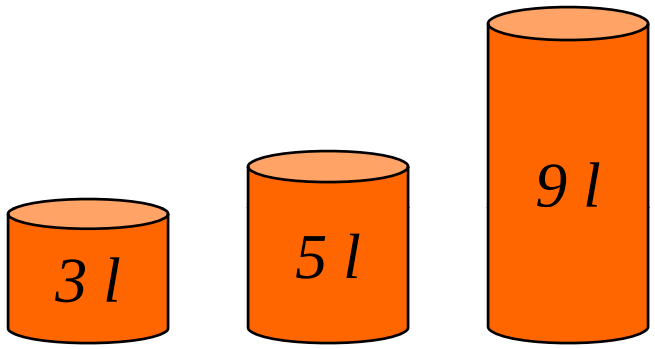
Outline: Giải quyết bài toán và tìm kiếm
(Problem solving and search, problem solving agents)

- **Hình thành bài toán (Problem formulation)**
  - Xác định các thành phần của bài toán (problem formulation)
  - Lược đồ chung để giải bài toán (general-solving procedure)
  - Đánh giá một giải thuật tìm kiếm
- **Uninformed (blink) search (Tìm không có thông tin phản hồi, hoặc tìm kiếm mù)**
  - Search strategies: depth-first, breadth-first (Các chiến lược tìm kiếm: theo chiều sâu, theo chiều rộng)
- **Informed search (Tìm kiếm dựa trên các hàm đánh giá, heuristics)**
  - Search strategies: **best-first search**
  - Heuristic functions (Các hàm heuristic)
    - ✓ Uniform search
    - ✓ Greedy search
    - ✓ A* search

Example: Measuring problem!

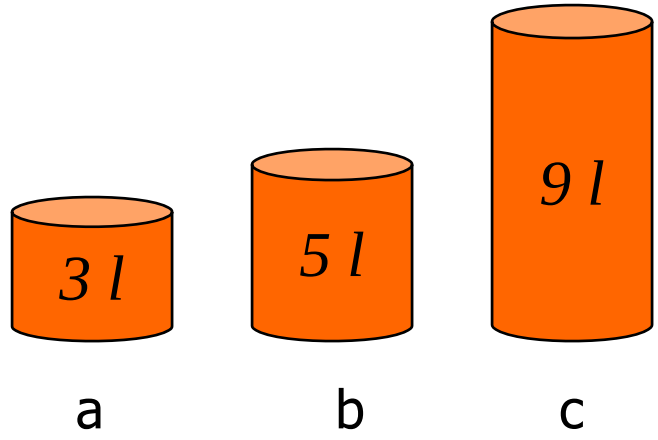


Problem: Using these three buckets,
measure 7 liters of water.

Example: Measuring problem!

- **(one possible) Solution:**

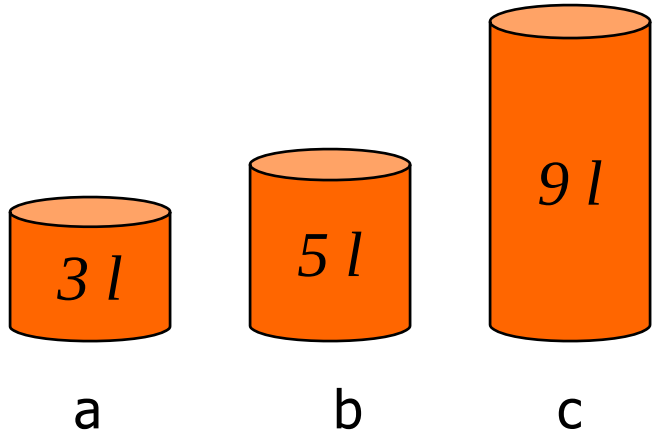
a	b	c	
0	0	0	start



Example: Measuring problem!

- **(one possible) Solution:**

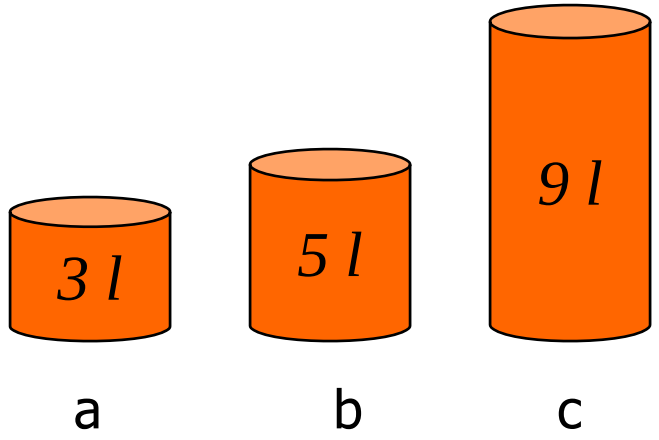
a	b	c	
0	0	0	start
3	0	0	



Example: Measuring problem!

- **(one possible) Solution:**

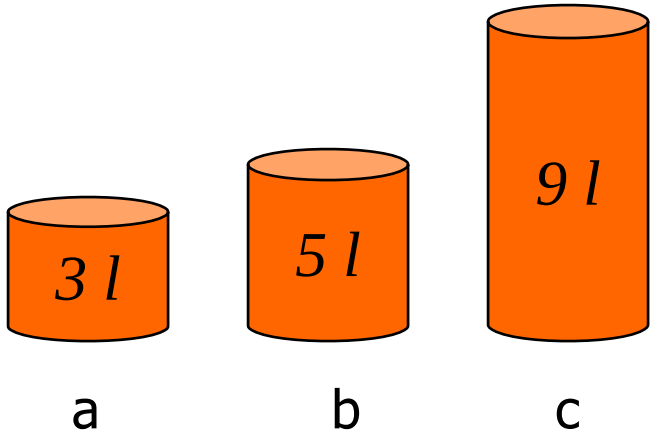
a	b	c	
0	0	0	start
3	0	0	
0	0	3	



Example: Measuring problem!

- **(one possible) Solution:**

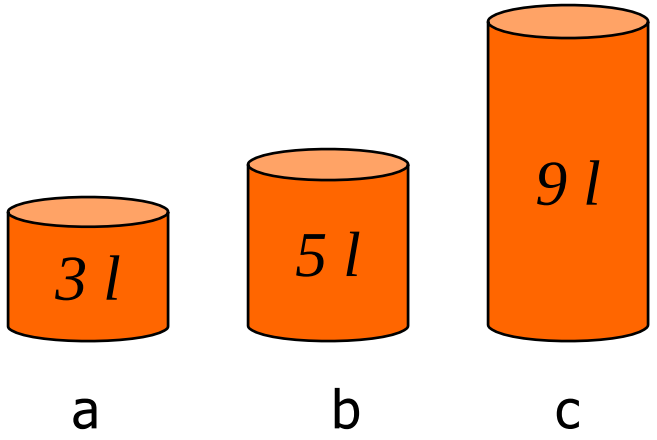
a	b	c	
0	0	0	start
3	0	0	
0	0	3	
3	0	3	



Example: Measuring problem!

- **(one possible) Solution:**

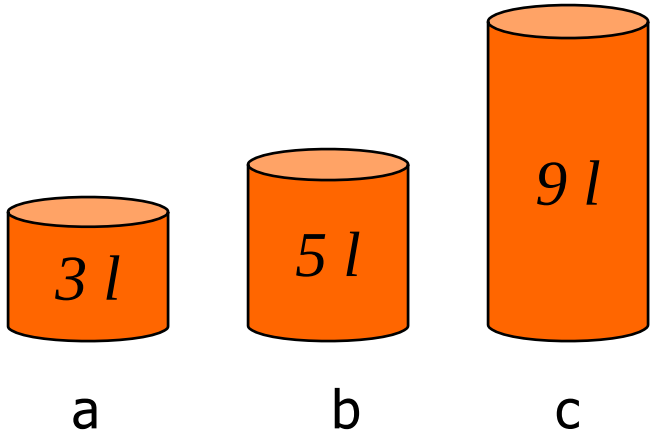
a	b	c	
0	0	0	start
3	0	0	
0	0	3	
3	0	3	
0	0	6	



Example: Measuring problem!

- **(one possible) Solution:**

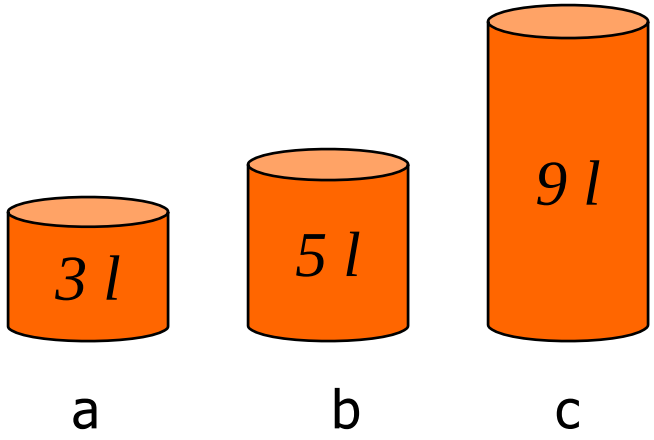
a	b	c	
0	0	0	start
3	0	0	
0	0	3	
3	0	3	
0	0	6	
3	0	6	



Example: Measuring problem!

- **(one possible) Solution:**

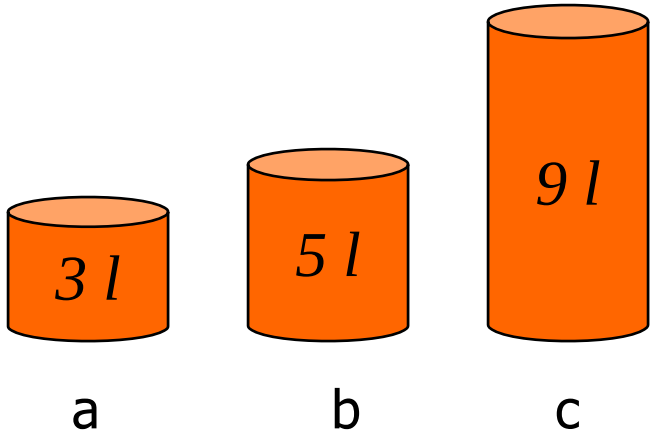
a	b	c	
0	0	0	start
3	0	0	
0	0	3	
3	0	3	
0	0	6	
3	0	6	
0	3	6	



Example: Measuring problem!

- **(one possible) Solution:**

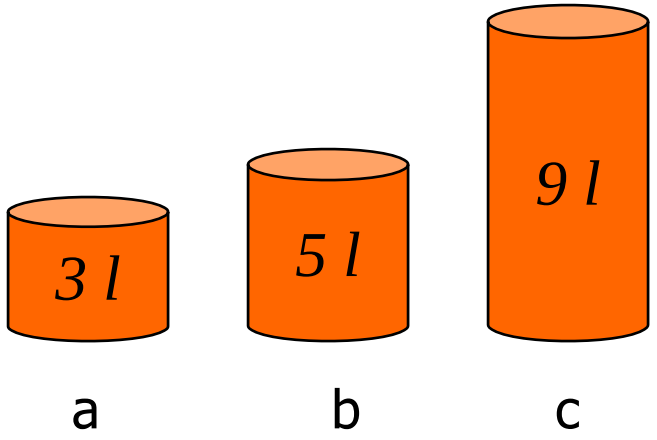
a	b	c	
0	0	0	start
3	0	0	
0	0	3	
3	0	3	
0	0	6	
3	0	6	
0	3	6	
3	3	6	



Example: Measuring problem!

- **(one possible) Solution:**

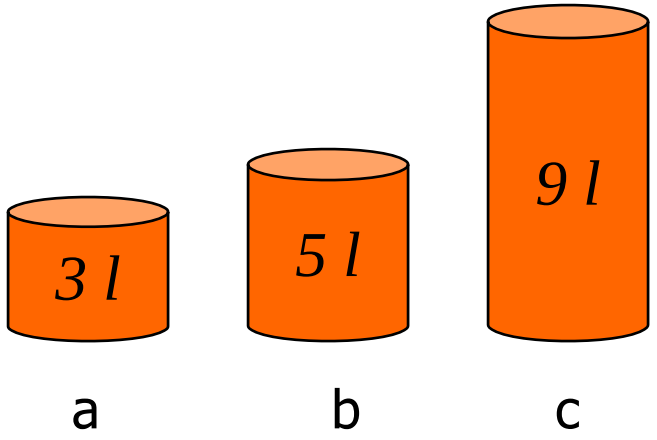
a	b	c	
0	0	0	start
3	0	0	
0	0	3	
3	0	3	
0	0	6	
3	0	6	
0	3	6	
3	3	6	
1	5	6	



Example: Measuring problem!

- **(one possible) Solution:**

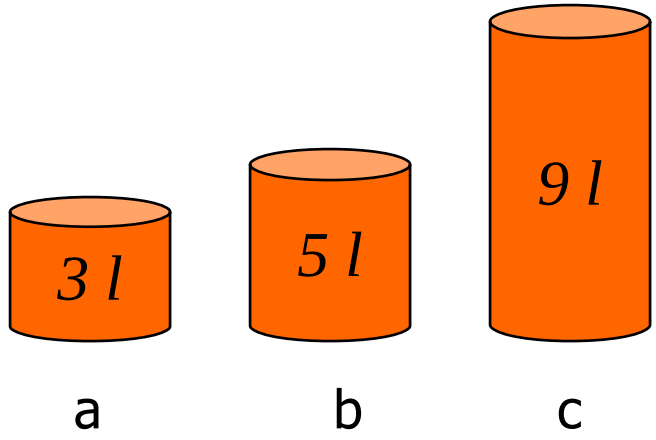
a	b	c	
0	0	0	start
3	0	0	
0	0	3	
3	0	3	
0	0	6	
3	0	6	
0	3	6	
3	3	6	
1	5	6	
0	5	**7**	**goal**



Example: Measuring problem!

- **Another Solution:**

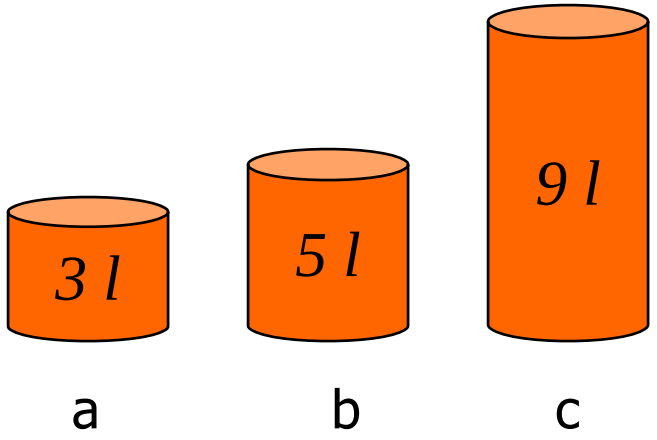
a	b	c	
0	0	0	start
0	5	0	



Example: Measuring problem!

- **Another Solution:**

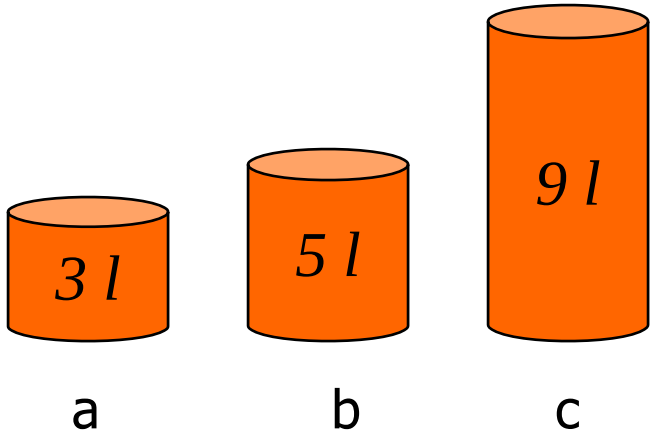
a	b	c	
0	0	0	start
0	5	0	
3	2	0	



Example: Measuring problem!

- **Another Solution:**

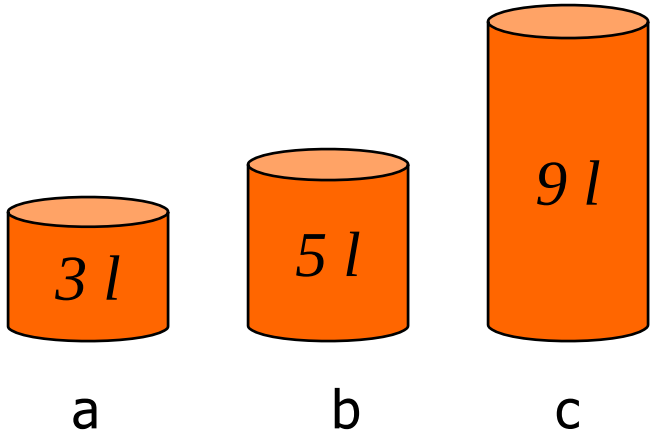
a	b	c	
0	0	0	start
0	5	0	
3	2	0	
3	0	2	



Example: Measuring problem!

- **Another Solution:**

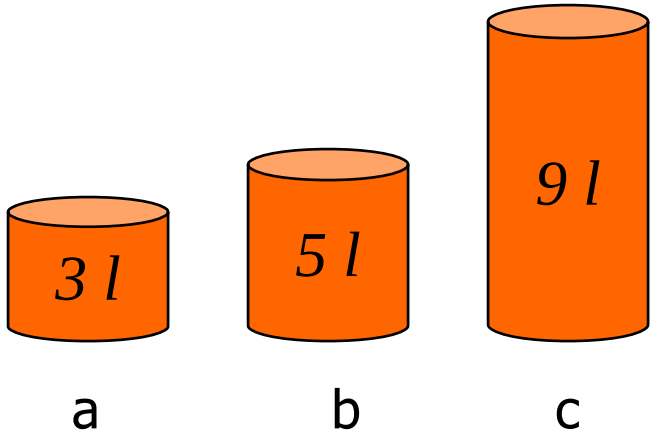
a	b	c	
0	0	0	start
0	5	0	
3	2	0	
3	0	2	
3	5	2	



Example: Measuring problem!

- **Another Solution:**

a	b	c	
0	0	0	start
0	5	0	
3	2	0	
3	0	2	
3	5	2	
3	**0**	**7**	**goal**



Which solution do we prefer? Nghiệm nào tốt hơn?

❑ **Solution 1:**

a	b	c	
0	0	0	start
3	0	0	
0	0	3	
3	0	3	
0	0	6	
3	0	6	
0	3	6	
3	3	6	
1	5	6	
0	5	**7**	**goal**

❑ **Solution 2:**

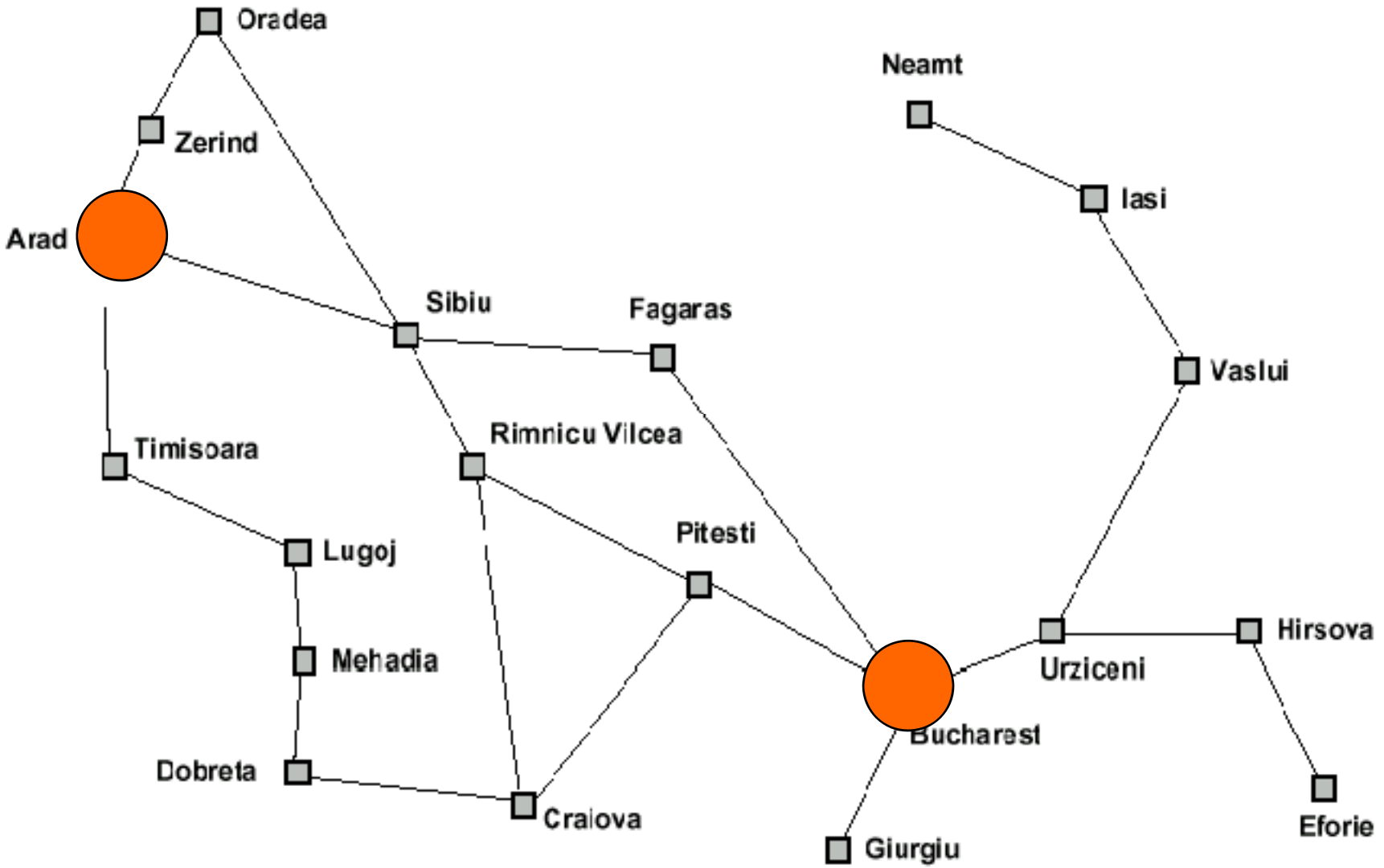
a	b	c	
0	0	0	start
0	5	0	
3	2	0	
3	0	2	
3	5	2	
3	**0**	**7**	**goal**

Problem formulation (Example: Buckets)

Measure 7 liters of water using a 3-liter, a 5-liter, and a 9-liter buckets.

- **Formulate problem (Xác định các thành phần của bài toán):**
  - *Initial state (trạng thái đầu)*: (0,0,0)
  - *Goal states (các trạng thái đích)*: (x,y,7)
  - *Operators/actions (Các thao tác)*: Fill bucket from source, empty bucket
  - *Cost (chi phí)*: 1 for each operator
- **Find solution (tìm nghiệm):** tìm dãy các thao tác (operators) chuyển từ trạng thái đầu đến trạng thái đích (sequence of operators that bring you from initial state to the goal state)

Example 2: Traveling from Arad To Bucharest



Problem formulation (Example 2: Romania)

- In Romania, on vacation. Currently in Arad.
- **Formulate problem:**
  - Initial states: at Arad
  - Goal state: at Bucharest
  - operators: drive between cities
  - Cost: distance between cities
- **Find solution:**
  - sequence of cities, such that total driving distance is minimized.

Problem formulation (xác định bài toán)

A *problem* is defined by four items:

initial state e.g., "at Arad"

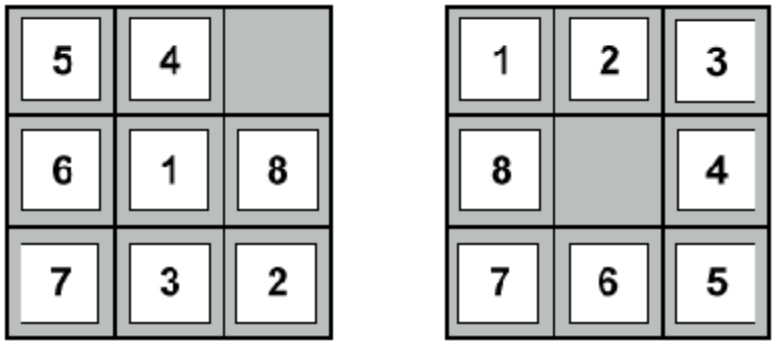
operators (or *successor function* $S(x)$)
e.g., Arad $\rightarrow$ Zerind Arad $\rightarrow$ Sibiu etc.

goal test, can be
explicit, e.g., $x =$ "at Bucharest"
implicit, e.g., $NoDirt(x)$

path cost (additive)
e.g., sum of distances, number of operators executed, etc.

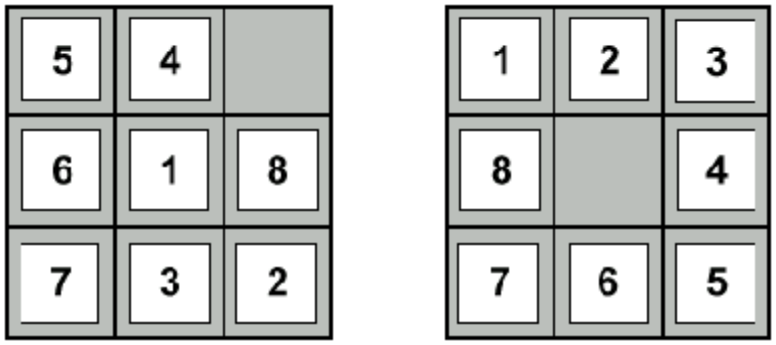
A *solution* is a sequence of operators
leading from the initial state to a goal state

Problem formulation (example 3: 8-puzzle)



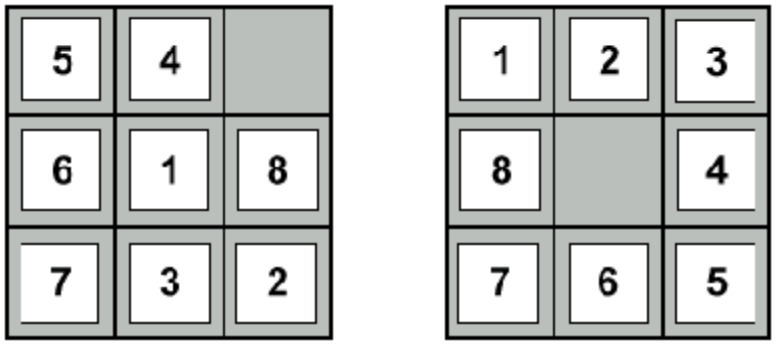
- State:
- Operators:
- Goal test:
- Path cost:

Problem formulation (example 3: 8-puzzle)



- State: integer location of tiles
- Operators: moving blank left, right, up, down
- Goal test: does state match goal state?
- Path cost: 1 per move

Problem solution: searching (example: 8-puzzle)



- Why search algorithms?
  - 8-puzzle has 362,880 states (=9x8x7x…1)
  - 15-puzzle has about 10^12 states (=16x15x14x…x1)
  - 24-puzzle has about 10^25 states
- So, we need a principled way to look for a solution in these huge search spaces…

Tìm lời giải của bài toán

- Không gian trạng thái của bài toán: là đồ thị mà các đỉnh là các trạng thái của bài toán (trong đó có trạng thái đầu và trạng thái đích), các cung tương ứng với phép chuyển từ trạng thái này đến trạng thái khác

- Tìm lời giải của bài toán chuyển về tìm đường đi (ngắn nhất) từ trạng thái đầu đến trạng thái cuối trong đồ thị

- Thông thường, nếu toàn bộ không gian đồ thị trạng thái có thể lưu trữ trong bộ nhớ máy tính thì tìm lời giải bài toán chỉ đơn giản áp dụng các thuật toán tìm đường trong lý thuyết đồ thị. Trong hầu hết các bài toán trong Trí tuệ nhân tạo, không gian trạng thái bài toán thường rất lớn, cần phương pháp biểu diễn và tìm kiếm lời giải mới mà **không cần quyét hết không gian trạng thái bài toán**

Giải thuật chung tìm lời giải của bài toán

❑ **Solution:** is a sequence of operators that bring you from current state to the goal state

❑ **Basic idea:** Sử dụng một chiến lược tìm kiếm (strategy) để phát triển **cây tìm kiếm**, dừng khi cây chứa trạng thái đích của bài toán**.**

❑ **Function** General-Search(*problem*, ***strategy***) returns a *solution*, or failure

```
cây-tìm-kiếm ← trạng-thái-đầu;
while (1)
{
        if (cây-tìm-kiếm không thể mở rộng được nữa) then return failure
        nút-lá ← Chọn-1-nút-lá(cây-tìm-kiếm, strategy)
        if (node-lá là trạng-thái-đích) then return Đường-đi(trạng-thái-đầu, nút-lá)
        else mở-rộng(cây-tìm-kiếm, các-trạng-thái-kề(nút-lá)
}
```

❑ **Strategy:** The search strategy is determined by the **order** in which the nodes are expanded. (Tìm kiếm được xác định bởi thứ tự các đỉnh được triển khai)

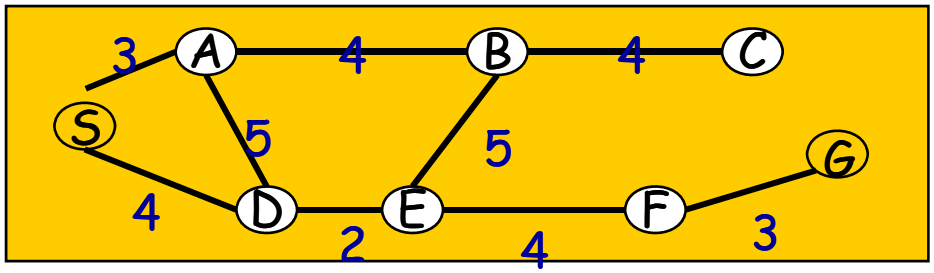
Cài đặt chi tiết giải thuật tìm lời giải

```
Function General-Search(problem, Queue/Stack) returns a solution, or failure
    Queue/Stack ← make-queue(make-node(initial-state[problem]));
    father(initial-state[problem]) = empty;
    while (1)
        if Queue/Stack is empty then return failure;
        node = pop(Queue/Stack) ;
        if test(node,Goal[problem]) then return path(node,father);
        expand-nodes ←adjacent-nodes(node, Operators[problem]);
        push(Queue/Stack, expand-nodes );
        foreach ex-node in expand-nodes
                father(ex-node) = node;
    end
```

```
Function path(node,father[]) : print the solution
    n ← node
    while (n # empty)
        cout<< n <<"<--" ;
        n = father[n];
    end
```

Cây tìm kiếm và không gian trạng thái

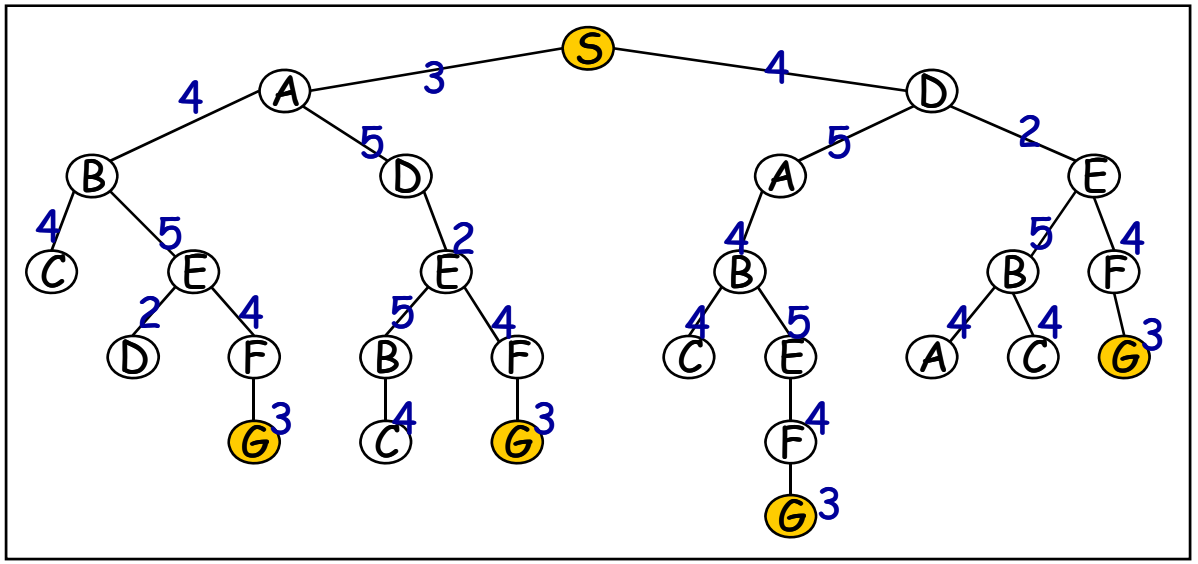
Đồ thị không gian trạng thái



Hoàn toàn xác định được từ bài toán (4 thành phần của bài toán)

Cây tìm kiếm là kết quả của một phương pháp duyệt đồ thị không gian trạng thái bài toán

Kg trạng thái + phương pháp duyệt
→ Cây tìm kiếmb



Cây tìm kiếm không lặp theo chiều rộng

Đánh giá phương pháp tìm kiếm

- A search method is defined by picking the order of node expansion. (Phương pháp tìm kiếm xác định thứ tự triển khai của các đỉnh trong cây tìm kiếm)
- Các giải thuật tìm kiếm thường được đánh giá dựa trên 4 tiêu chí sau:
  - **Completeness (tính đầy đủ):** Liệu nó luôn tìm ra nghiệm không nếu bài toán tồn tại nghiêm.
  - **Time complexity (độ phức tạp thời gian):** how long does it take as function of num. of nodes?
  - **Space complexity (độ phức tạp không gian):** how much memory does it require?
  - **Optimality (tính tối ưu):** Giải thuật có đảm bảo tìm ra nghiệm với hàm chi phí ít nhất không?
- Time and space complexity are measured in terms of:
  - *b* – max branching factor of the search tree
  - *d* – depth of the least-cost solution
  - *m* – max depth of the search tree (may be infinity)

Outline: Giải quyết bài toán và tìm kiếm
(Problem solving and search, problem solving agents)

- **Hình thành bài toán (Problem formulation)**
  - Xác định các thành phần của bài toán (problem formulation)
  - Lược đồ chung để giải bài toán (general-solving procedure)
  - Đánh giá một giải thuật tìm kiếm
- **Uninformed (blink) search (Tìm không có thông tin phản hồi, hoặc tìm kiếm mù)**
  - Search strategies: depth-first, breadth-first (Các chiến lược tìm kiếm: theo chiều sâu, theo chiều rộng)
- **Informed search (Tìm kiếm dựa trên các hàm đánh giá, heuristics)**
  - Search strategies: **best-first search**
  - Heuristic functions (Các hàm heuristic)
    - ✓ Uniform search
    - ✓ Greedy search
    - ✓ A* search

Uninformed search strategies

Use only information available in the problem formulation

- Breadth-first
- Depth-first
- Depth-limited
- Iterative deepening (ko co trong tai lieu tieng Viet)

Tìm kiếm lời giải theo chiều rộng

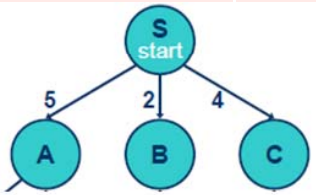
```
Function General-Search(problem, Queue) returns a solution, or failure
    Queue ← make-queue(make-node(initial-state[problem]));
    father(initial-state[problem]) = empty;
    while (1)
        if Queue is empty then return failure;
        node = pop(Queue) ;
        if test(node,Goal[problem]) then return path(node,father);
        expand-nodes ←adjacent-nodes(node, Operators[problem]);
        push(Queue, expand-nodes );
        foreach ex-node in expand-nodes
            father(ex-node) = node;
    end
```

Lấy các đỉnh kề với node, nhưng chưa xuất hiện trong cây tìm kiếm

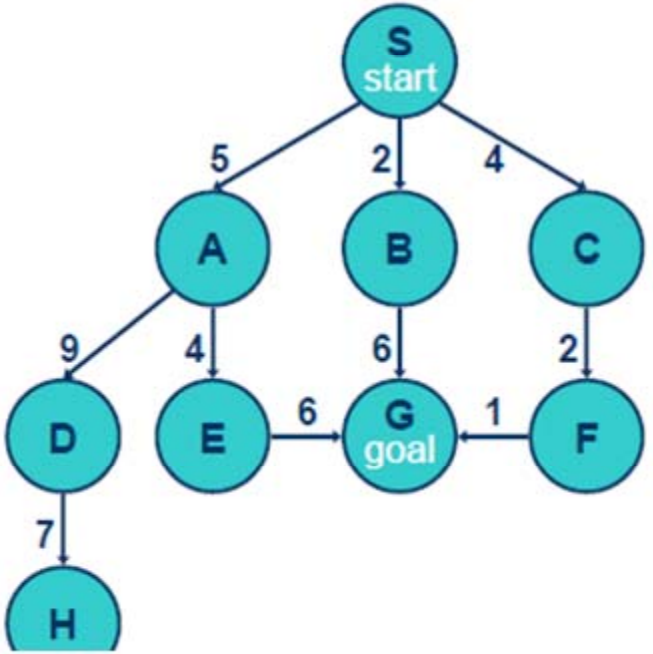
```
Function path(node,father[]) : print the solution
    n ← node
    while (n # empty)
        cout<< n <<"<--" ;
        n = father[n];
    end
```

Tìm kiếm theo chiều rộng

node	Queue	father
	S	
S	A, B, C	Father[A,B,C]=S



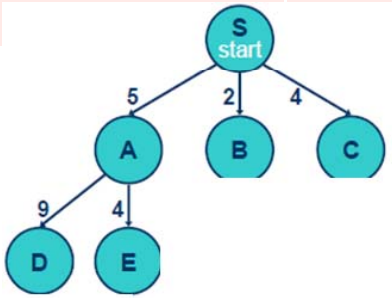
Cây tìm kiếm theo chiều rộng



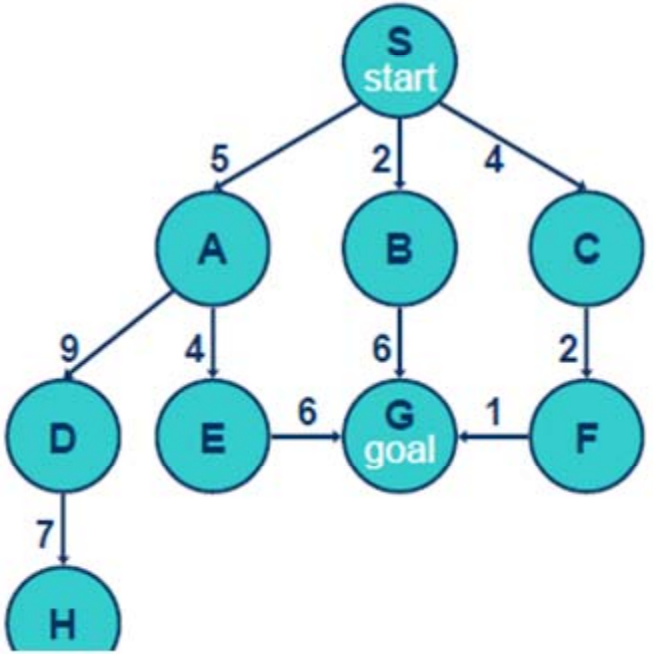
Đồ thị không gian trạng thái

Tìm kiếm theo chiều rộng

node	Queue	father
	S	
S	A, B, C	Father[A,B,C]=S
A	B, C, D, E	Father[D,E]=A



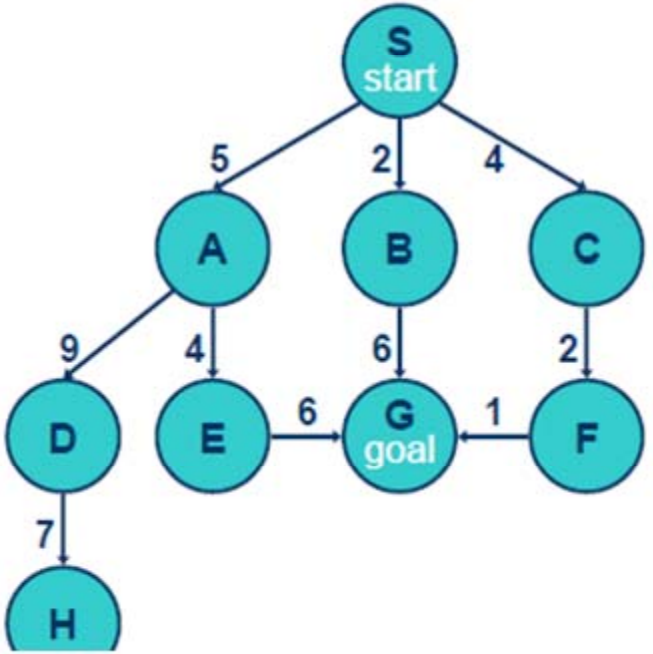
Cây tìm kiếm theo chiều rộng



Đồ thị không gian trạng thái

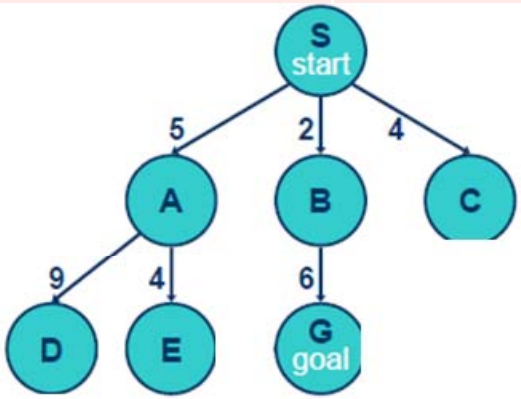
Tìm kiếm theo chiều rộng

node	Queue	father
	S	
S	A, B, C	Father[A,B,C]=S
A	B, C, D, E	Father[D,E]=A
B	C,D,E,G	Father[G]=B



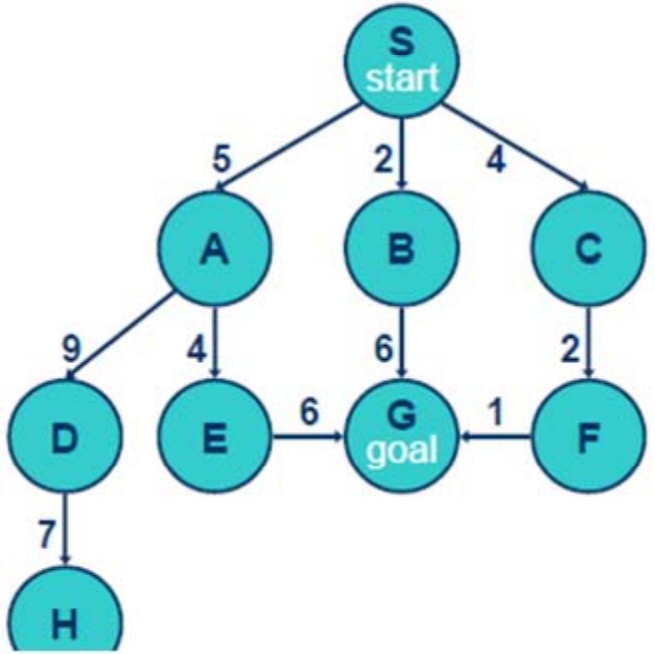
Đồ thị không gian trạng thái

Cây tìm kiếm theo chiều rộng



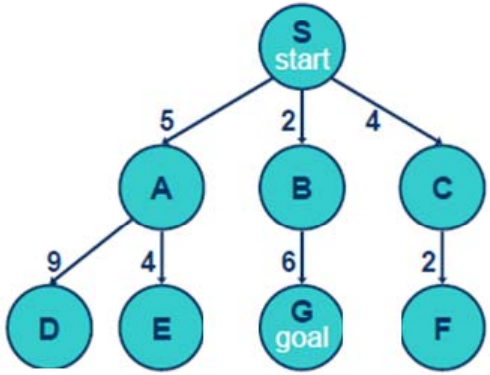
Tìm kiếm theo chiều rộng

node	Queue	father
	S	
S	A, B, C	Father[A,B,C]=S
A	B, C, D, E	Father[D,E]=A
B	C,D,E,G	Father[G]=B
C	D, E, G, F	Father[F]=C



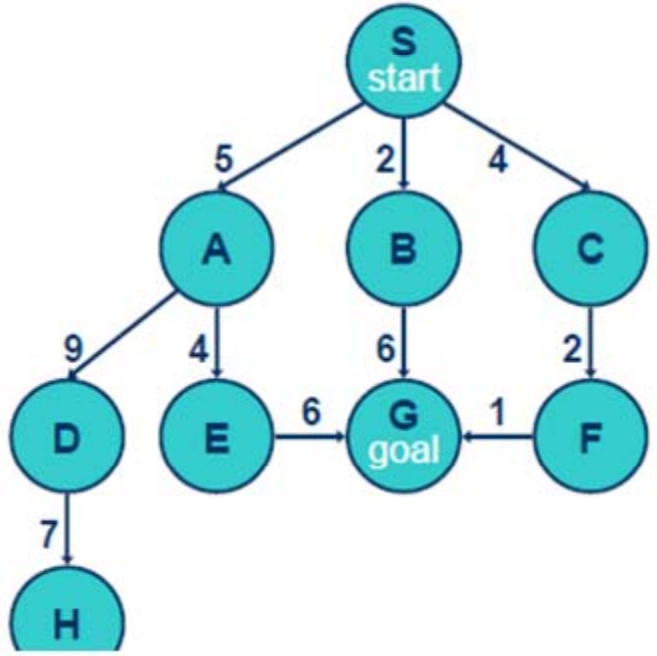
Đồ thị không gian trạng thái

Cây tìm kiếm theo chiều rộng



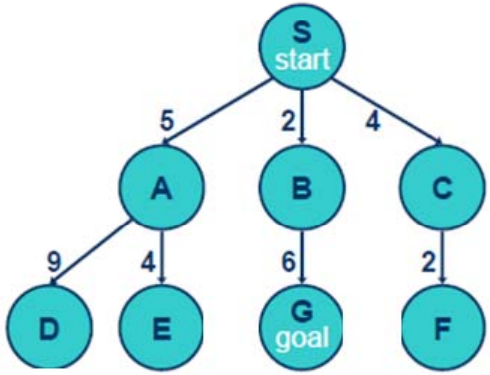
Tìm kiếm theo chiều rộng

node	Queue	father
	S	
S	A, B, C	Father[A,B,C]=S
A	B, C, D, E	Father[D,E]=A
B	C,D,E,G	Father[G]=B
C	D, E, G, F	Father[F]=C
D	E,G, F, H	Father[H]=F



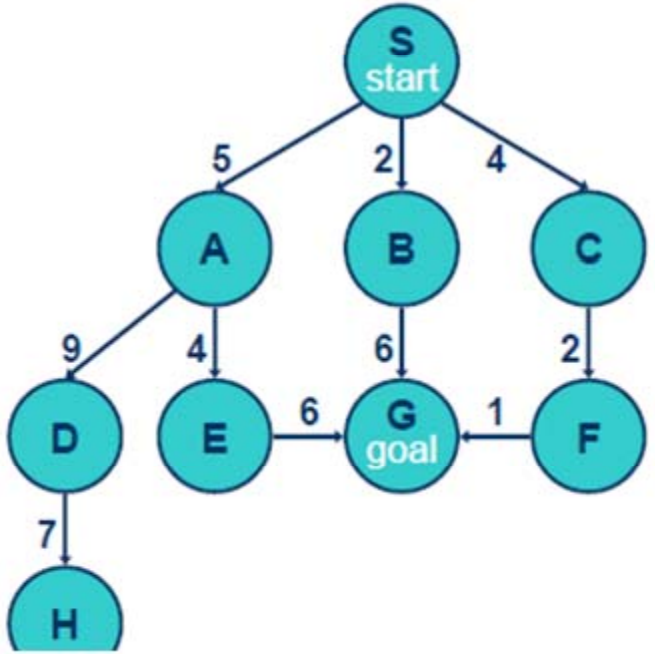
Đồ thị không gian trạng thái

Cây tìm kiếm theo chiều rộng



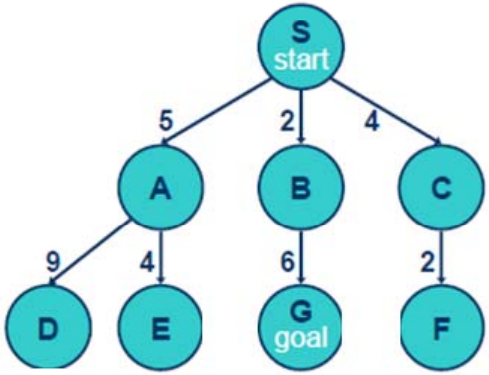
Tìm kiếm theo chiều rộng

node	Queue	father
	S	
S	A, B, C	Father[A,B,C]=S
A	B, C, D, E	Father[D,E]=A
B	C,D,E,G	Father[G]=B
C	D, E, G, F	Father[F]=C
D	E,G, F, H	Father[H]=F
E	G, F, H	



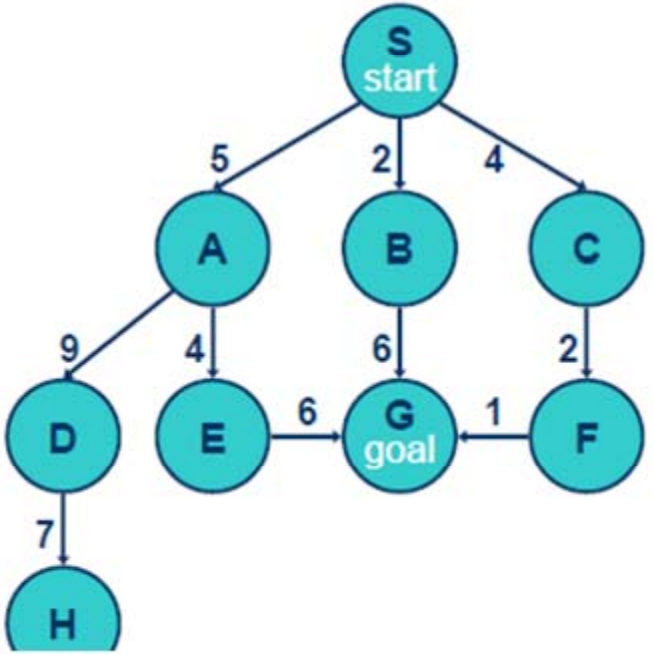
Đồ thị không gian trạng thái

Cây tìm kiếm theo chiều rộng



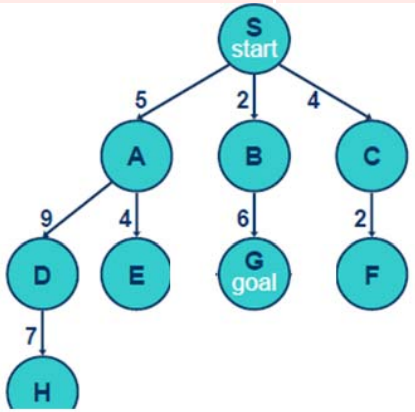
Tìm kiếm theo chiều rộng

node	Queue	father
	S	
S	A, B, C	Father[A,B,C]=S
A	B, C, D, E	Father[D,E]=A
B	C,D,E,G	Father[G]=B
C	D, E, G, F	Father[F]=C
D	E,G, F, H	Father[H]=F
E	G, F, H	
G	F, H	

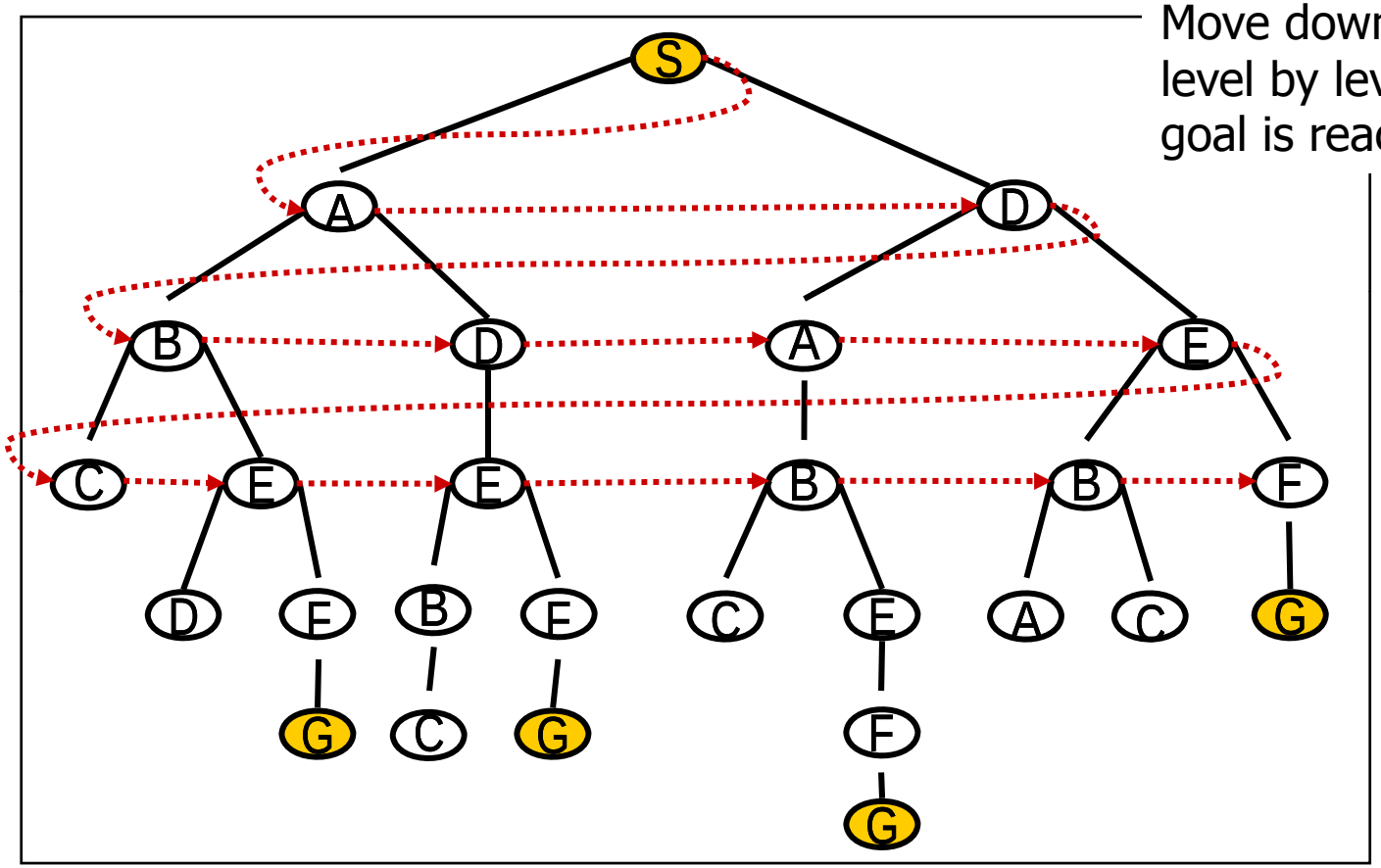


Đồ thị không gian trạng thái

Cây tìm kiếm theo chiều rộng



Tìm kiếm theo chiều rộng



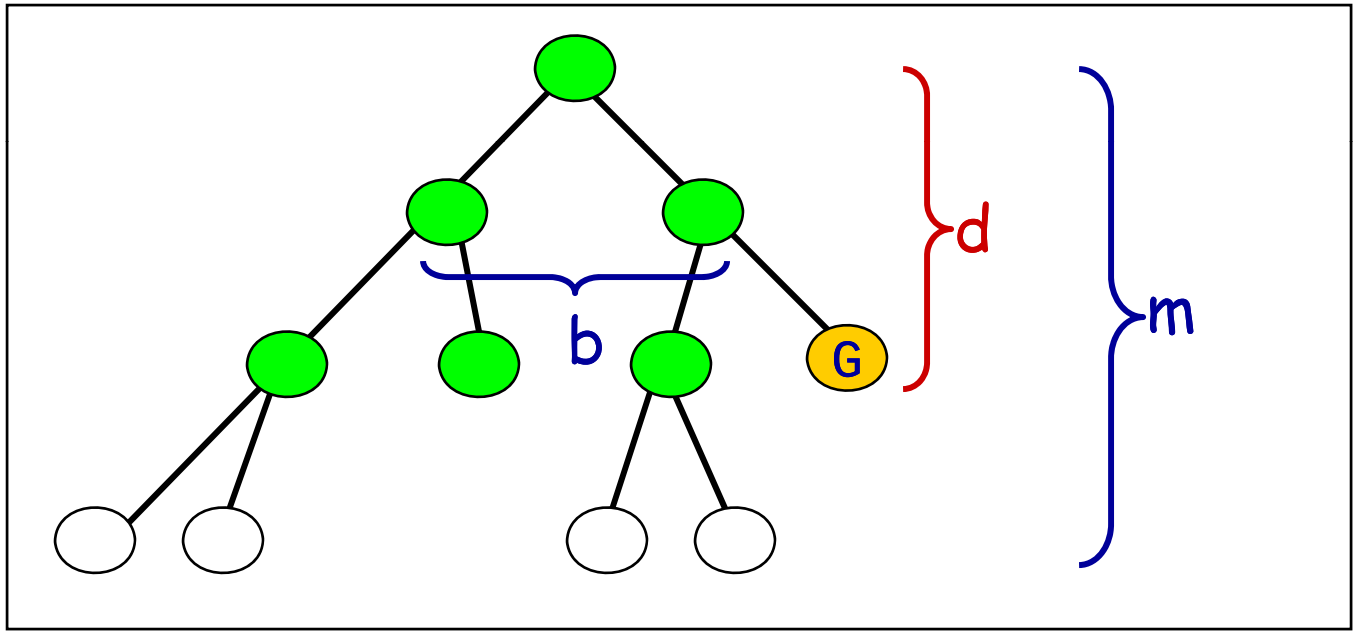
Move downwards, level by level, until goal is reached.

Properties of breadth-first search

- Completeness: Yes, if *b* is finite
- Time complexity: $1+b+b^2+\ldots+b^d = O(b^d)$, i.e., exponential in *d*
- Space complexity: $O(b^d)$ (see following slides)
- Optimality: Yes (assuming cost = 1 per step)

Time complexity of breadth-first search

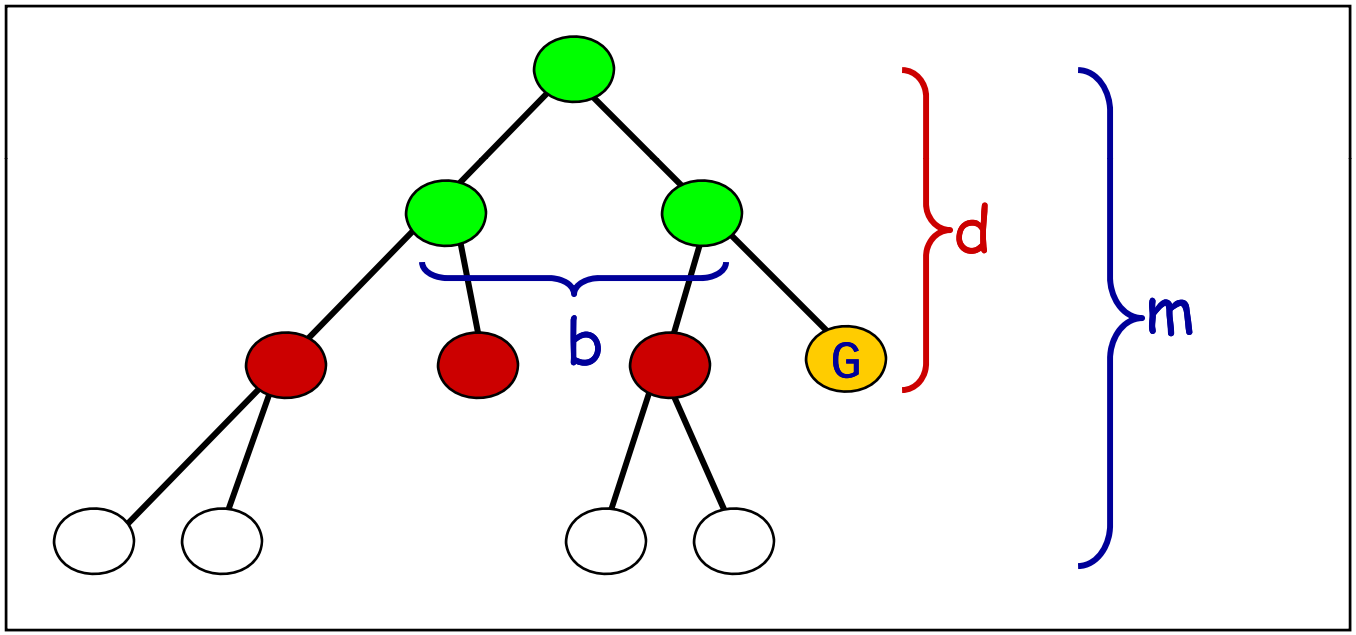
- If a goal node is found on depth d of the tree, all nodes up till that depth are created.



- Thus: $O(b^d)$

Space complexity of breadth-first

- Largest number of nodes in QUEUE is reached on the level d of the goal node.



- QUEUE contains all  and  nodes. (Thus: 4) .
- In General: b^{d}

Note: Loop Detection

- Problem: the search may fail or be sub-optimal if:
  - no loop detection: then algorithm runs into infinite cycles
    (A -> B -> A -> B -> …)
  - not queuing-up a node that has a state which we have already visited: may yield suboptimal solution
  - simply avoiding to go back to our parent: looks promising, but we have not proven that it works
- Solution? do not enter queue a node if its state matches the state of any of its parents (assuming path costs>0).

  Indeed, if path costs > 0, it will always cost us more to consider a node with that state again than it had already cost us the first time.
- **Is that enough??**

Tìm kiếm lời giải theo chiều sâu

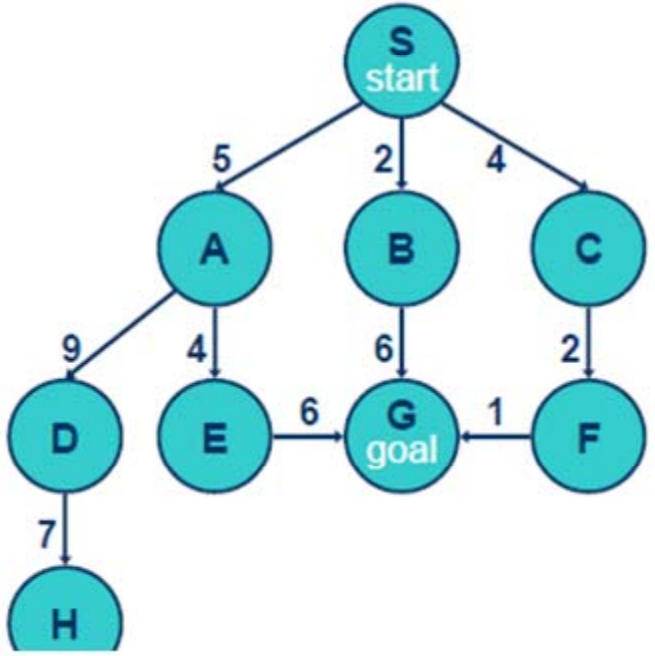
```
Function General-Search(problem, Stack) returns a solution, or failure
    Stack ← make-queue(make-node(initial-state[problem]));
    father(initial-state[problem]) = empty;
    while (1)
        if Stack is empty then return failure;
        node = pop(Stack) ;
        if test(node,Goal[problem]) then return path(node,father);
        expand-nodes ←adjacent-nodes(node, Operators[problem]);
        push(Stack, expand-nodes );
        foreach ex-node in expand-nodes
                father(ex-node) = node;
    end
```

Lấy các đỉnh kề với node, nhưng chưa xuất hiện trong cây tìm kiếm

```
Function path(node,father[]) : print the solution
    n ← node
    while (n # empty)
        cout<< n <<"<--" ;
        n = father[n];
    end
```

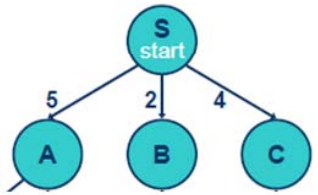
Tìm kiếm theo chiều sâu

node	Stack	father
	S	
S	A, B, C	Father[A,B,C]=S



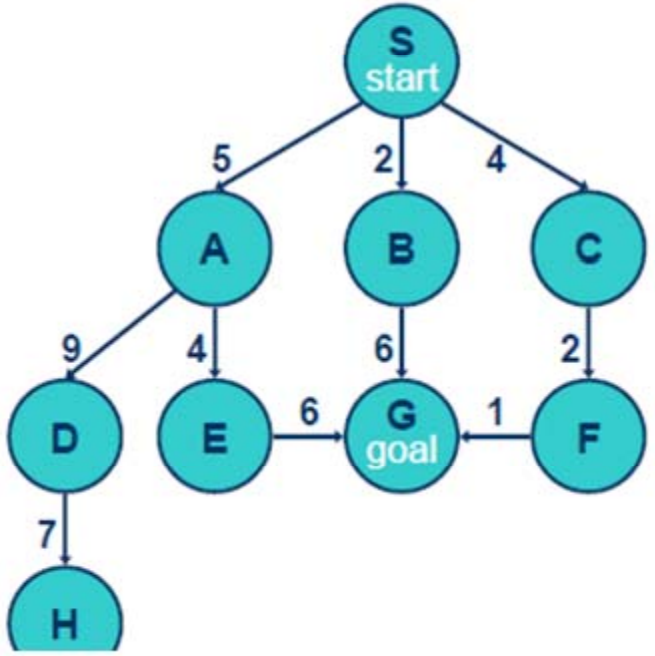
Đồ thị không gian trạng thái

Cây tìm kiếm theo chiều sâu



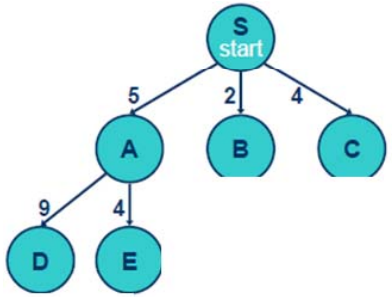
Tìm kiếm theo chiều sâu

node	Stack	father
	S	
S	A, B, C	Father[A,B,C]=S
A	D, E, B, C	Father[D,E]=A



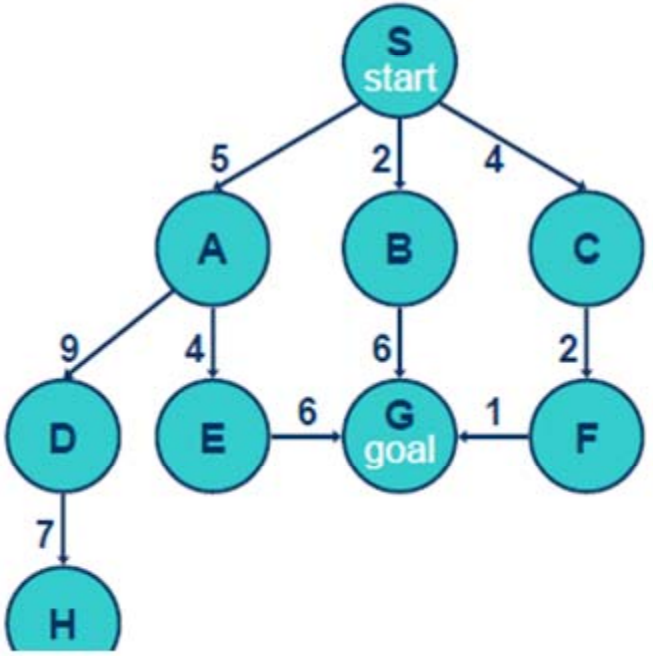
Đồ thị không gian trạng thái

Cây tìm kiếm theo chiều sâu



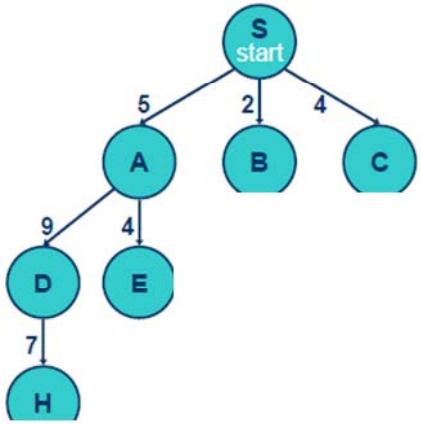
Tìm kiếm theo chiều sâu

node	Stack	father
	S	
S	A, B, C	Father[A,B,C]=S
A	D, E, B, C	Father[D,E]=A
D	H, E, B, C	Father[H]=D



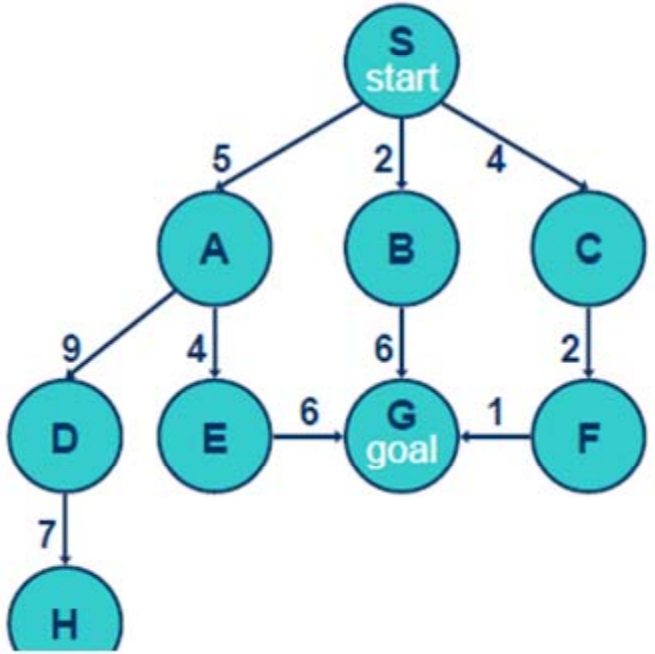
Đồ thị không gian trạng thái

Cây tìm kiếm theo chiều sâu



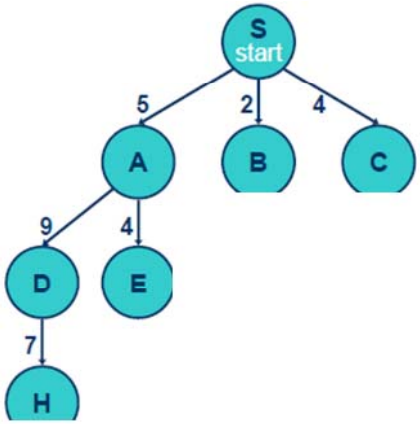
Tìm kiếm theo chiều sâu

node	Stack	father
	S	
S	A, B, C	Father[A,B,C]=S
A	D, E, B, C	Father[D,E]=A
D	H, E, B, C	Father[H]=D
H	E, B, C	



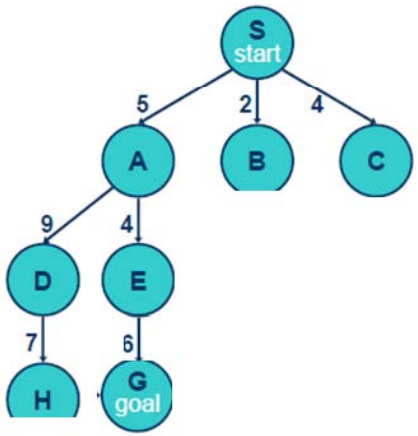
Đồ thị không gian trạng thái

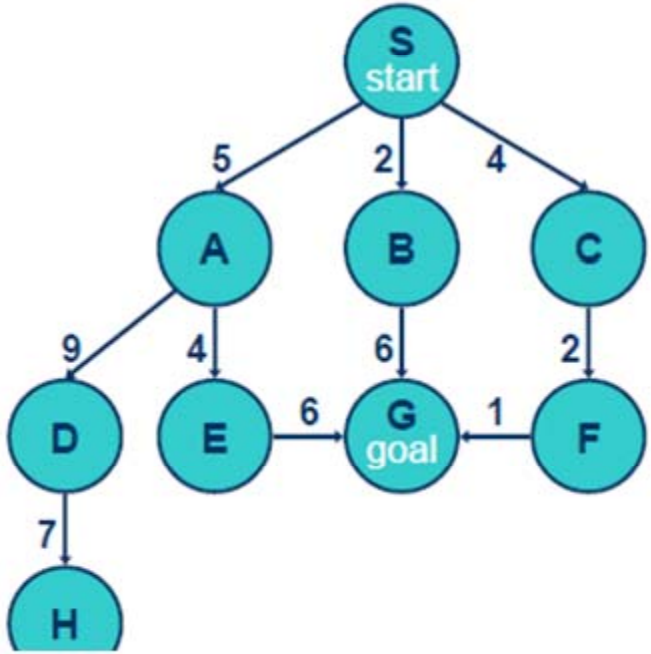
Cây tìm kiếm theo chiều sâu



Tìm kiếm theo chiều sâu

node	Stack	father
	S	
S	A, B, C	Father[A,B,C]=S
A	D, E, B, C	Father[D,E]=A
D	H, E, B, C	Father[H]=D
H	E, B, C	
E	G, B, C	Father[G]=E

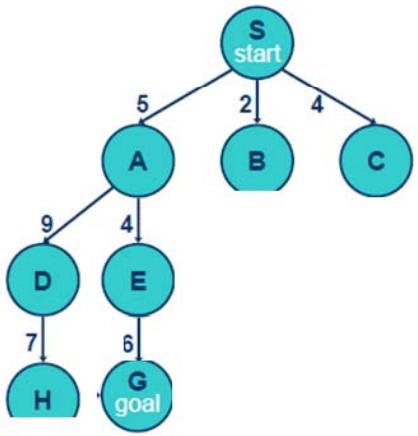


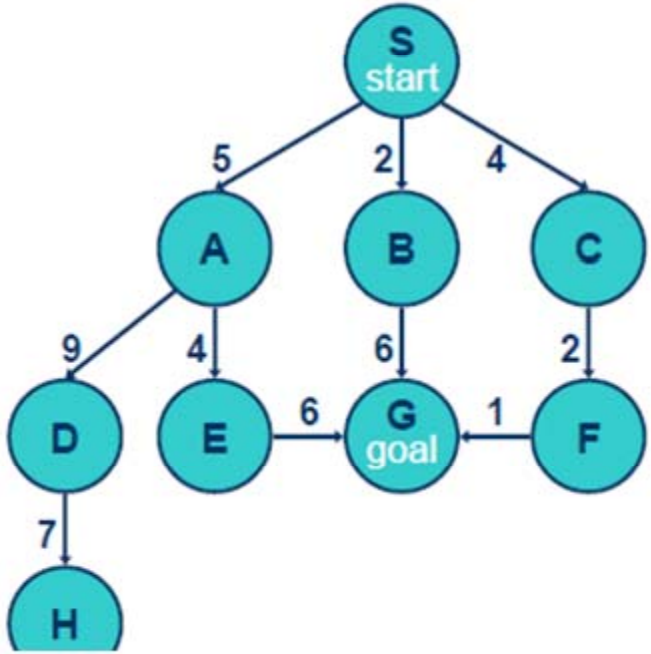


Đồ thị không gian trạng thái

Tìm kiếm theo chiều sâu

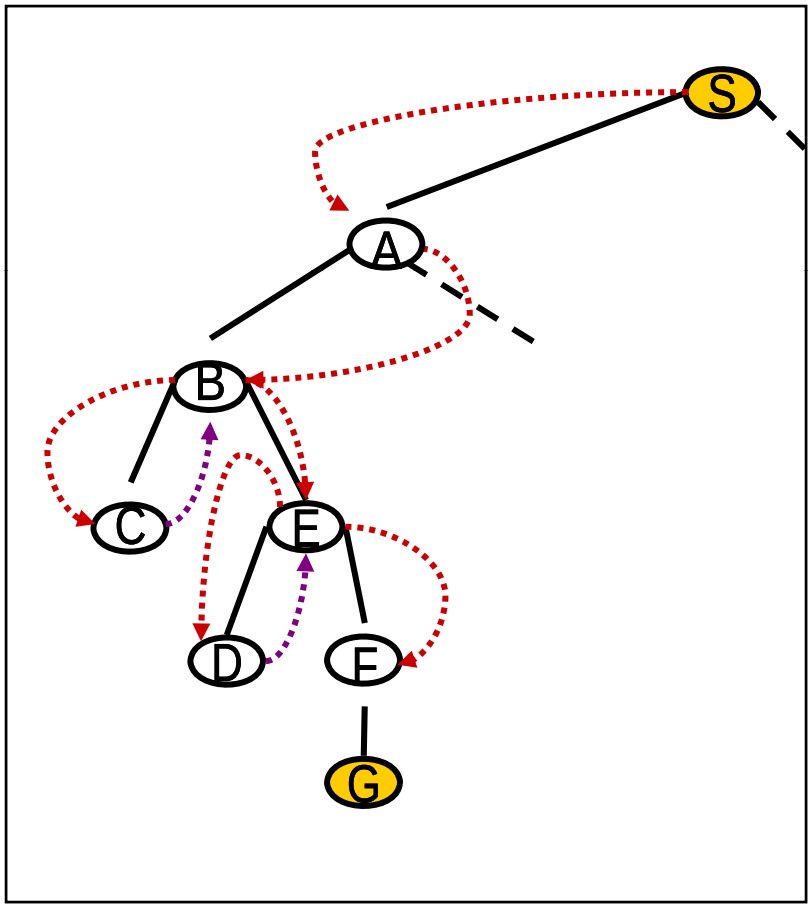
node	Stack	father
	S	
S	A, B, C	Father[A,B,C]=S
A	D, E, B, C	Father[D,E]=A
D	H, E, B, C	Father[H]=D
H	E, B, C	
E	G, B, C	Father[G]=E
G		





Đồ thị không gian trạng thái

Depth First Search



Properties of depth-first search

- Completeness: No, fails in infinite state-space (yes if finite state space)
- Time complexity: $O(b^m)$
- Space complexity: $O(bm)$
- Optimality: No

Remember:

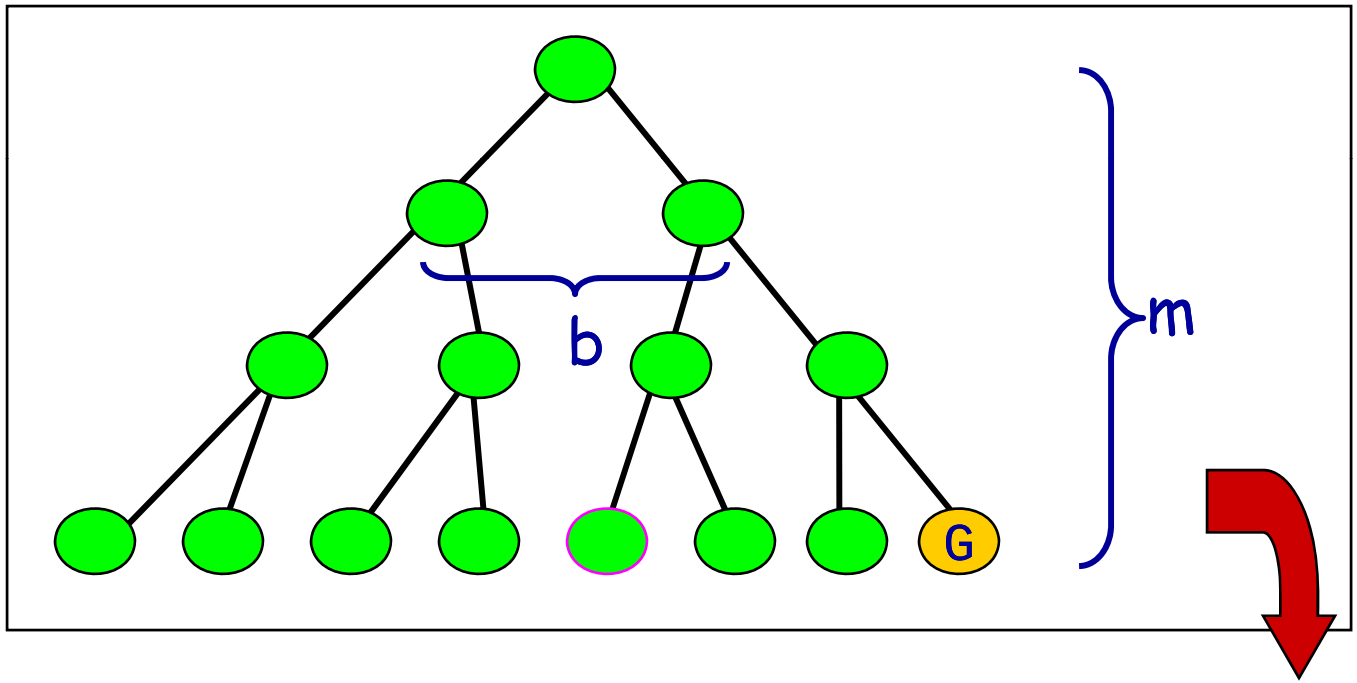
b = branching factor

m = max depth of search tree

Time complexity of depth-first: details

- In the worst case:

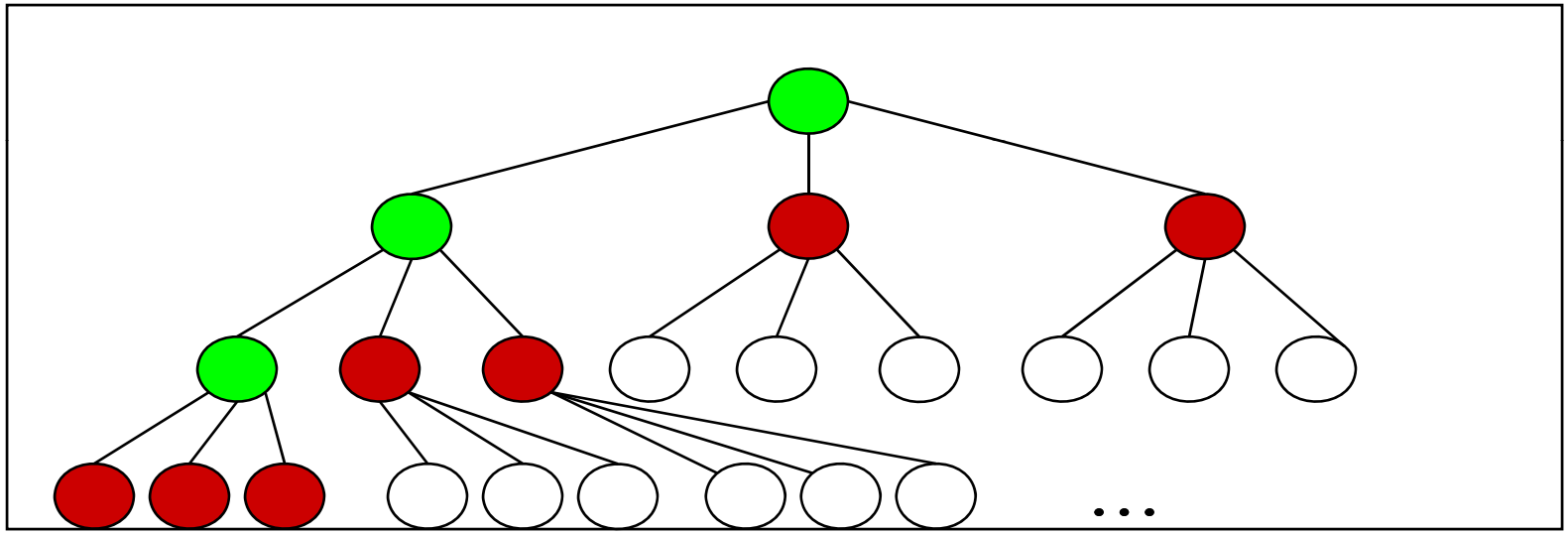
  the (only) goal node may be on the right-most branch,



- Time complexity == $b^{m} + b^{m-1} + \ldots + 1 = (b^{m+1} - 1)/(b-1)$
- Thus: $O(b^{m})$

Space complexity of depth-first

- Largest number of nodes in STACK is reached in bottom left-most node.
- Example: m = 3, b = 3 :



- STACK contains all  nodes. Thus: 7.
- In General: ((b-1) * m) + 1
- Order: O(m*b)

Avoiding repeated states

In increasing order of effectiveness and computational overhead:

- do not return to state we come from, i.e., expand function will skip possible successors that are in same state as node's parent.
- do not create paths with cycles, i.e., expand function will skip possible successors that are in same state as any of node's ancestors.
- do not generate any state that was ever generated before, by keeping track (in memory) of every state generated, unless the cost of reaching that state is lower than last time we reached it.

Depth-limited search

Is a depth-first search with depth limit

Implementation:

Nodes at depth *l* have no successors.

Complete:

if cutoff chosen appropriately then it is guaranteed to find a solution.

Optimal: it does not guarantee to find the least-cost solution

Iterative deepening search

❑ **Function** Iterative-deepening-Search(*problem*) **returns** a solution, or failure

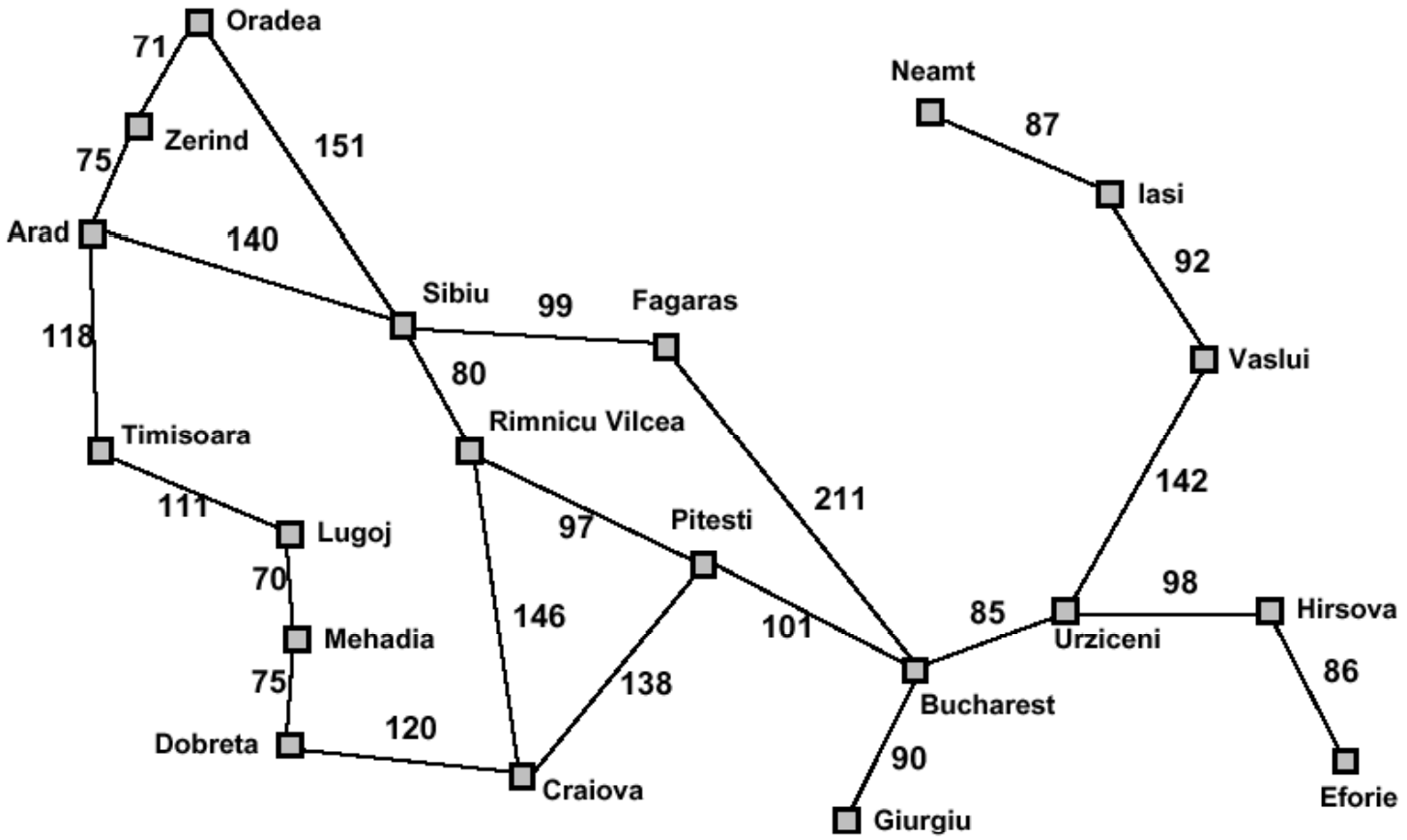
```
for depth = 0 to ∞ do
        result ← Depth-Limited-Search(problem, depth)
        if result succeeds then return result
end
return failure
```

❑ Idea: Combines the best of breadth-first and depth-first search strategies.

❑ Evaluation:

- Completeness: Yes,
- Time complexity: $O(b^d)$
- Space complexity: $O(bd)$
- Optimality: Yes, if step cost = 1

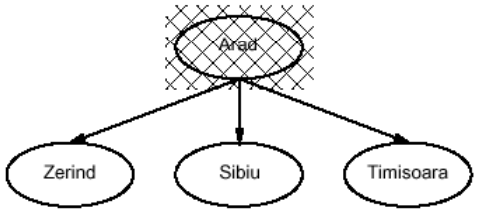
Romania with step costs in km



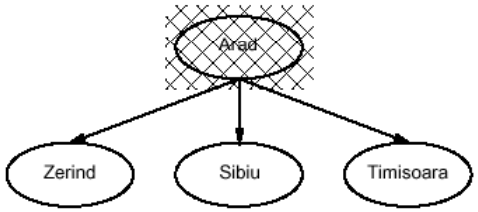
Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

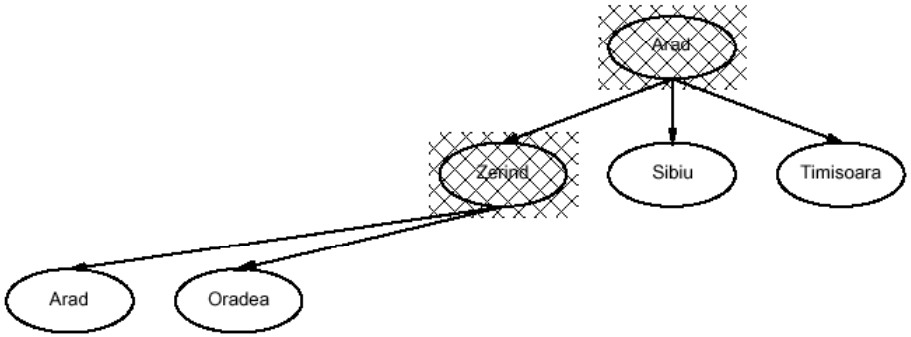
Iterative deepening search $l = 0$

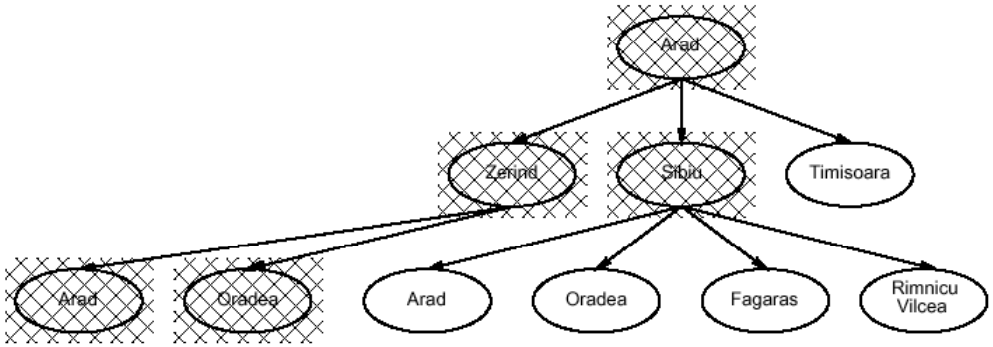
Iterative deepening search $l = 1$

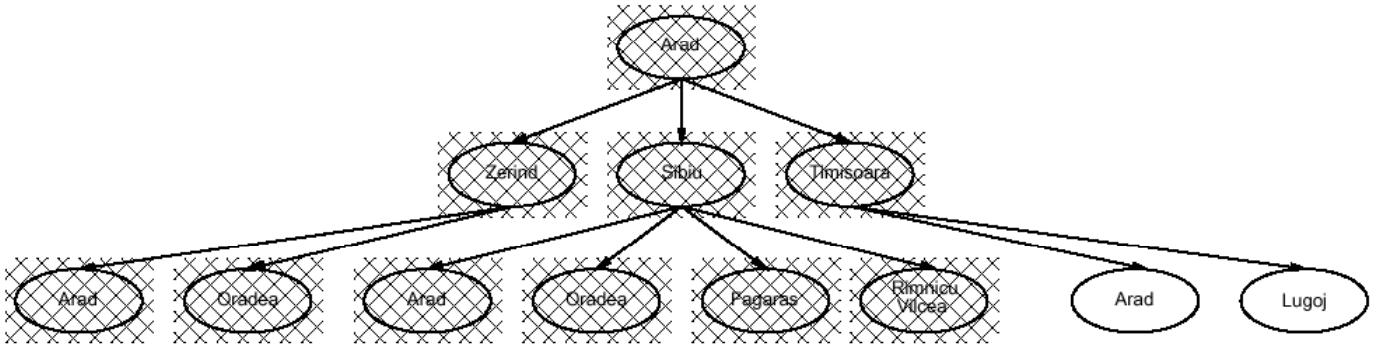

Arad
Zerind
Sibiu
Timisoara

Iterative deepening search $l = 2$

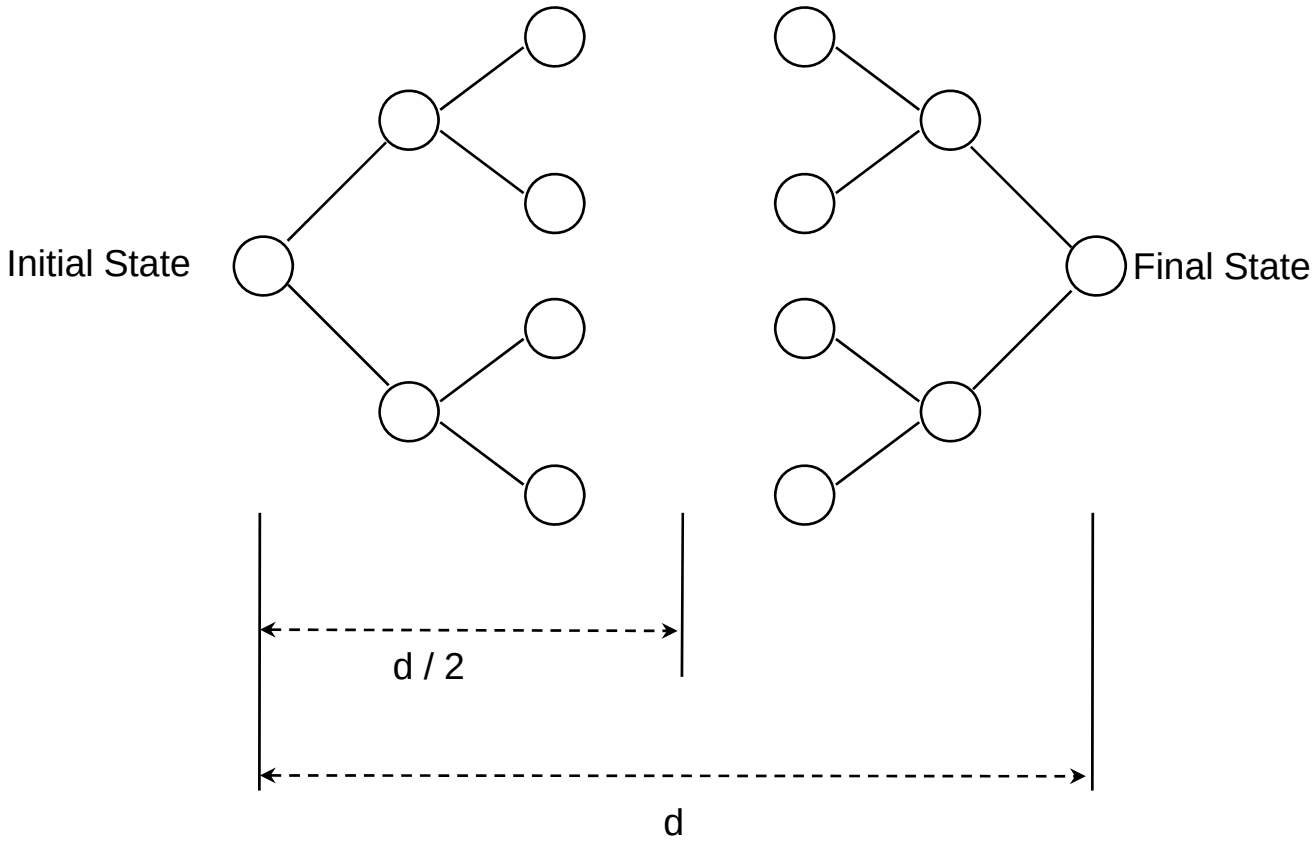

Arad
Zerind
Sibiu
Timisoara


Arad
Zerind
Sibiu
Timisoara
Arad
Oradea


Arad
Zerind
Sibiu
Timisoara
Arad
Oradea
Arad
Oradea
Fagaras
Rimnicu
Vilcea


Arad
Zerind
Sibiu
Timisoara
Arad
Oradea
Arad
Oradea
Pagaras
Rimnicu Vilcea
Arad
Lugoj

Bidirectional Search



Bidirectional search

1. QUEUE1 <-- path only containing the root;
   QUEUE2 <-- path only containing the goal;

2. WHILE both QUEUEs are not empty
   AND QUEUE1 and QUEUE2 do NOT share a state

   DO remove their first paths;
   create their new paths (to all children);
   reject their new paths with loops;
   add their new paths to back;

3. IF QUEUE1 and QUEUE2 share a state
   THEN success;
   ELSE failure;

Comparing uninformed search strategies

Criterion	Breadth-first	Depth-first	Depth-limited	Iterative deepening	Bidirectional (if applicable)
Time	b^d	b^m	b^l	b^d	b^(d/2)
Space	b^d	bm	bl	bd	b^(d/2)
Optimal?	Yes	No	No	Yes	Yes
Complete?	Yes	No	Yes,	Yes if l≥d	Yes

- *b* – max branching factor of the search tree
- *d* – depth of the least-cost solution
- *m* – max depth of the state-space (may be infinity)
- *l* – depth cutoff

Outline: Giải quyết bài toán và tìm kiếm (Problem solving and search, problem solving agents)

- **Hình thành bài toán (Problem formulation)**
  - Xác định các thành phần của bài toán (problem formulation)
  - Lược đồ chung để giải bài toán (general-solving procedure)
  - Đánh giá một giải thuật tìm kiếm
- **Uninformed (blink) search (Tìm không có thông tin phản hồi, hoặc tìm kiếm mù)**
  - Search strategies: depth-first, breadth-first (Các chiến lược tìm kiếm: theo chiều sâu, theo chiều rộng)
- **Informed search (Tìm kiếm dựa trên các hàm đánh giá, heuristics)**
  - Search strategies: **best-first search**
  - Heuristic functions (Các hàm heuristic)
    - ✓ Uniform search
    - ✓ Greedy search
    - ✓ A* search

(Next time)