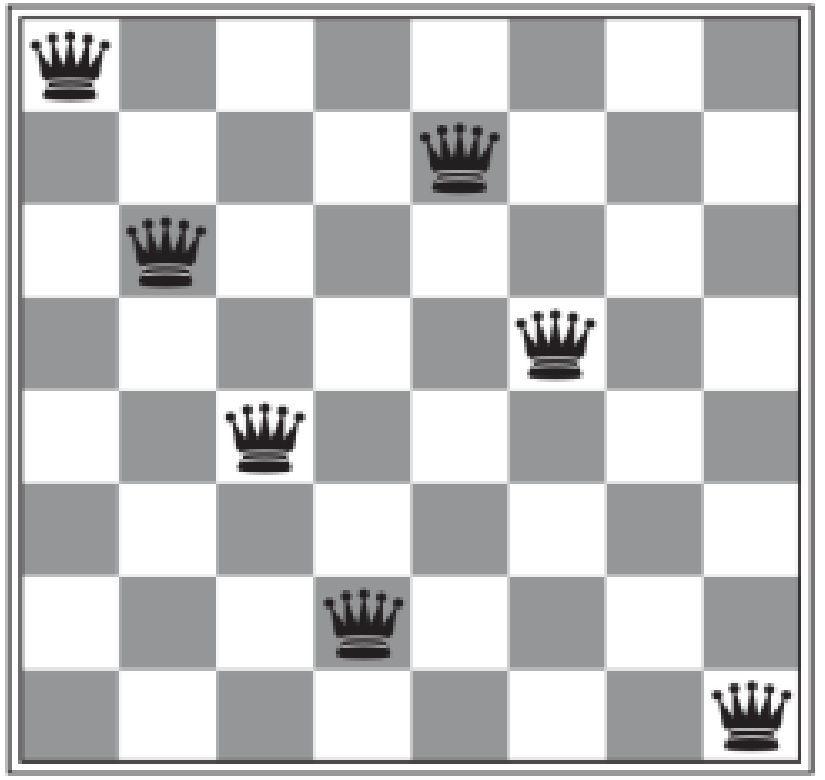
- Không gian trạng thái?
  - Chổ bẩn và vị trí máy hút bụi
- Các hành động?
  - Sang trái, sang phải, hút bụi, không làm gì
- Trạng thái xuất phát?
  - Trạng thái bên trái (ở hình trên)
- Trạng thái mục tiêu?
  - Không còn vị trí nào bẩn
- Chi phí?
  - 1 (mỗi hành động), 0 (không làm gì)



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?
- Trạng thái xuất phát?
- Trạng thái mục tiêu?
- Chi phí?

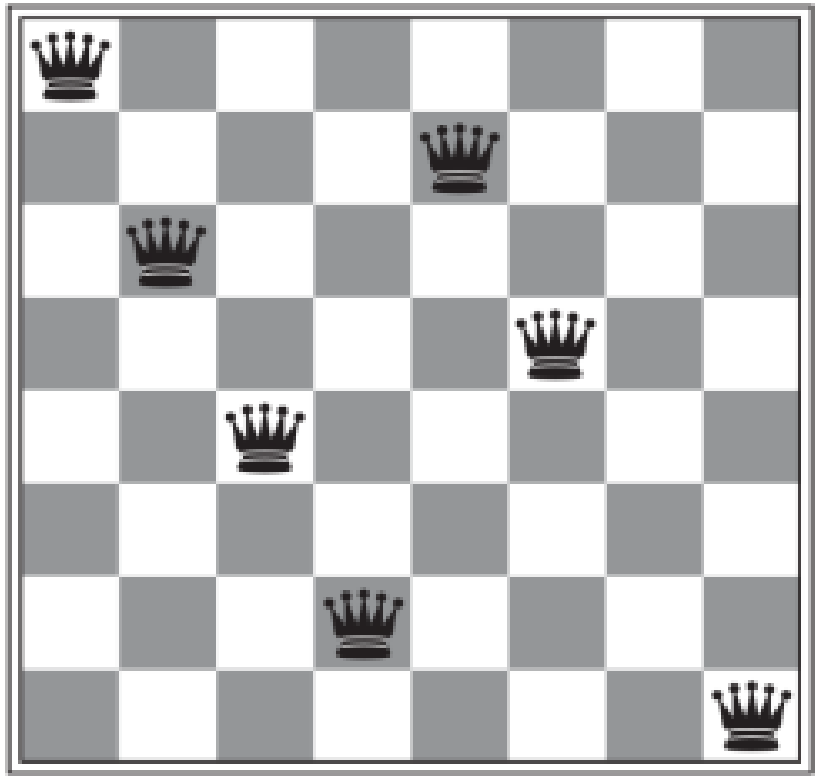


(© S. Russell & P. Norwig, AIMA)

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?
  - thêm một quân hậu vào bất kỳ ô trống nào
- 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
- Chi phí?
  - Không có



(© S. Russell & P. Norwig, AIMA)

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?
- Trạng thái xuất phát?
- Trạng thái mục tiêu?
- Chi phí?

Search Problems

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

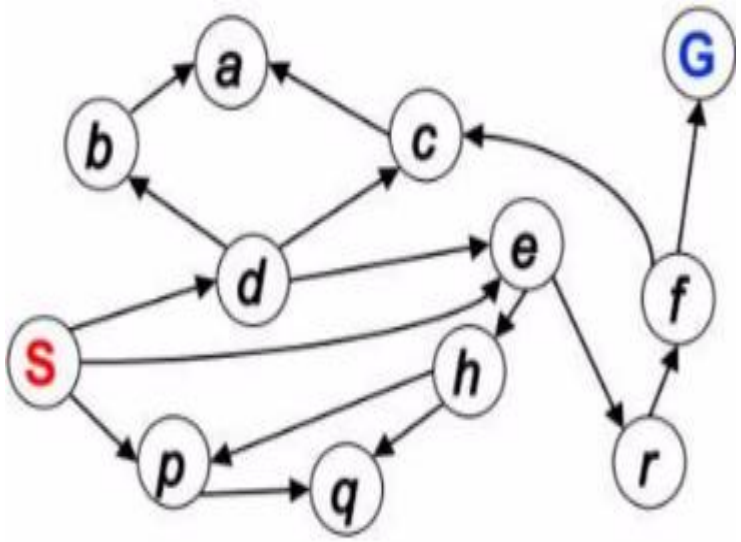
- Không gian trạng thái?
  - địa điểm, thời gian, giá vé cơ sở, chuyến bay nội địa/quốc tế, …
- Các hành động?
  - Đi bất kỳ chuyến bay nào từ vị trí hiện tại, ở bất kỳ hạng ghế nào, khởi hành sau giờ hiện tại, chừa đủ thời gian để trung chuyển trong sân bay nếu cần
- Trạng thái xuất phát?
  - được chỉ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

Search Problems

- **Làm sao để biểu diễn một search problem???**
  - Tree search
  - Graph search

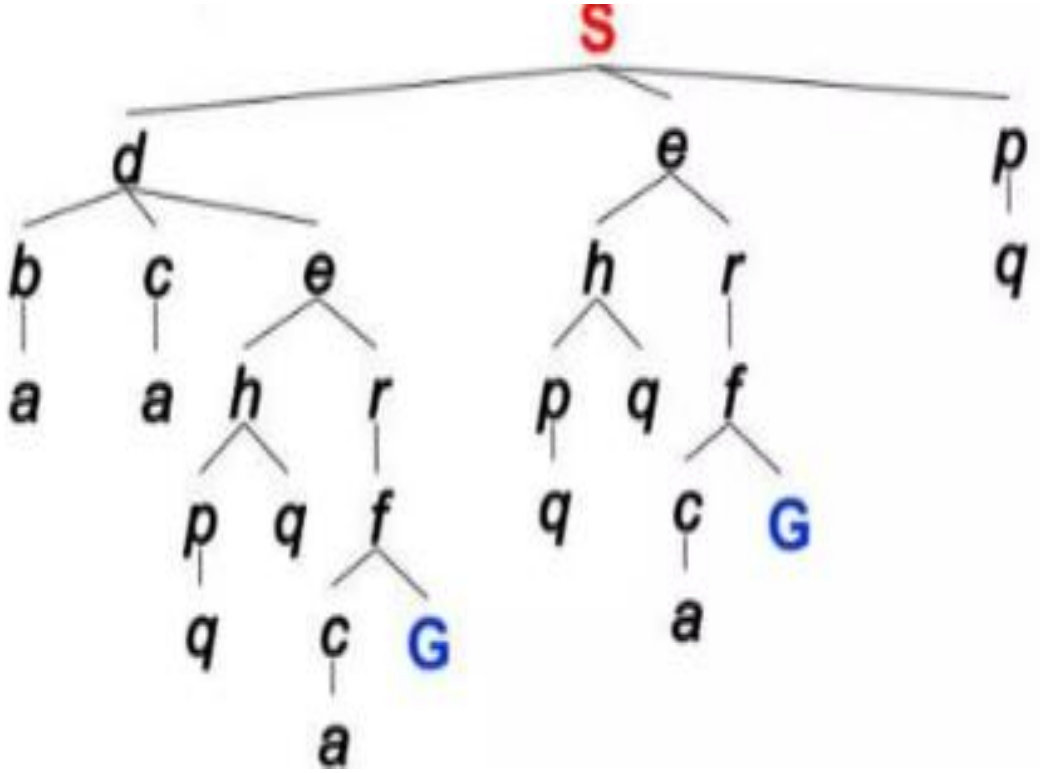
Đồ thị Không gian trạng thái?

- Đồ thị không gian trạng thái là một hiểu diễn toán học của một search problem
  - Mỗi node tương ứng với một cấu hình của môi trường hay trạng thái
  - Các cung biểu diễn trạng thái kế tiếp, mỗi cạnh tương ứng với một hành động
  - Trạng thái đích là tập các node đích (thường chỉ có một)
- Trong đồ thị không gian trạng thái, mỗi trạng thái chỉ xuất hiện duy nhất một lần. Và có thể chứa vòng lặp.
- **Thường chúng ta không xây dựng đầy đủ đồ thị trạng thái trên bộ nhớ vì rất lớn**



Cây Không gian trạng thái?

- Cây không gian trạng thái là một trường hợp đặc biệt của đồ thị dùng để biểu diễn một chuỗi các hành động tạo ra trạng thái của node
  - Mỗi node tương ứng với một cấu hình của môi trường hay trạng thái
  - Node trên cùng trong cây được gọi là node gốc; mọi hoạt động trên cây đều bắt đầu từ node này



Cây không gian trạng thái

Solution = sequence of actions = a search tree

o Nodes = Các trạng thái

o Edges (Branches) = Các hành động

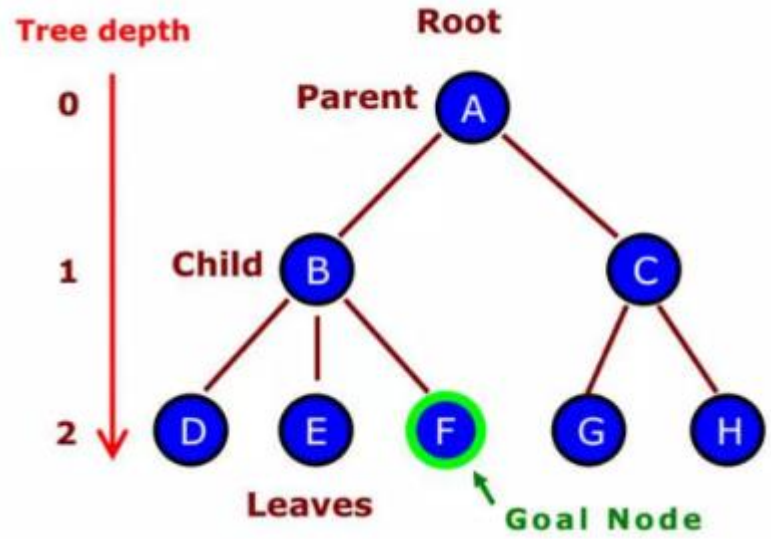
Một số thuật ngữ:

o Mở rộng nút

o Nút lá

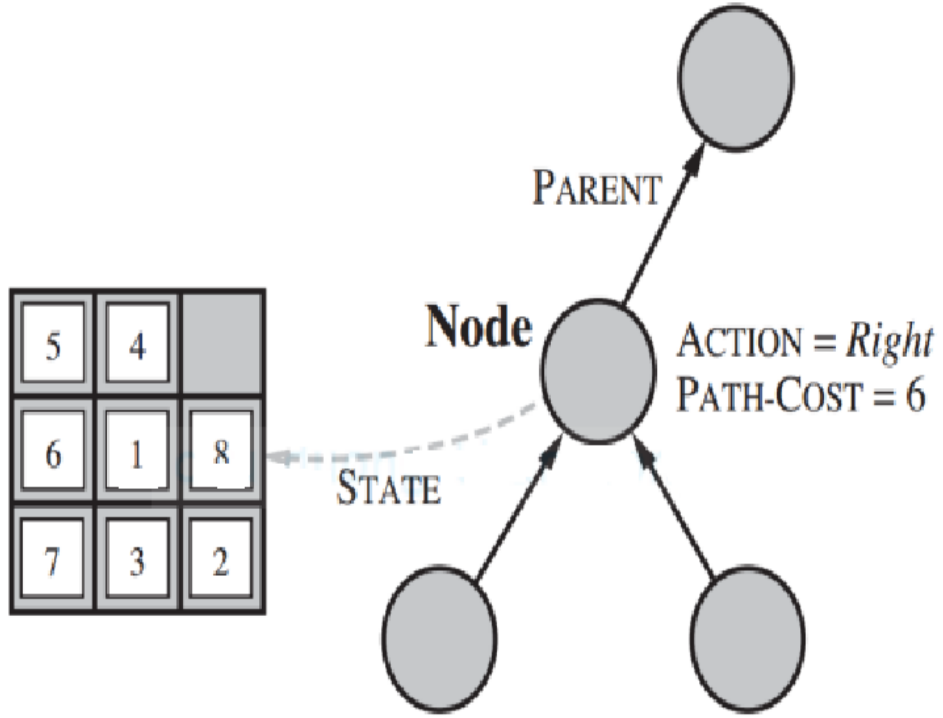
o Các nút biên (hay còn gọi là nút mở)

o Các nút đóng (tập đóng)



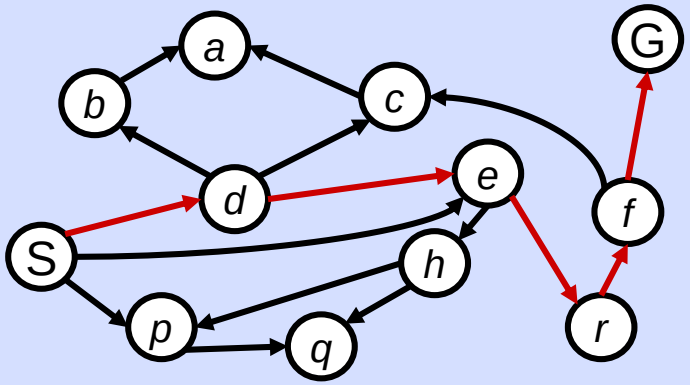
Cây không gian trạng thái

- Trạng thái là (biểu diễn của) một cấu hình vật lý
- Một nút là một cấu trúc dữ liệu tạo thành một phần của cây tìm kiếm bao gồm:
  - TRẠNG THÁI
  - CHA MẸ
  - HÀNH ĐỘNG
  - CHI PHÍ ĐƯỜNG DẪN

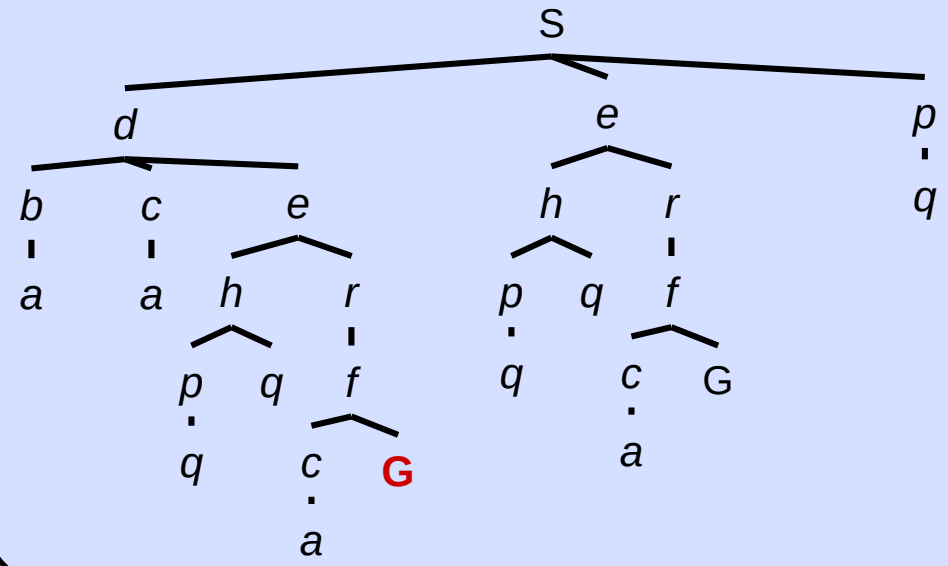


State Space Graphs vs. Search Trees

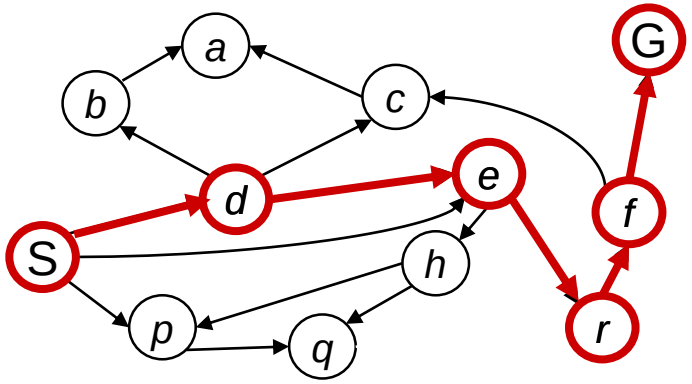
State Space Graph

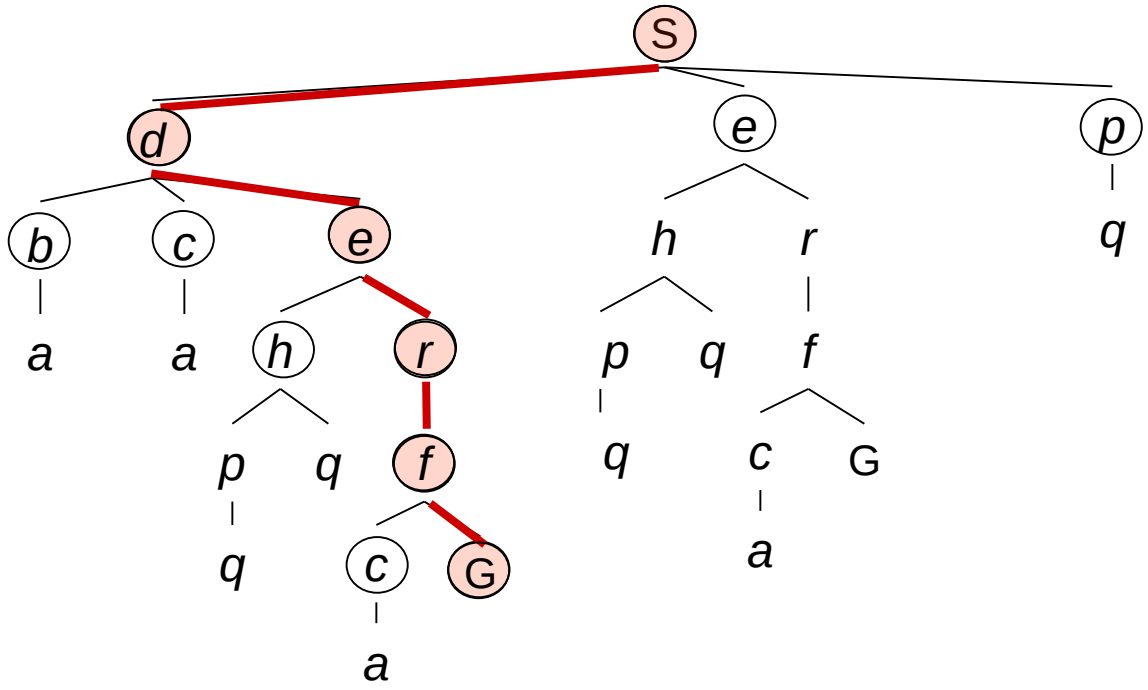


Search Tree



State Space Graphs vs. Search Trees





Các bài toán tìm kiếm trên đồ thị có thể được chuyển thành các bài toán tìm kiếm trên cây:

Thay thế mỗi liên kết (cạnh) vô hướng bằng 2 liên kết (cạnh) có hướng

Loại bỏ các vòng lặp tồn tại trong đồ thị (để tránh không duyệt 2 lần đối với một nút trong bất kỳ đường đi nào)

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

Thuật toán tìm kiếm tổng quát

3	1	6
5		8
2	7	4

Trạng thái xuất phát

	1	2
3	4	5
6	7	8

Trạng thái đích

- Xét Start state
- Kiểm tra start state = goal state?
- Nếu chưa:

+ áp dụng các chuyển động để sinh ra các trạng thái khác

+ xét trạng thái vừa sinh ra (trạng thái hiện thời):

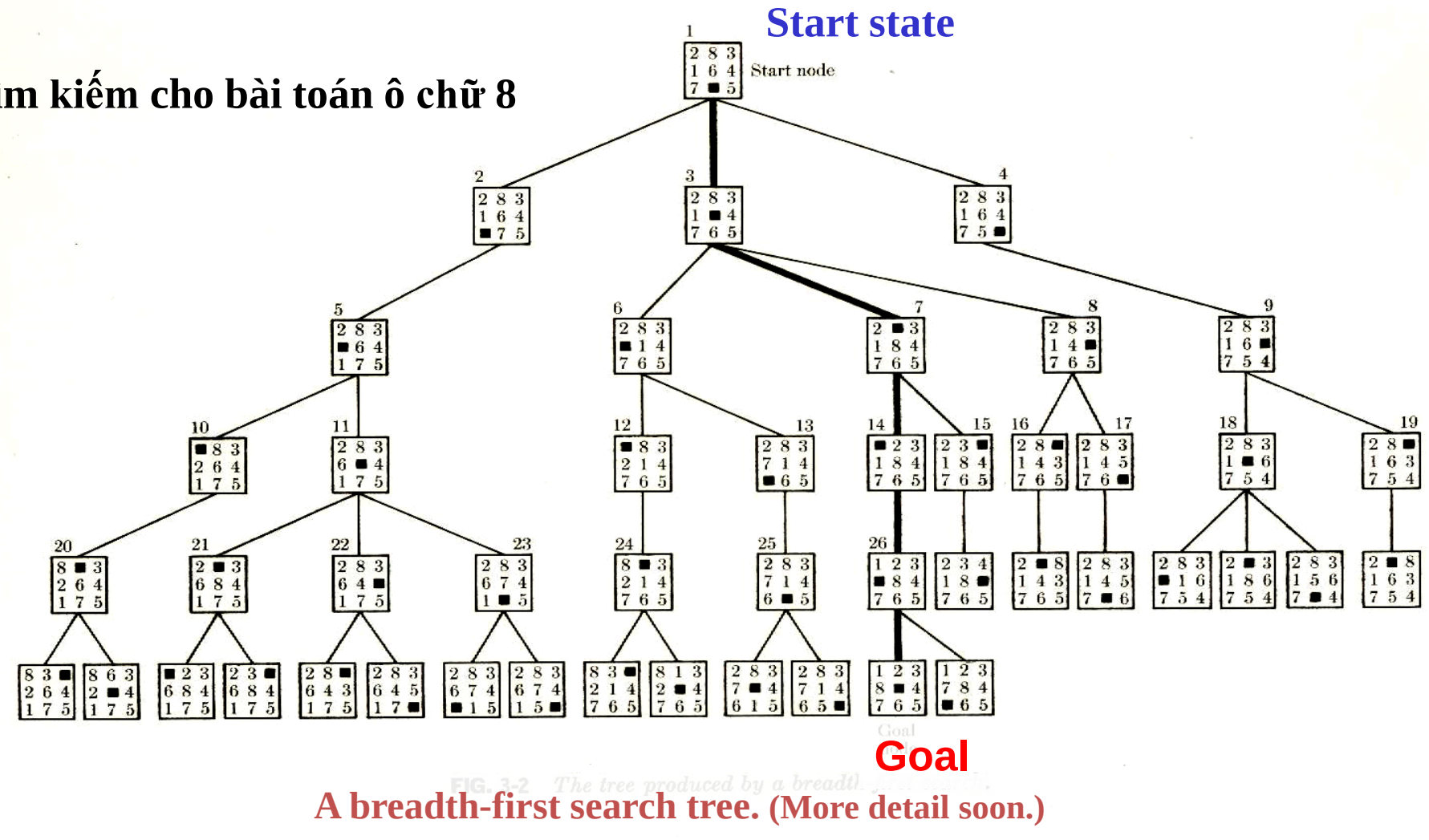
! Kiểm tra trạng thái hiện thời = goal state

! áp dụng các chuyển động để sinh ra các trạng thái khác

=> QT này kết thúc khi tìm được goal state hoặc không còn trạng thái để mở rộng

Search Trees

Ví dụ:
Một phần cây tìm kiếm cho bài toán ô chữ 8 số.



A breadth-first search tree. (More detail soon.)

Search Trees

- **Thuật toán tìm kiếm tổng quát:**

Search(Q, S, G, P) (Q: không gian trạng thái, S: trạng thái bắt đầu, G: đích, P: hành động)
Đầu vào: bài toán tìm kiếm với 4 thành phần như trên
Đầu ra: trạng thái đích

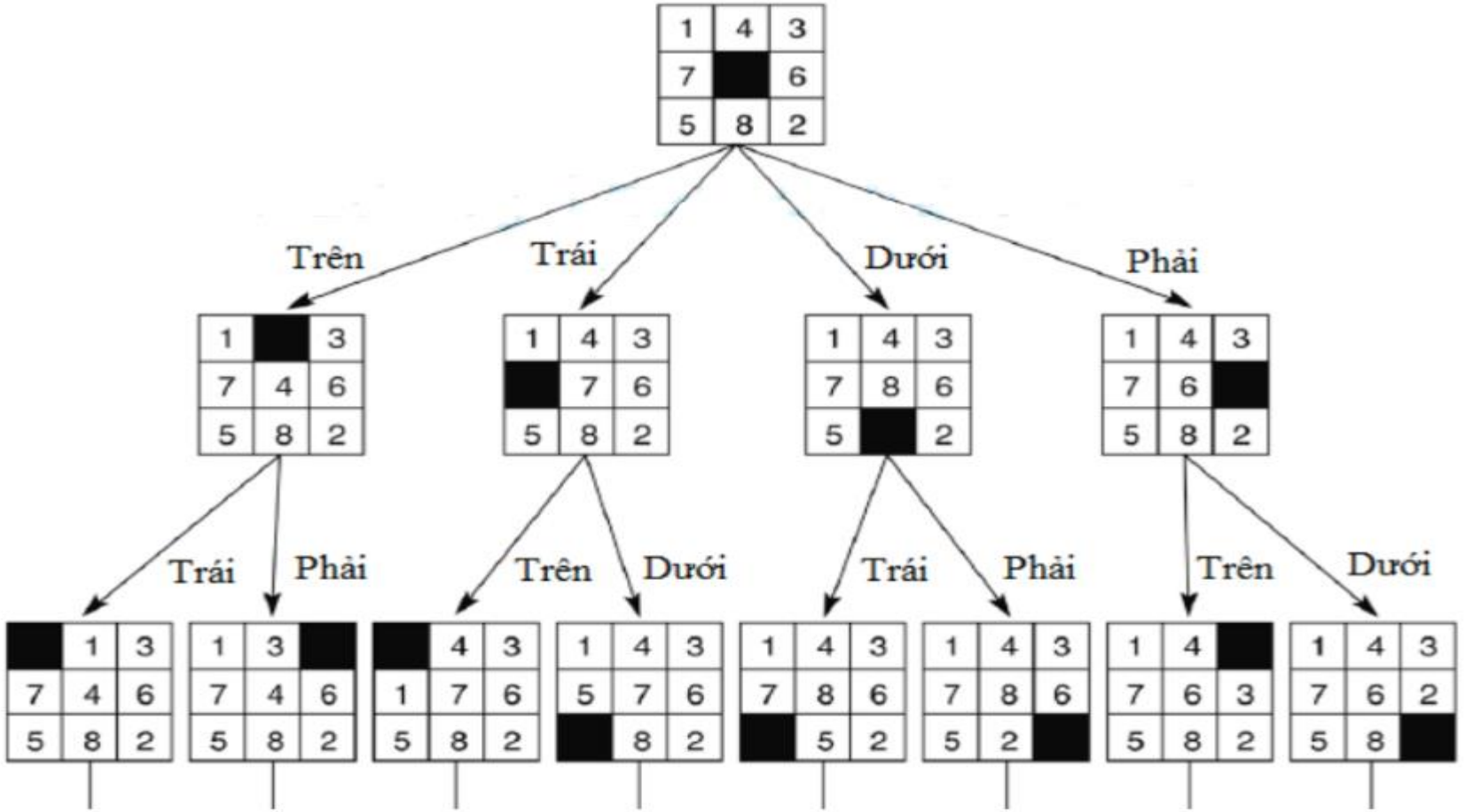
Khởi tạo: O ← S (O: tập các nút biên, bước này khởi tạo giá trị ban đầu cho O bằng S)
While(O ≠ Ø) do

1. Chọn nút n ∈ O và xóa n khỏi O
2. If n ∈ G, return (đường đi tới n P(n))
3. Thêm P(n) vào O

Return: Không có lời giải

Search Trees

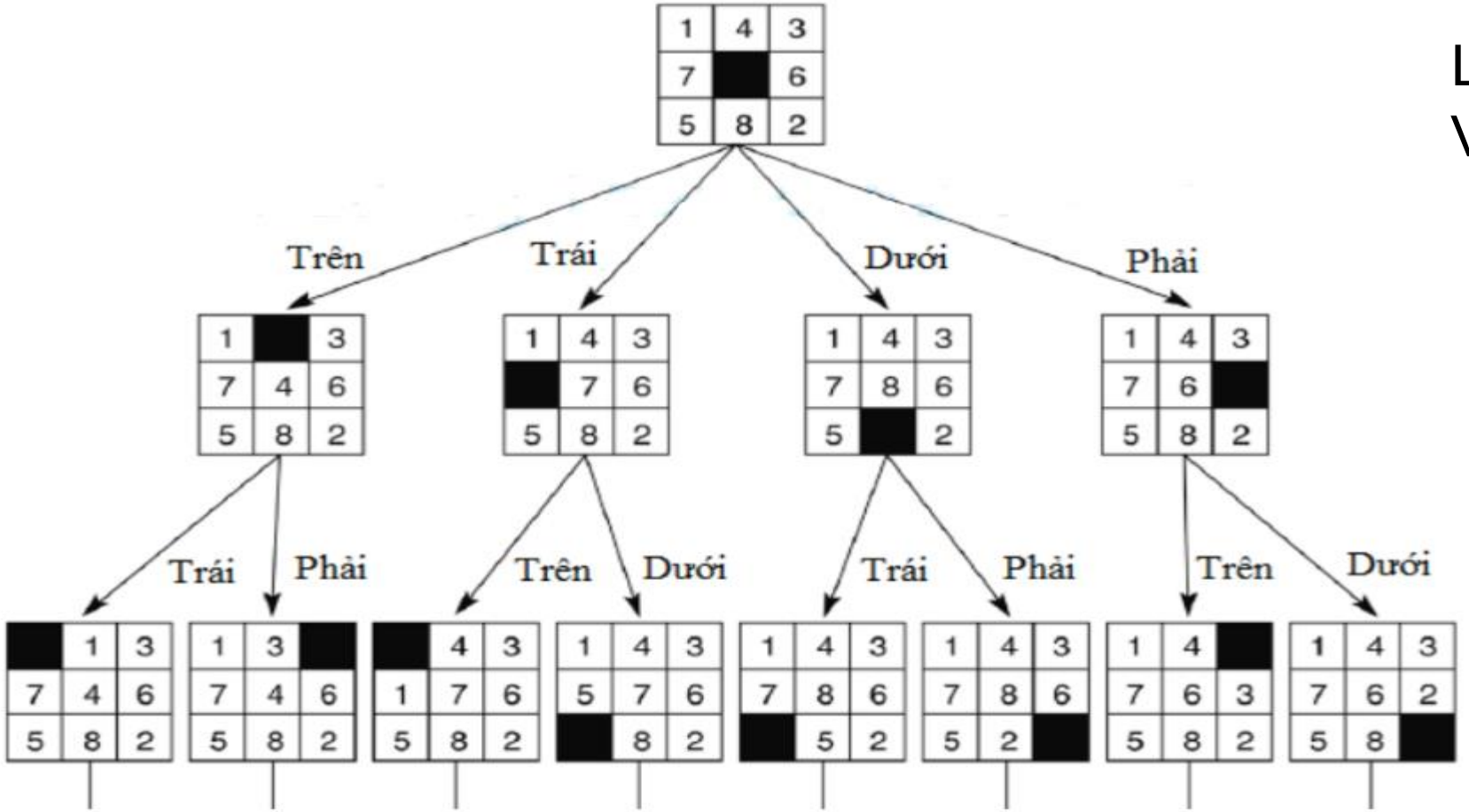
- **Thuật toán tìm kiếm tổng quát:**



Giả sử: từ trạng thái ngoài cùng bên trái thuộc lớp giữa =>quay về trạng thái xuất phát nếu sử dụng chuyển động “Dưới”. => Vòng lặp xuất hiện.

Search Trees

- **Thuật toán tìm kiếm tổng quát:**



Làm sao để tránh
Vòng lặp???

Search Trees

- **Thuật toán tìm kiếm tổng quát:**

Graph_Search(Q, S, G, P) (Q: tập trạng thái, S: trạng thái bắt đầu, G: đích, P: hành động)
Đầu vào: bài toán tìm kiếm với 4 thành phần như trên
Đầu ra: trạng thái đích
Khởi tạo: O ← S // O: tập các nút biên, bước này khởi tạo giá trị ban đầu cho O bằng S
D ← Ø // D là tập nút đóng, được khởi tạo bằng rỗng

While(O ≠ Ø) do
1. chọn nút n ∈ O và xóa n khỏi O
2. If n ∈ G, return (đường đi tới n)
3. Thêm n vào D 4. Thêm các nút thuộc P(n) vào O nếu nút đó không thuộc D và O
Return: Không có lời giả

Các tiêu chuẩn đánh giá một thuật toán tìm kiếm

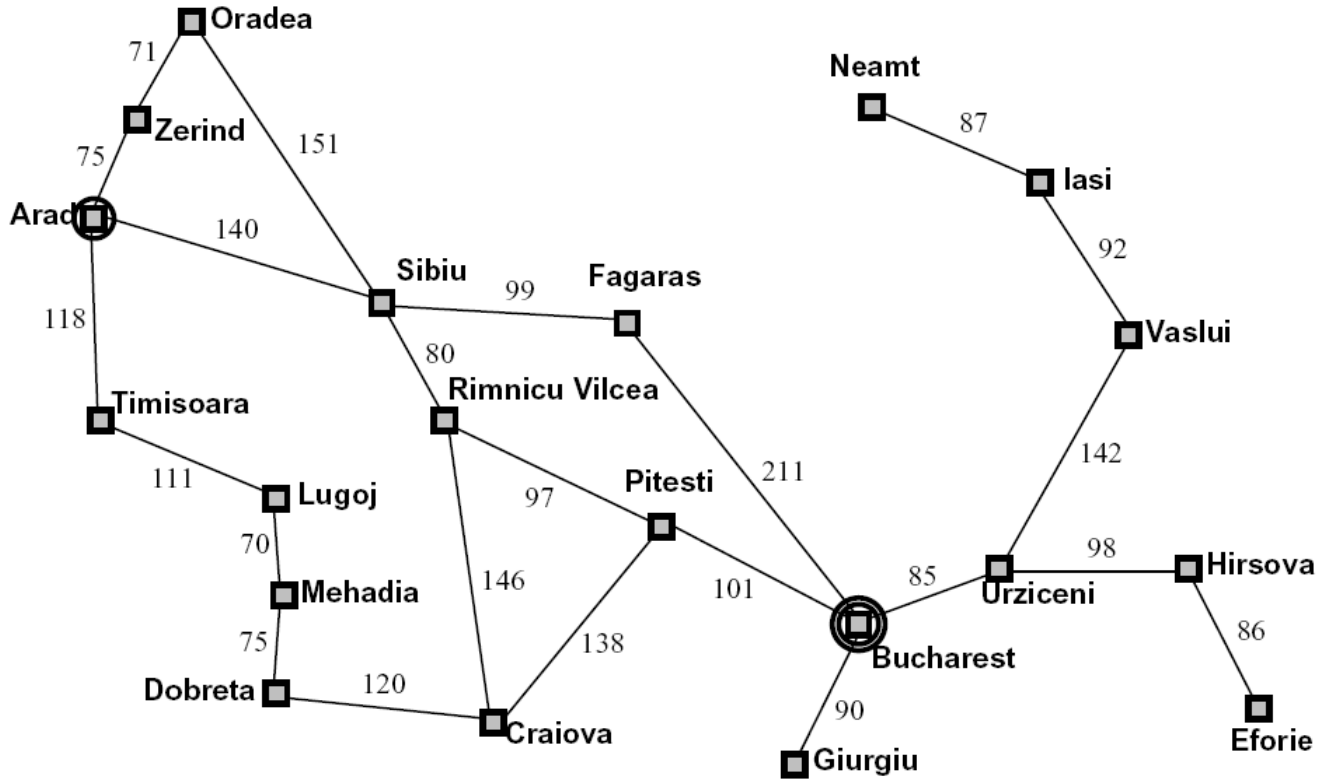
Bốn tiêu chuẩn để đánh giá các thuật toán tìm kiếm:

- Tính đầy đủ: Có đảm bảo tìm ra giải pháp nếu có không?
- Tính tối ưu: Đảm bảo tìm được đường đi có chi phí thấp nhất?
- Độ phức tạp tính toán?
- Yêu cầu bộ nhớ?

- Giả sử:
  - Số nhánh rẽ ở mỗi node là b
  - Độ sâu tối đa của cây là m
  - Các lời giải nằm ở các độ sâu khác nhau
- Khi đó, số lượng node trên cây là?
  - $1 + b + b^2 + \ldots. b^m = O(b^m)$

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

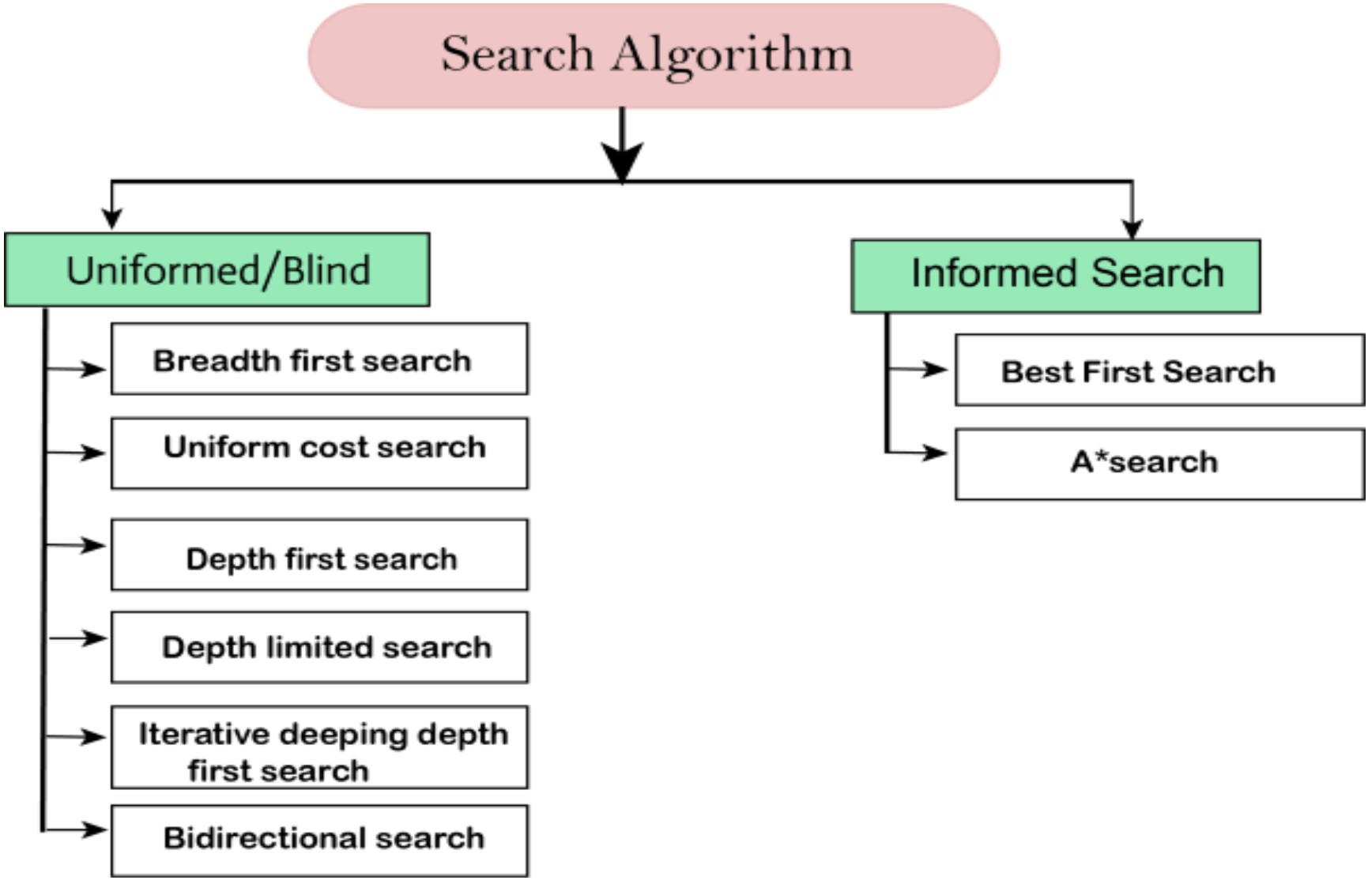
2	6	5
	8	7
4	3	1

trạng thái đích

1	2	3
4	5	6
7	8	

Các thuật toán tìm kiếm

- Search approaches



Uninformed search algorithms

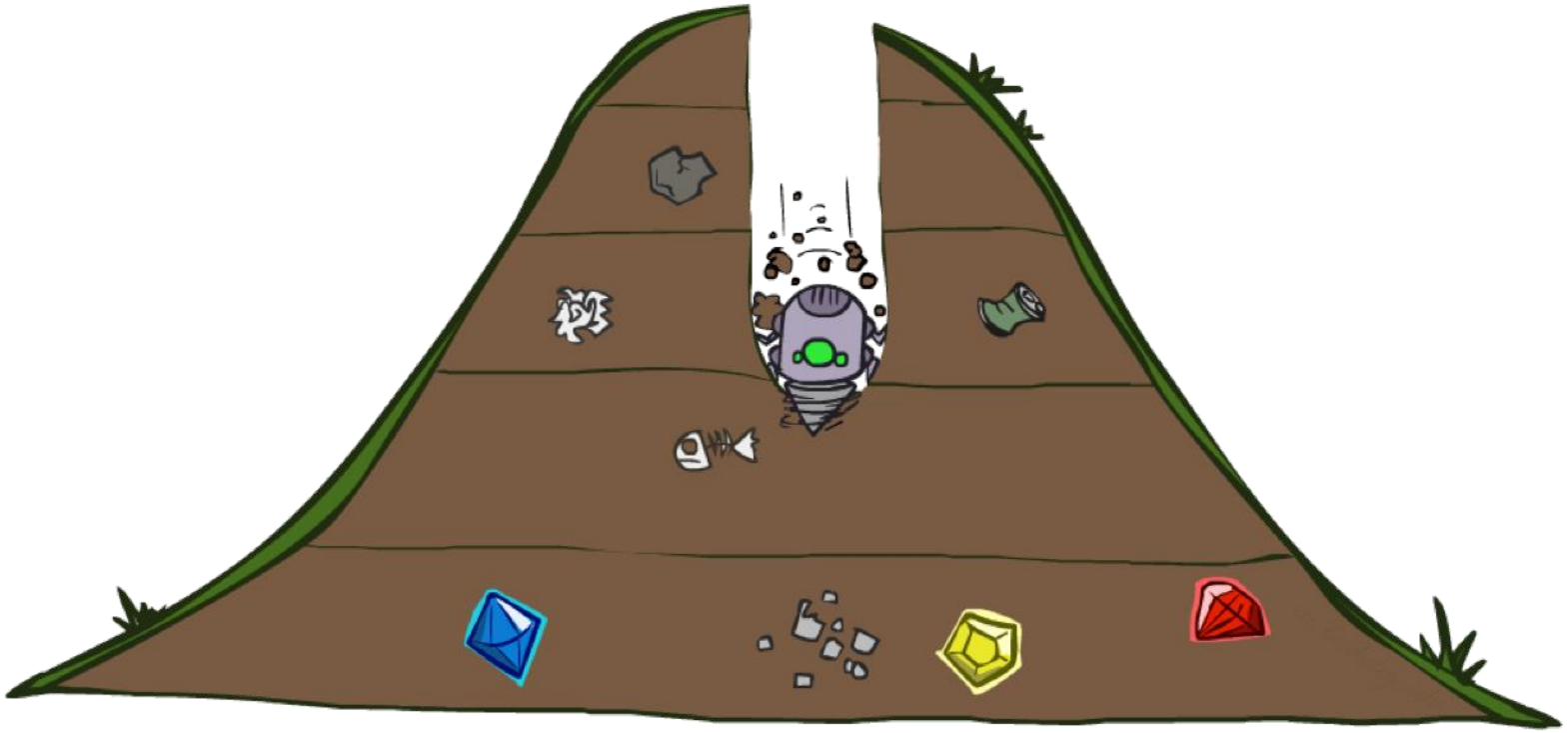
Breadth-first search

Depth-first search

Iterative Deepening

Uniform Cost Search

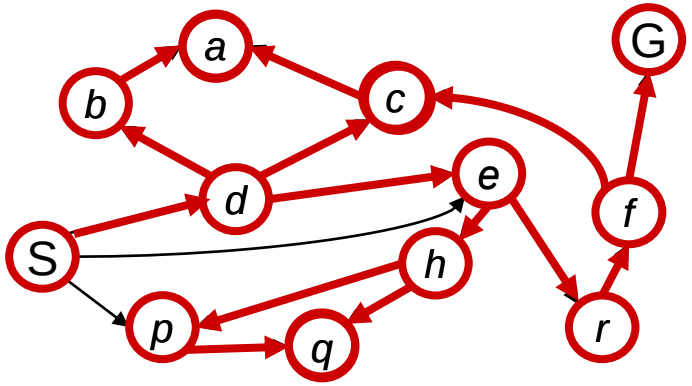
Depth-First Search

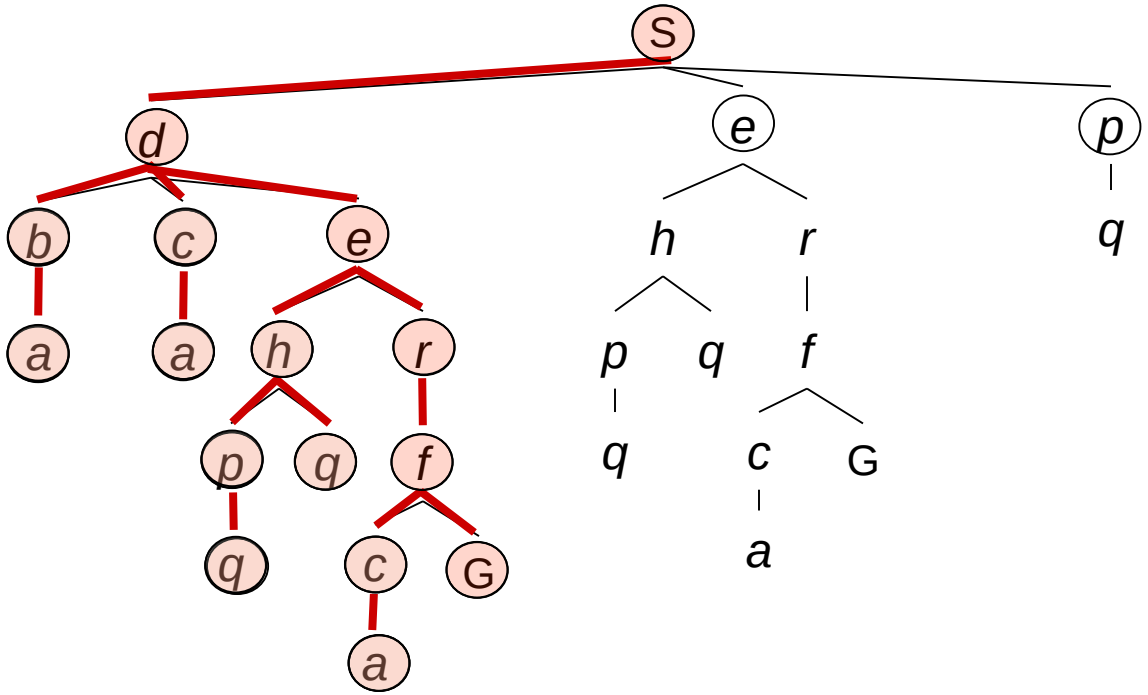


Depth-First Search

Phát triển các nút chưa xét theo chiều sâu – Các nút được xét theo thứ tự độ sâu giảm dần

Cài đặt: một cấu trúc kiểu ngăn xếp (LIFO)-các node mới được bổ sung vào đầu.





Depth-First Search

Algorithm - Depth-first search

- Put the root node on a stack;

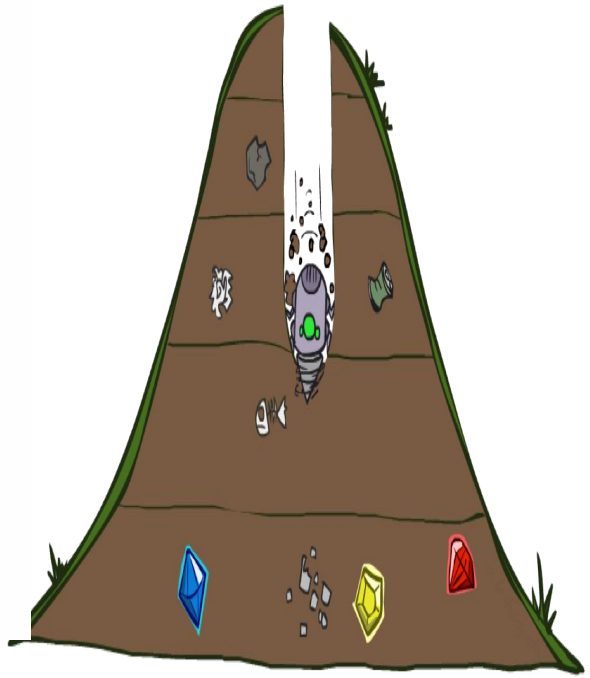
  while (stack is not empty)

    { remove a node from the stack;

      if (node is a goal node) return success;

      put all children of node onto the stack; }

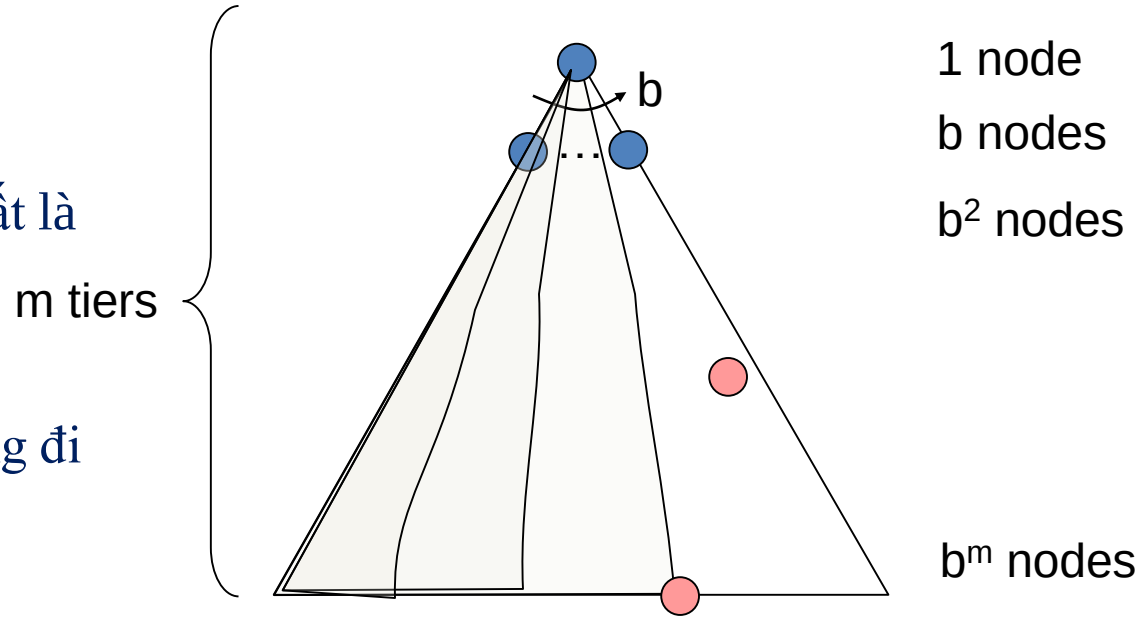
  return failure;



Depth-First Search

Các đặc trưng của DFS

- Cách mở rộng các node?
  - Từ bên trái nhất của cây trước.
  - Có thể phải duyệt toàn bộ cây!
  - Nếu m là xác định, độ phức tạp tính toán xấu nhất là $O(b^m)$
- Độ phức tạp không gian lưu trữ?
  - Chỉ lưu các node anh/em của các node trên đường đi đang xét, $O(b^m)$
- Nó có đầy đủ không?
  - Không – Thất bại (không tìm được lời giải) nếu không gian trạng thái có độ sâu vô hạn, hoặc nếu không gian trạng thái chứa các vòng lặp giữa các trạng thái
- Nó có tối ưu không?
  - Không, vì luôn tìm thấy lời giải "trái nhất", bất chấp độ sâu hay chi phí



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

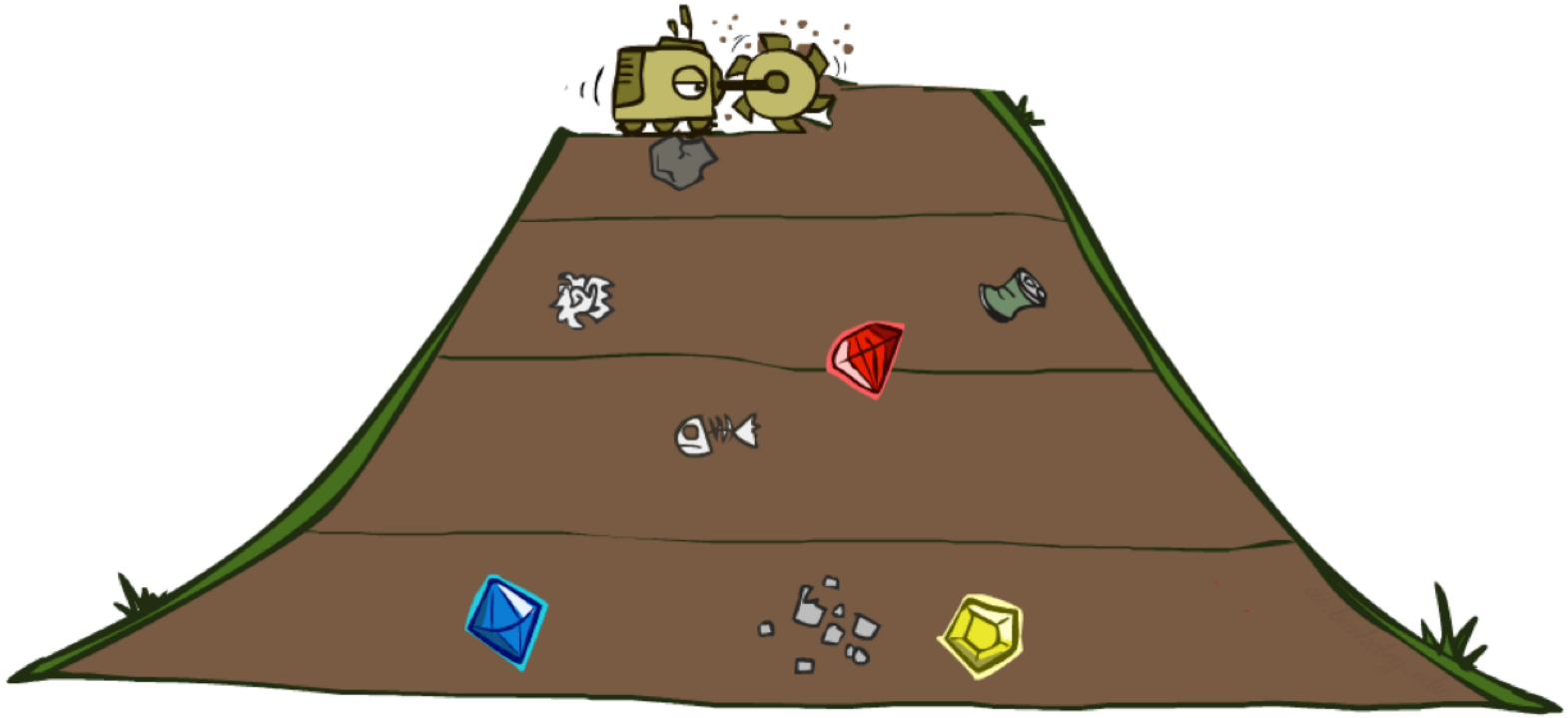
trạng thái xuất phát

2	6	5
	8	7
4	3	1

trạng thái đích

1	2	3
4	5	6
7	8	

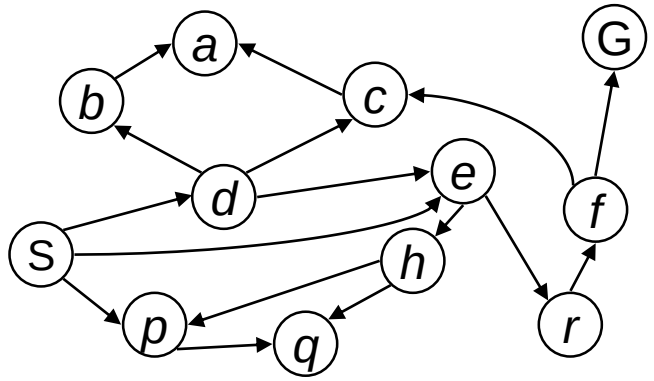
Breadth-First Search

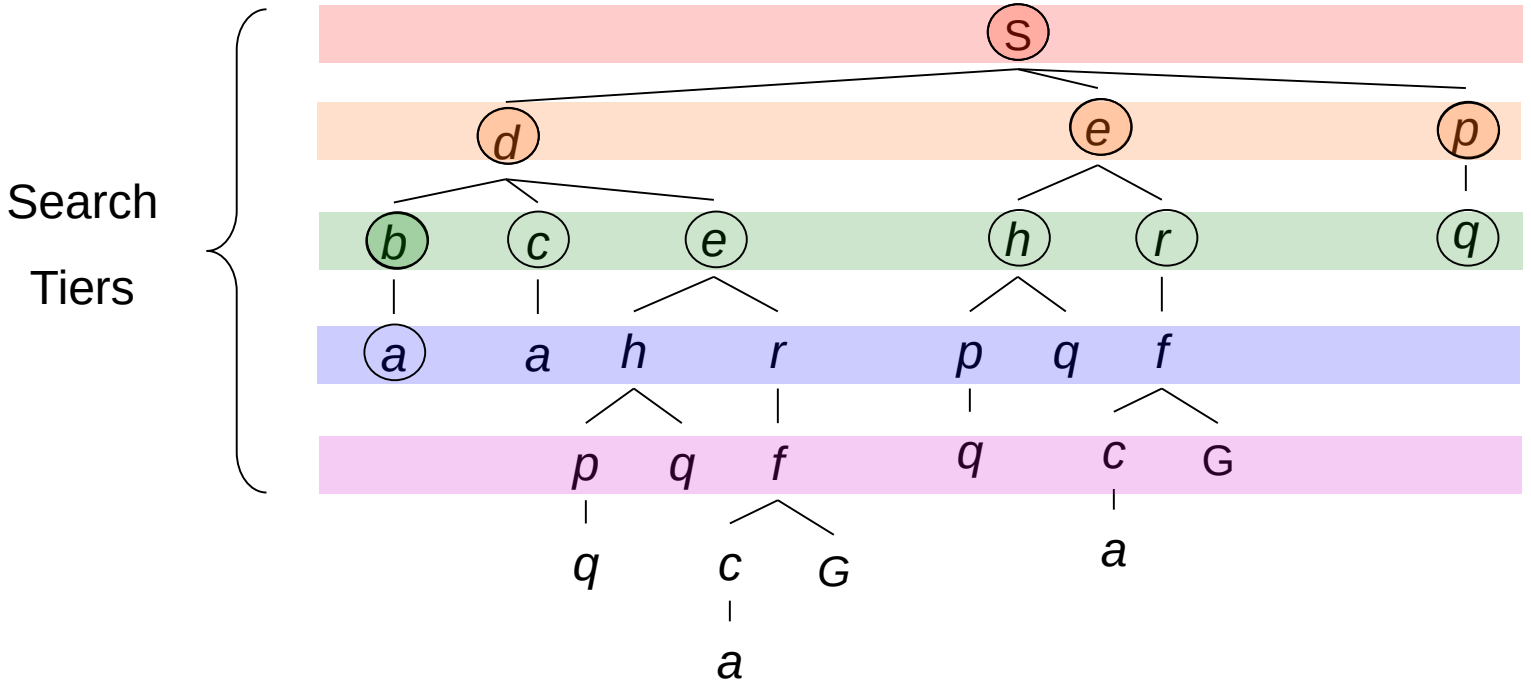


Breadth-First Search

Chiến lược: mở rộng nút cạn nhất trước

Implementation: một cấu trúc kiểu hàng đợi(FIFO – các nút mới được bổ sung vào cuối





Breadth-First Search

Algorithm - Breadth-first search

- Put the root node on a queue;

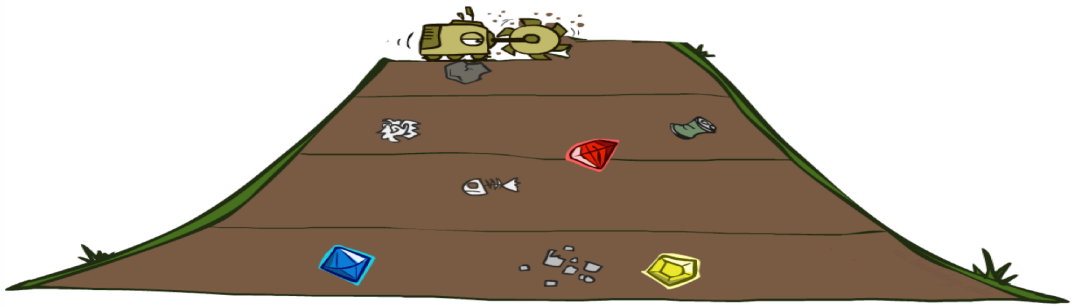
  while (queue is not empty)

    { remove a node from the queue;

      if (node is a goal node) return success;

      put all children of node onto the queue; }

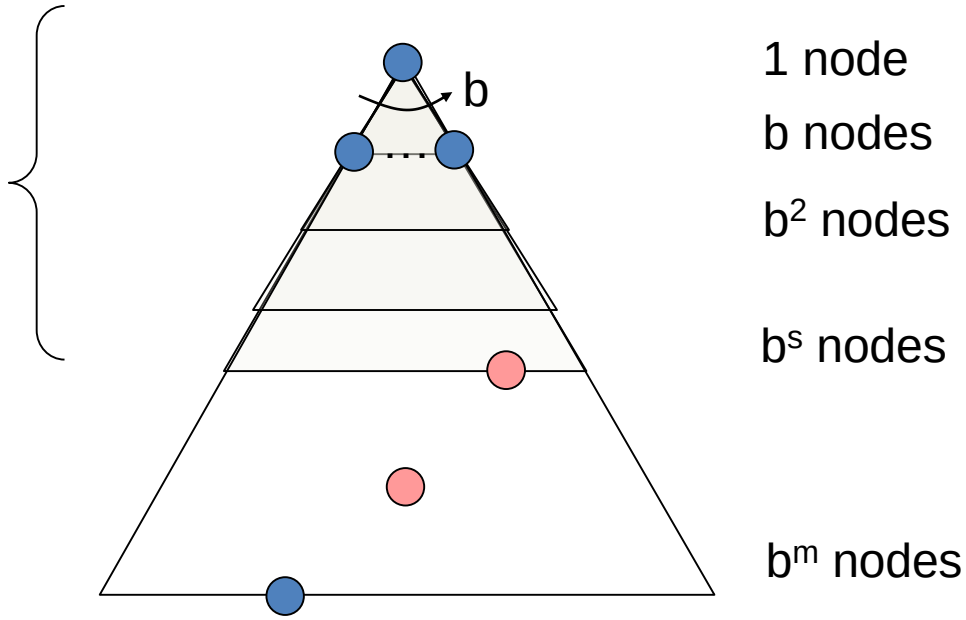
  return failure;



Breadth-First Search

Các đặc trưng của BFS

- Cách mở rộng các node?
  - Xử lý tất cả các node ở phía trên lời giải "cạn nhất".
  - Gọi độ sâu của lời giải "cạn nhất" là s, thời gian tìm kiếm là $O(b^s)$
- Dộ phức tạp không gian lưu trữ?
  - Lưu tất cả các node ở tầng cuối cùng, $O(b^s)$
- Nó có đầy đủ không?
  - Có, vì s là xác định nếu tồn tại lời giải
- Nó có tối ưu không?
  - Chỉ khi tất cả các chi phí đều bằng nhau



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	

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

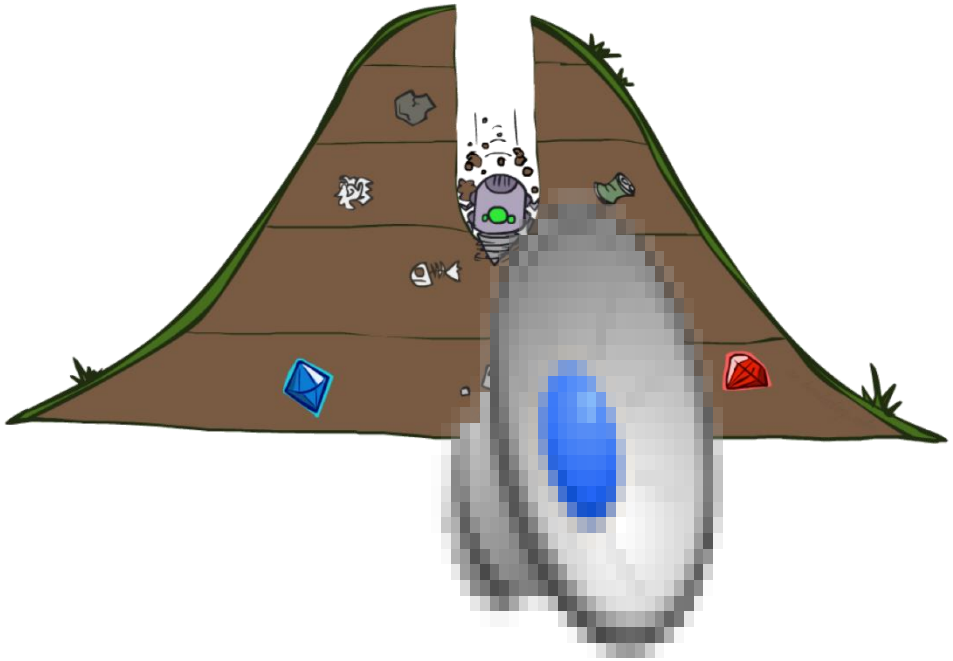
trạng thái xuất phát

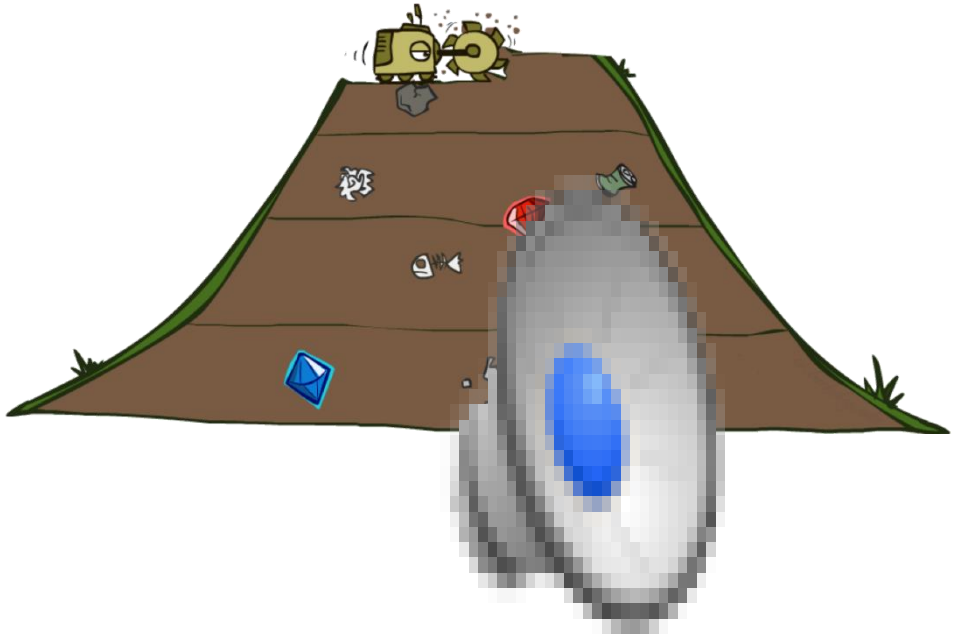
2	6	5
	8	7
4	3	1

trạng thái đích

1	2	3
4	5	6
7	8	

DFS vs BFS





Video of
Demo Maze
Water
DFS/BFS

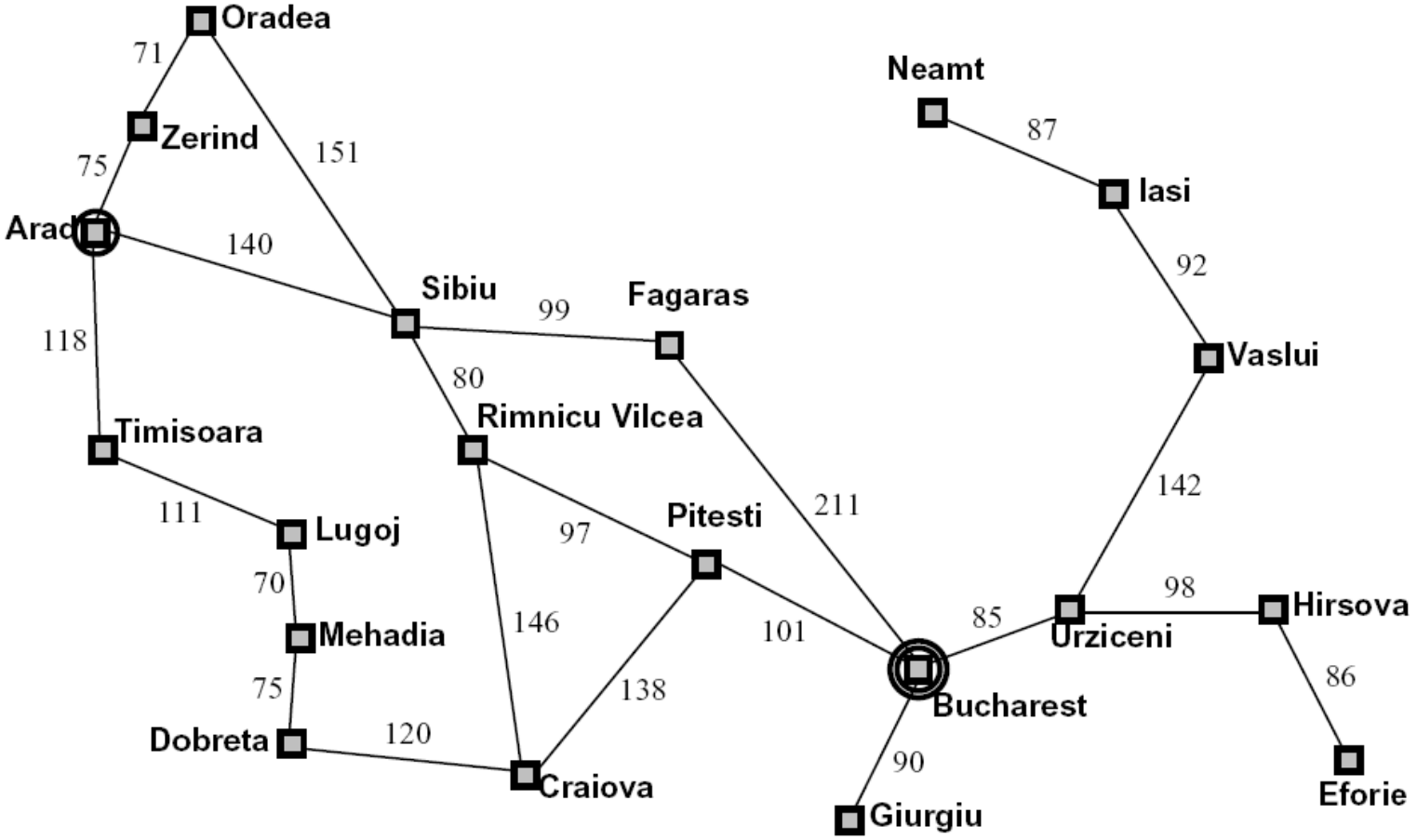
- When will BFS outperform DFS?
- When will DFS outperform BFS?

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

N puzzle solvable

```python
def inversions(state,size):
    #Computing the sum of inversion
    inversion_sum = 0
    L=size[0]*size[1]
    for i in range(L-1):
        for j in range(i+1,L):
            if ((state[i] > 0) and (state[j] >0)
              and (state[i] > state[j])):
                inversion_sum += 1
    return inversion_sum


def is_solvable(state,size):
    #"Checks solvable or not"
    N=Board.inversions(state,size=size)
    ii=state.index(0)
    h=(ii //size[1])
    print("N=",N,", 0=>",ii,", h=",h)
    if (size[0]*size[1])%2==1:
        if (N%2==0):
            return True;
        return False;
    elif (N%2==0 and h%2 == 0) or (N%2==1 and h%2==1):
        return True;
```

```python
def is_solvable(state,size):  #"Checks solvable or not"
    N=Board.inversions(state,size=size)
    ii=state.index(0)
    h=(ii //size[1])+1
    print("N=",N,", 0=>",ii,", h=",h,".  SIZE=",size)
    #if (size[0]%2==1)or(size[1]%2==1):
    if ( N%2==0 and ((h%2==1 and size[0]*size[1]%2==0)
                     or(h%2==0 and size[1]*size[1]%2==1))):
        return True;
    return False
```

Thanks for your attention!

Q&A