



# **CS540 Intro to AI Uninformed Search**

**University of Wisconsin-Madison  
Spring 2023**

# Announcements

## Homeworks:

- Homework 7 due Tuesday

## Class roadmap:

Thursday, April 6	Uninformed Search
Tuesday, April 11	Informed Search
Thursday, April 13	Advanced Search
Tuesday, April 18	Games I
Thursday, April 20	Games II

# Announcements

## Homeworks:

- Homework 7 due Tuesday

## Class roadmap:

Thursday, April 6	Uninformed Search
Tuesday, April 11	Informed Search
Thursday, April 13	Advanced Search
Tuesday, April 18	Games I
Thursday, April 20	Games II

(Mostly) Done with neural networks — check understanding with practice questions on Canvas.

**Many AI problems can be formulated as search.**

# **Many AI problems can be formulated as search.**

How to make a sequence of decisions to reach a desired goal.

# Many AI problems can be formulated as search.

How to make a sequence of decisions to reach a desired goal.

Leverage computation and a known model of world **dynamics** to make decisions.

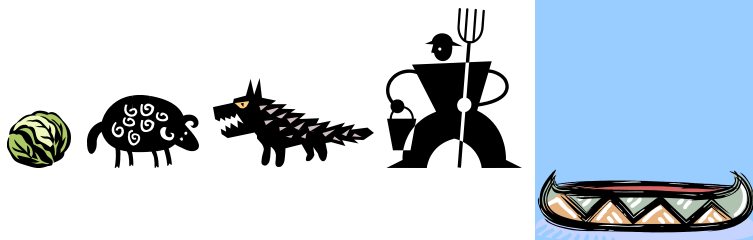
# Many AI problems can be formulated as search.

How to make a sequence of decisions to reach a desired goal.

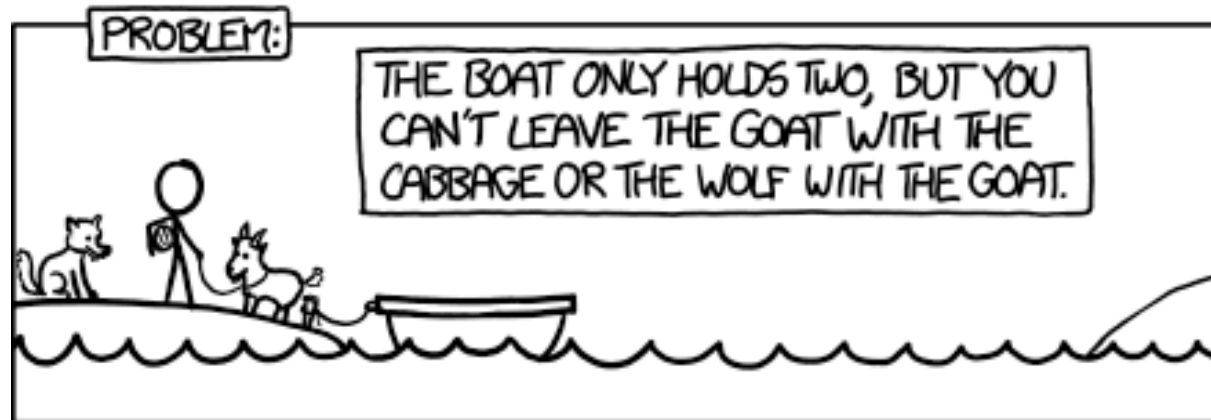
Leverage computation and a known model of world **dynamics** to make decisions.

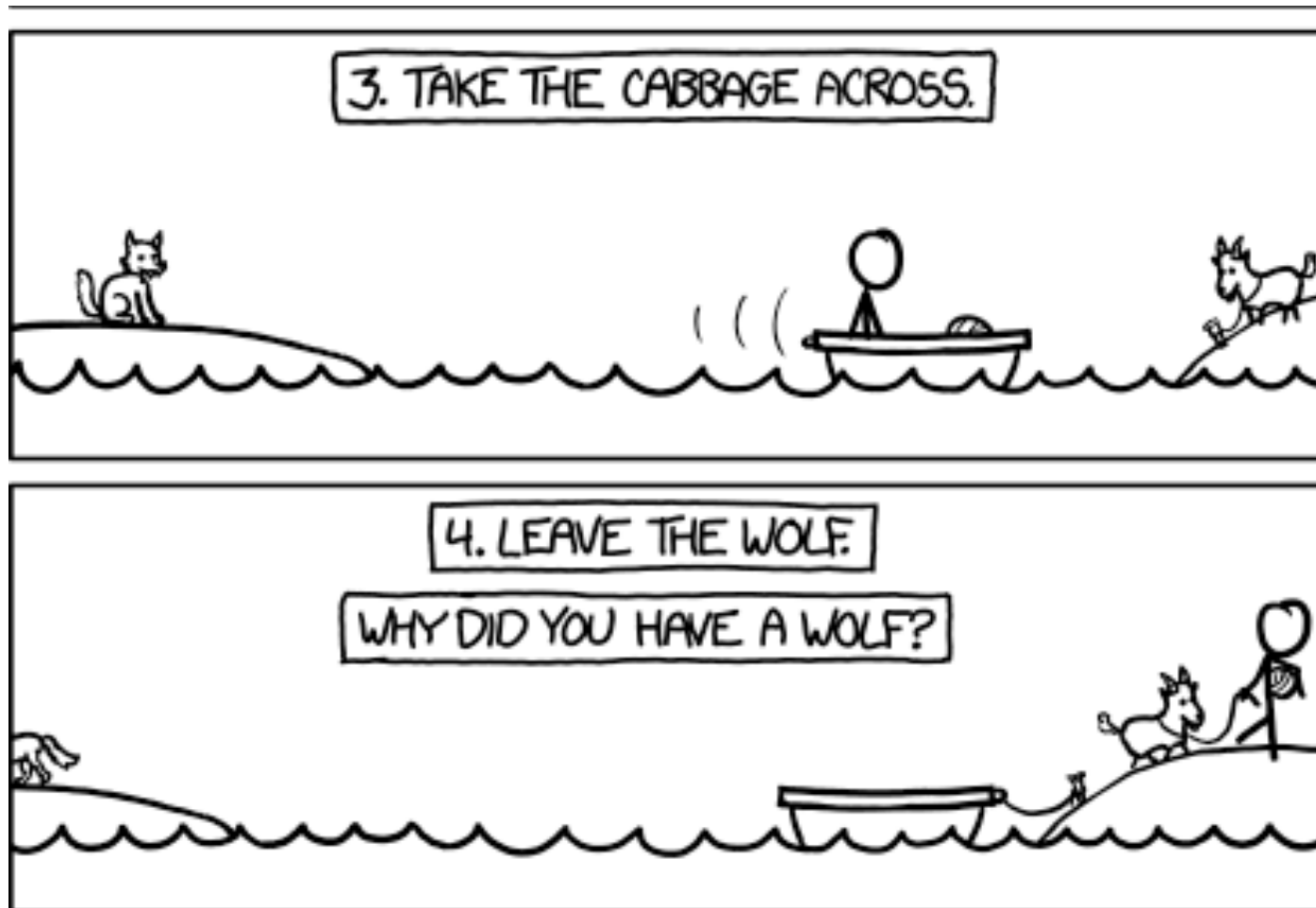


“How the world changes in response to agent actions”

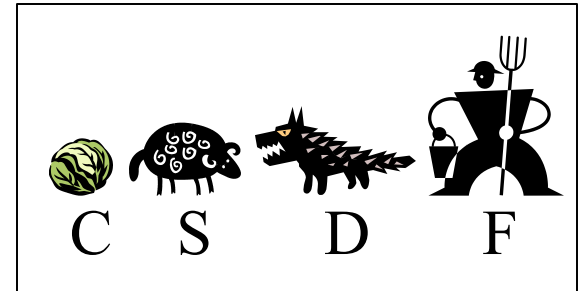






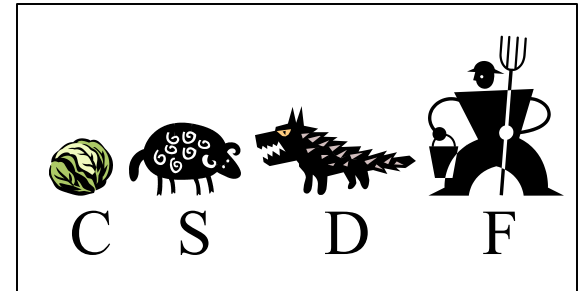


# The search problem



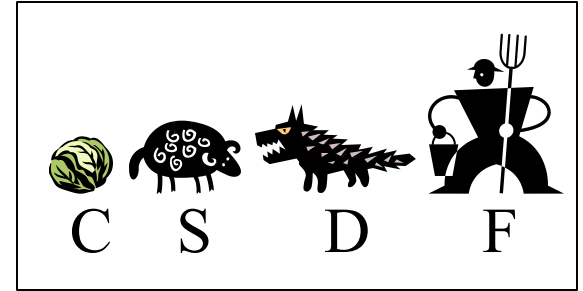
# The search problem

- State space **S** : all valid configurations



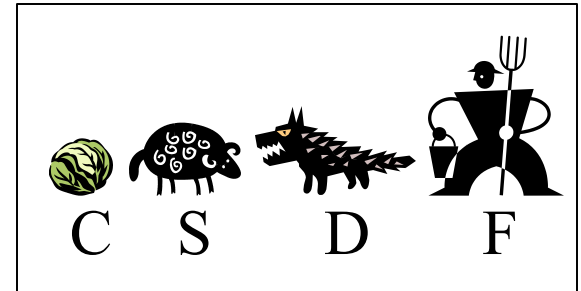
# The search problem

- State space **S** : all valid configurations
- Initial state  $I = \{(CSDF,)\} \subseteq S$



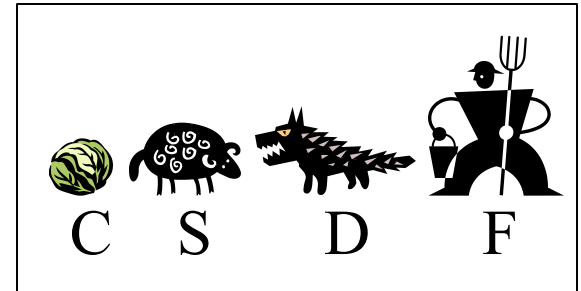
# The search problem

- State space **S** : all valid configurations
- Initial state **I** = {(CSDF,)}  $\subseteq$  **S**
- Goal state **G** = {(,CSDF)}  $\subseteq$  **S**



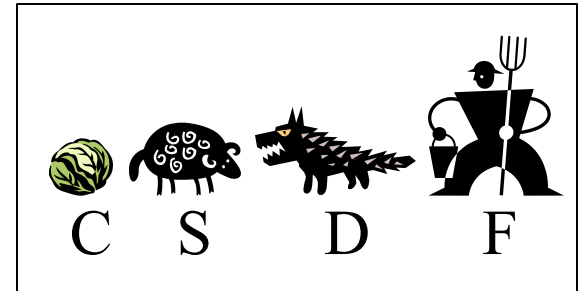
# The search problem

- State space **S** : all valid configurations
- Initial state **I** = {(CSDF,)}  $\subseteq$  **S**
- Goal state **G** = {(,CSDF)}  $\subseteq$  **S**
- Successor function **succs**(s)  $\subseteq$  **S** : states reachable in one step from state s
  - **succs**((CSDF,)) = {(CD, SF)}
  - **succs**((CDF,S)) = {(CD,FS), (D,CFS), (C, DFS)}



# The search problem

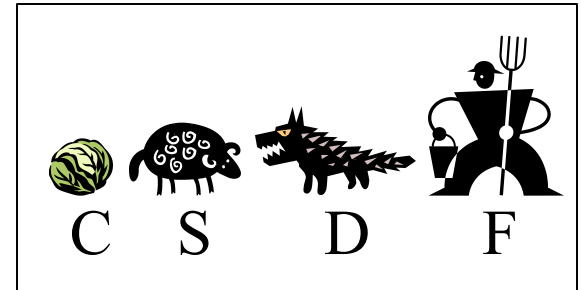
- State space **S** : all valid configurations
- Initial state **I** =  $\{(CSDF,)\} \subseteq \mathbf{S}$
- Goal state **G** =  $\{(\,,CSDF)\} \subseteq \mathbf{S}$
- Successor function **succs**(s)  $\subseteq \mathbf{S}$  : states reachable in one step from state s
  - **succs**((CSDF,)) = {(CD, SF)}
  - **succs**((CDF,S)) = {(CD,FS), (D,CFS), (C, DFS)}
- **Cost**(s,s')=1 for all steps. (weighted later)





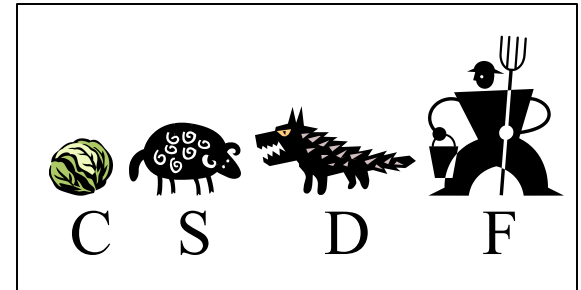
# The search problem

- State space  $\mathbf{S}$  : all valid configurations
- Initial state  $I = \{(CSDF,)\} \subseteq \mathbf{S}$
- Goal state  $\mathbf{G} = \{(\,,CSDF)\} \subseteq \mathbf{S}$
- Successor function  $\mathbf{succs}(s) \subseteq \mathbf{S}$  : states reachable in one step from state  $s$ 
  - $\mathbf{succs}((CSDF,)) = \{(CD, SF)\}$
  - $\mathbf{succs}((CDF,S)) = \{(CD,FS), (D,CFS), (C, DFS)\}$
- Cost( $s,s'$ )=1 for all steps. (weighted later)
- The search problem: find a solution path from a state in  $I$  to a state in  $\mathbf{G}$ .
  - Optionally minimize the cost of the solution.

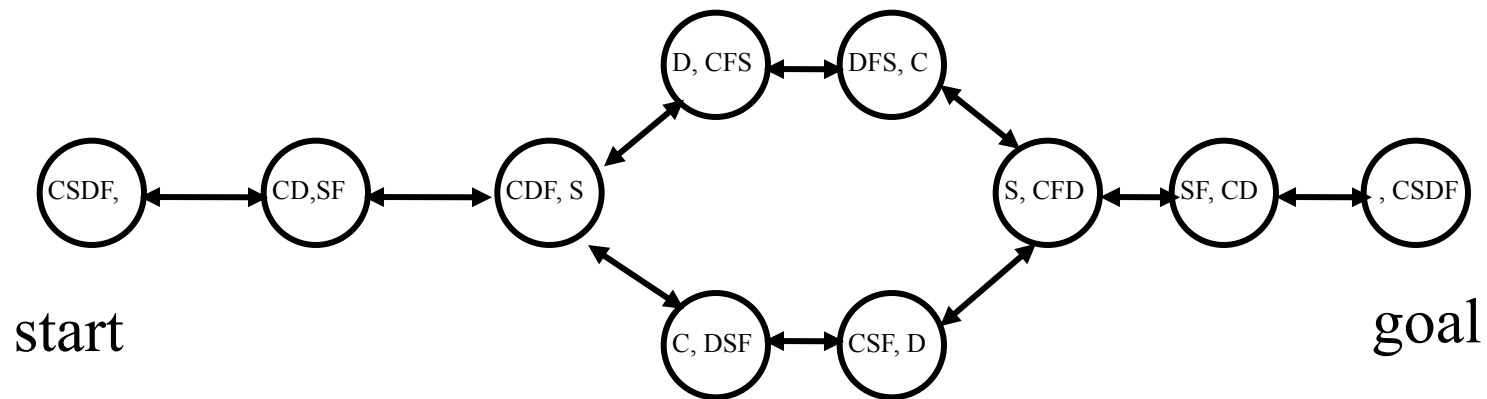
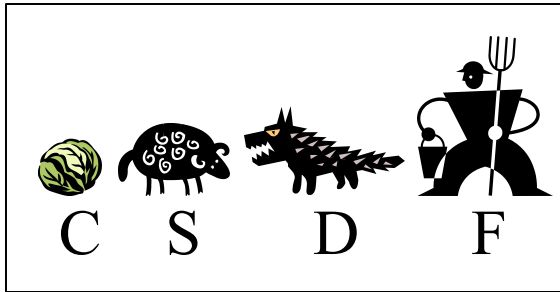


# The search problem

- State space **S** : all valid configurations
- Initial state **I** = {(CSDF,)}  $\subseteq$  **S**
- Goal state **G** = {(,CSDF)}  $\subseteq$  **S**
- Successor function **succs**(s)  $\subseteq$  **S** : states reachable in one step from state s
  - **succs**((CSDF,)) = {(CD, SF)}
  - **succs**((CDF,S)) = {(CD,FS), (D,CFS), (C, DFS)}
- **Cost**(s,s')=1 for all steps. (weighted later)
- The search problem: find a solution path from a state in **I** to a state in **G**.
  - Optionally minimize the cost of the solution.



# A directed graph in state space



# Search examples

- 8-puzzle

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

# Search examples

- 8-puzzle

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

- States = 3x3 array configurations

# Search examples

- 8-puzzle

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

- States = 3x3 array configurations
- Actions / Operators = up to 4 kinds of movement

# Search examples

- 8-puzzle

7	2	4
5		6
8	3	1

Start State

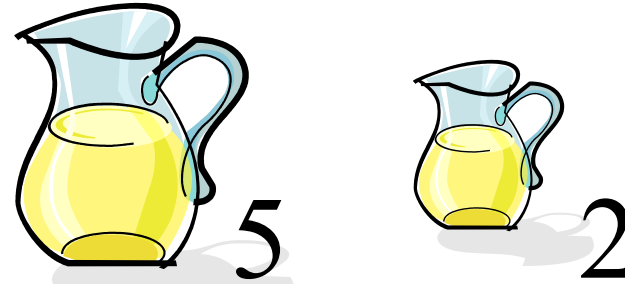
	1	2
3	4	5
6	7	8

Goal State

- States = 3x3 array configurations
- Actions / Operators = up to 4 kinds of movement
- Cost = 1 for each move

# Search examples

- Water jugs: how to get 1?



State =  $(x,y)$ , where  $x$  = number of gallons of water in the 5-gallon jug and  $y$  is gallons in the 2-gallon jug

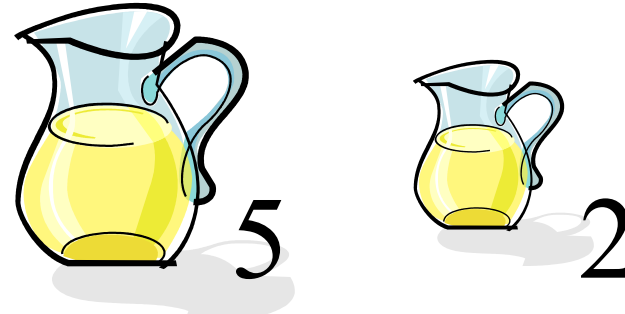
Initial State =  $(5,0)$

Goal State =  $(*,1)$ , where  $*$  means any amount



# Search examples

- Water jugs: how to get 1?



State =  $(x,y)$ , where  $x$  = number of gallons of water in the 5-gallon jug and  $y$  is gallons in the 2-gallon jug

Initial State =  $(5,0)$

Goal State =  $(*,1)$ , where  $*$  means any amount

Operators

$(x,y) \rightarrow (0,y)$  ; empty 5-gal jug

$(x,y) \rightarrow (x,0)$  ; empty 2-gal jug

$(x,2)$  and  $x \leq 3 \rightarrow (x+2,0)$  ; pour 2-gal into 5-gal

$(x,0)$  and  $x \geq 2 \rightarrow (x-2,2)$  ; pour 5-gal into 2-gal

$(1,0) \rightarrow (0,1)$  ; empty 5-gal into 2-gal

# Search examples

- Route finding (State? Successors? Cost weighted)

Google Maps - from: 1210 W Dayton St, Madison, WI 53706 to: State St, Madison, WI 53703 - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

Google Maps [Maps](#) [Local Search](#) [Directions](#)

1210 W Dayton St, Madison, WI 53706 Start address

State St, Madison, WI 53703 End address

Search Help

Maps

Map Satellite

Print Email Link to this page

Start address: 1210 W Dayton St  
Madison, WI 53706

End address: State St  
Madison, WI 53703

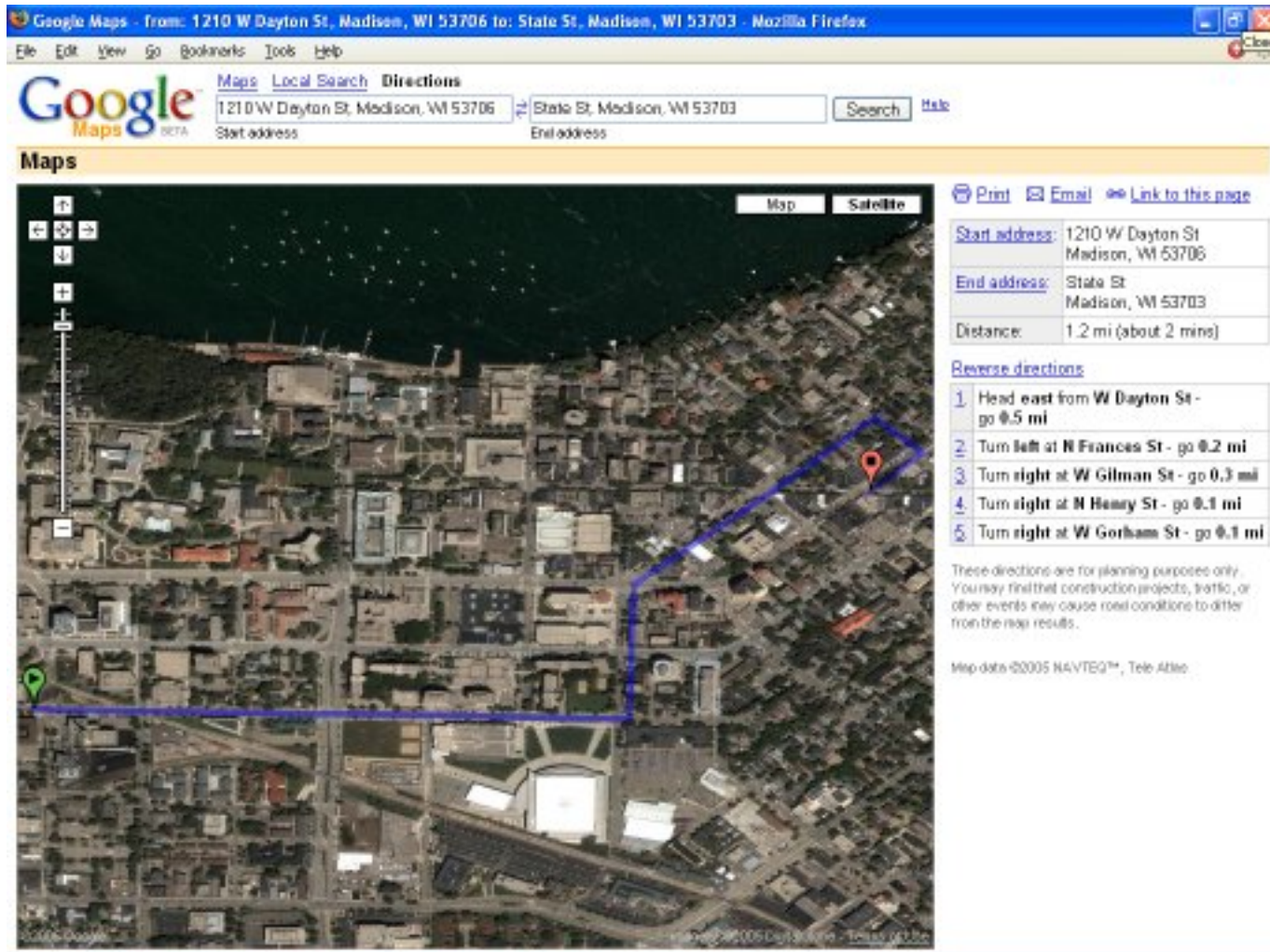
Distance: 1.2 mi (about 2 mins)

[Reverse directions](#)

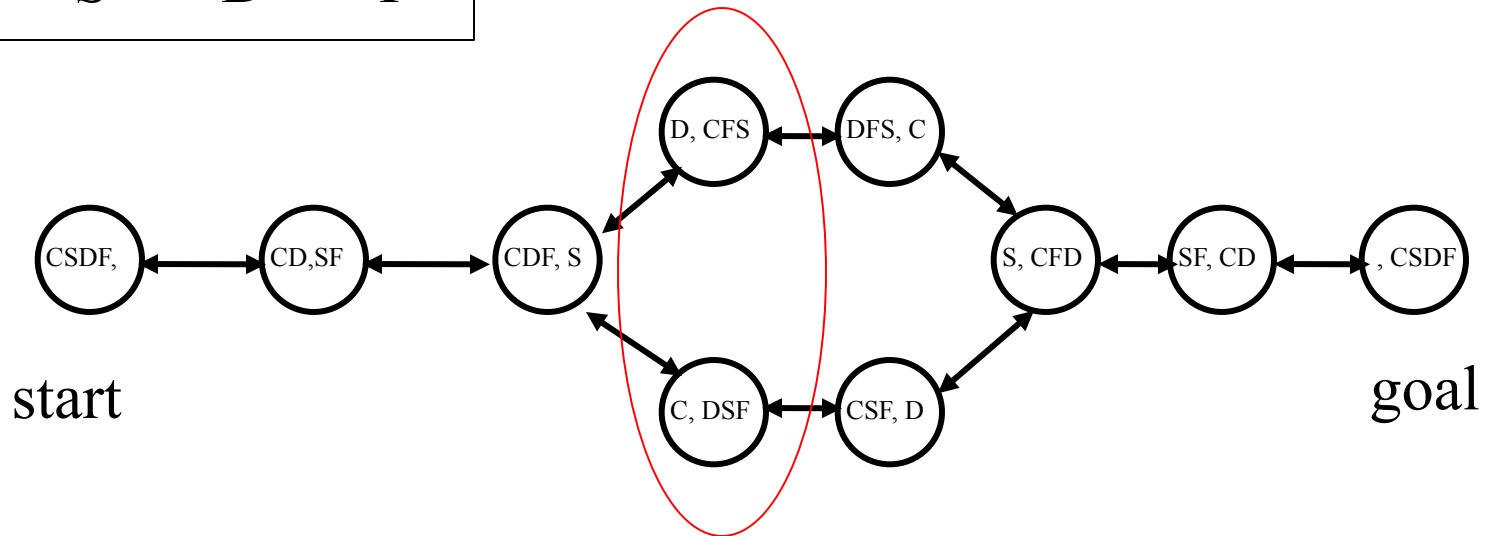
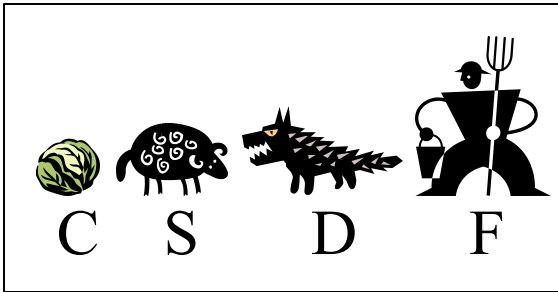
- 1 Head east from W Dayton St - go 0.5 mi
- 2 Turn left at N Frances St - go 0.2 mi
- 3 Turn right at W Gilman St - go 0.3 mi
- 4 Turn right at N Henry St - go 0.1 mi
- 5 Turn right at W Gorham St - go 0.1 mi

These directions are for planning purposes only. You may find that construction projects, traffic, or other events may cause road conditions to differ from the map results.

Map data ©2005 NAVTEQ™, Tele Atlas



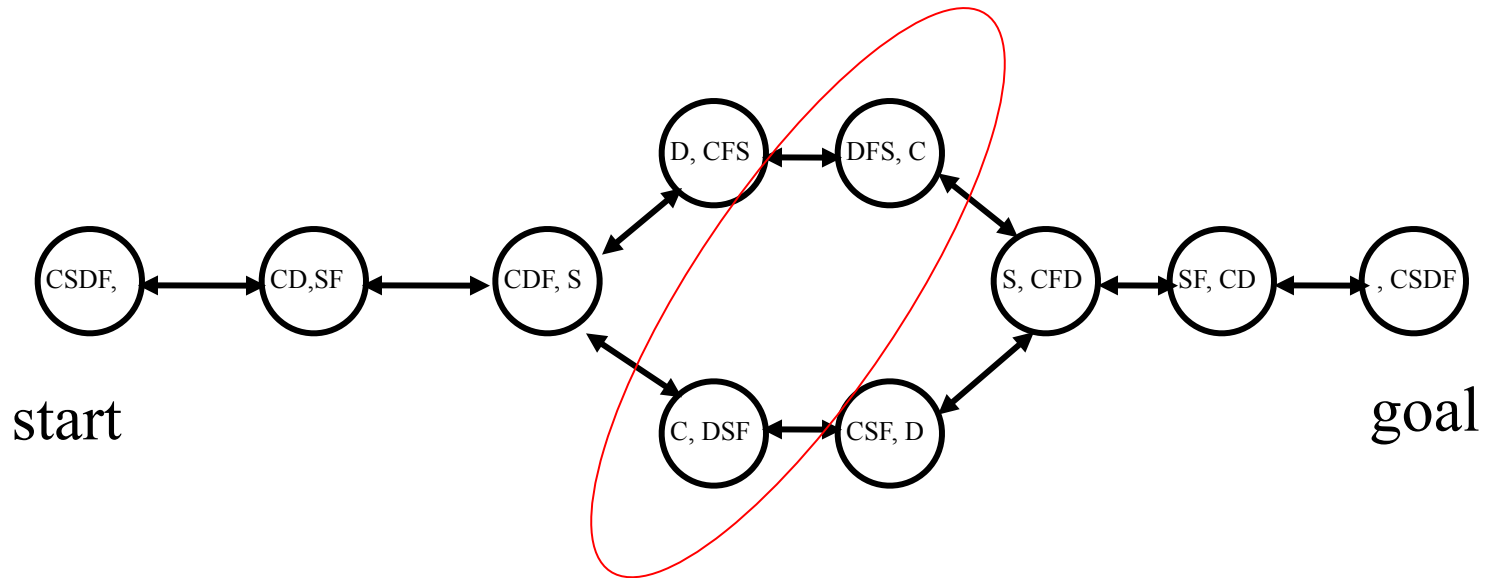
# A directed graph in state space



- In general there will be many generated, but un-expanded states at any given time
- One has to choose which one to expand next

# Different search strategies

- The generated, but not yet expanded states form the **fringe** (**OPEN**).
- The essential difference is **which one to expand first**.
- Deep or shallow?



# Uninformed search on trees

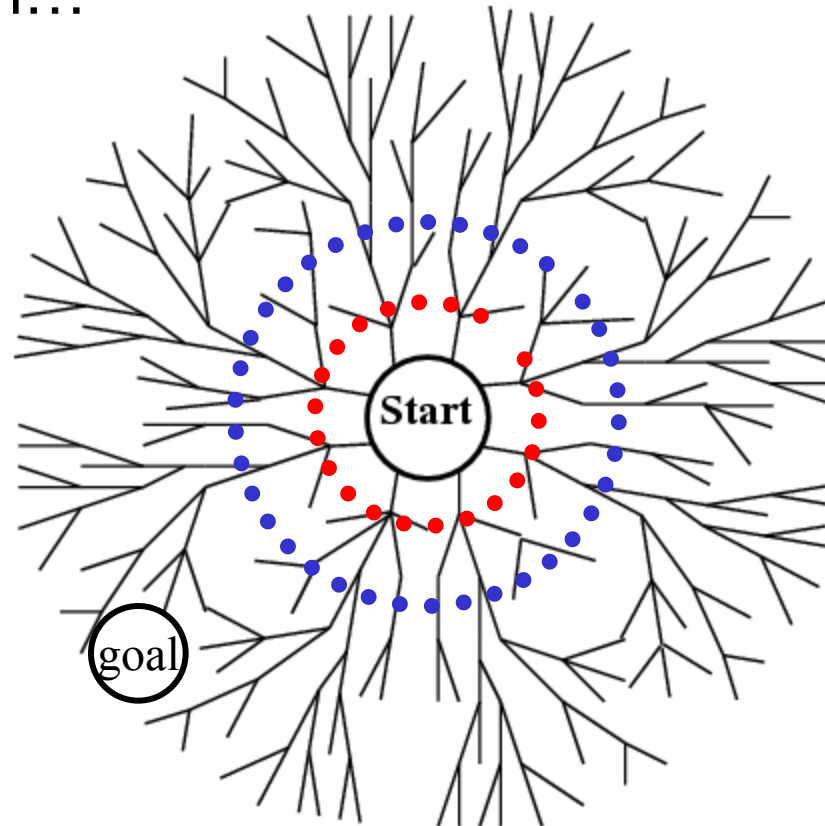
- **Uninformed** means we only know:
  - The goal test
  - The *succs()* function
- But **not** which non-goal states are better: that would be informed search (next topic).
- For now, we also assume *succs()* graph is **a tree**.
  - Won't encounter repeated states.
  - We will relax it later.
- Many search strategies:
  - We will see BFS, UCS, DFS, IDS
- Differ by what un-expanded nodes to expand

# Breadth-first search (BFS)

Expand the shallowest node first

- Examine states **one** step away from the initial states
- Examine states **two** steps away from the initial states
- and so on...

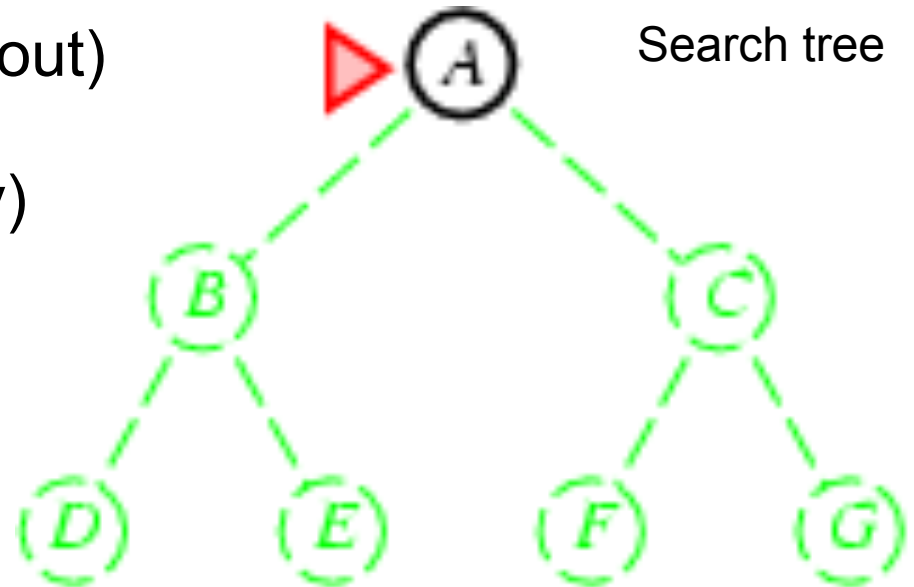
ripple



# Breadth-first search (BFS)

Use a **queue** (First-in First-out)

1. en\_queue(Initial states)
2. While (queue not empty)
3.   s = de\_queue()
4.   if (s==goal) success!
5.   T = succs(s)
6.   en\_queue(T)
7. endwhile



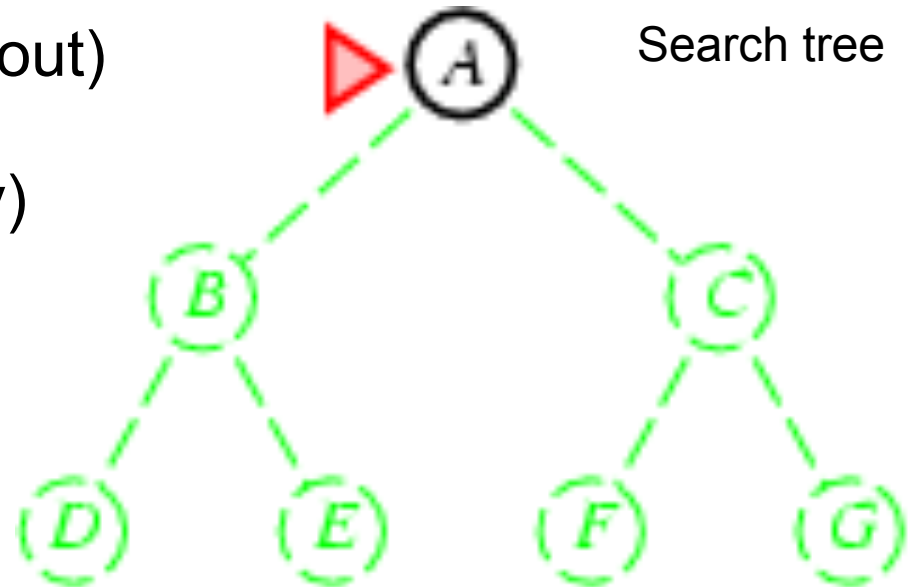
Initial state: **A**

Goal state: **G**

# Breadth-first search (BFS)

Use a **queue** (First-in First-out)

1. en\_queue(Initial states)
2. While (queue not empty)
3.   s = de\_queue()
4.   if (s==goal) success!
5.   T = succs(s)
6.   en\_queue(T)
7. endwhile



queue (**fringe**, **OPEN**)  
→ [A] →

Initial state: **A**

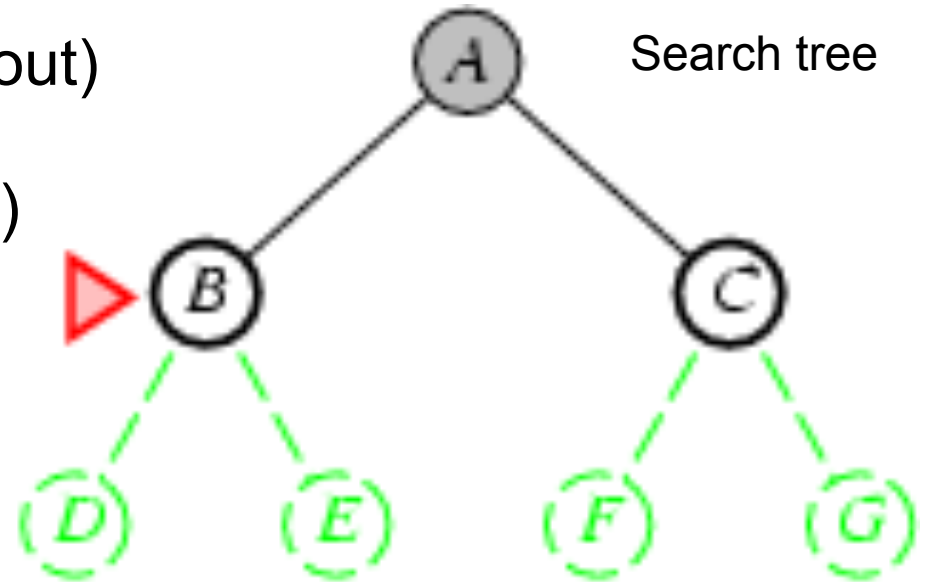
Goal state: **G**



# Breadth-first search (BFS)

Use a **queue** (First-in First-out)

1. en\_queue(Initial states)
2. While (queue not empty)
3.   s = de\_queue()
4.   if (s==goal) success!
5.   T = succs(s)
6.   en\_queue(T)
7. endwhile



queue (**fringe**, **OPEN**)  
→ [CB] → A

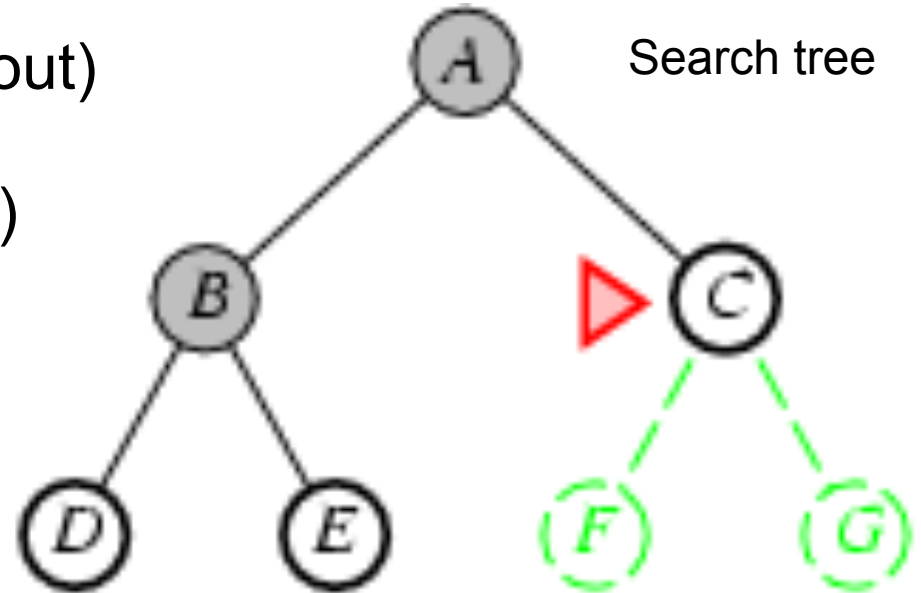
Initial state: **A**

Goal state: **G**

# Breadth-first search (BFS)

Use a **queue** (First-in First-out)

1. en\_queue(Initial states)
2. While (queue not empty)
3.   s = de\_queue()
4.   if (s==goal) success!
5.   T = succs(s)
6.   en\_queue(T)
7. endwhile



queue (**fringe**, **OPEN**)  
→ [EDC] → B

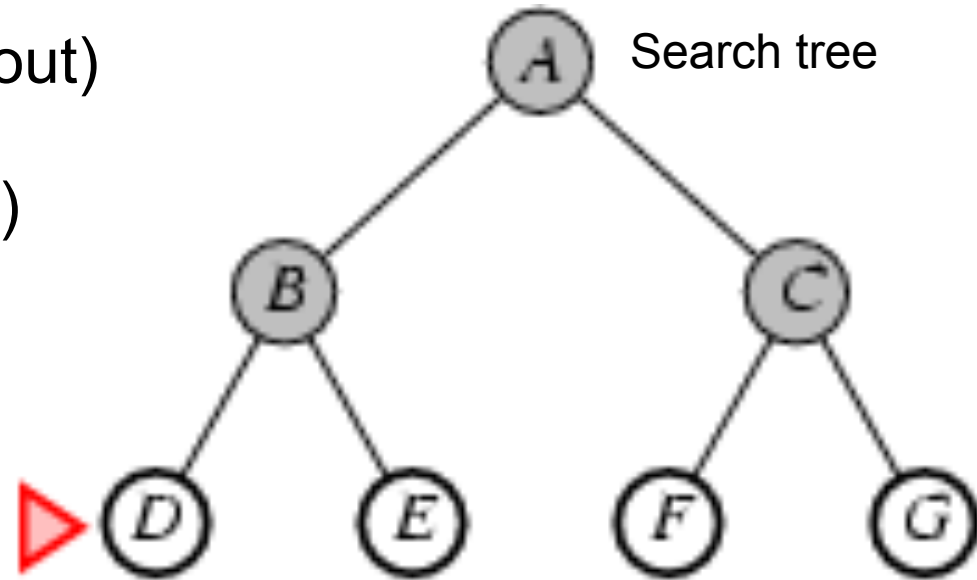
Initial state: **A**

Goal state: **G**

# Breadth-first search (BFS)

Use a **queue** (First-in First-out)

1. en\_queue(Initial states)
2. While (queue not empty)
3.   s = de\_queue()
4.   if (s==goal) success!
5.   T = succs(s)
6.   en\_queue(T)
7. endwhile



queue (**fringe**, **OPEN**)  
→[GFED] → C

Initial state: **A**

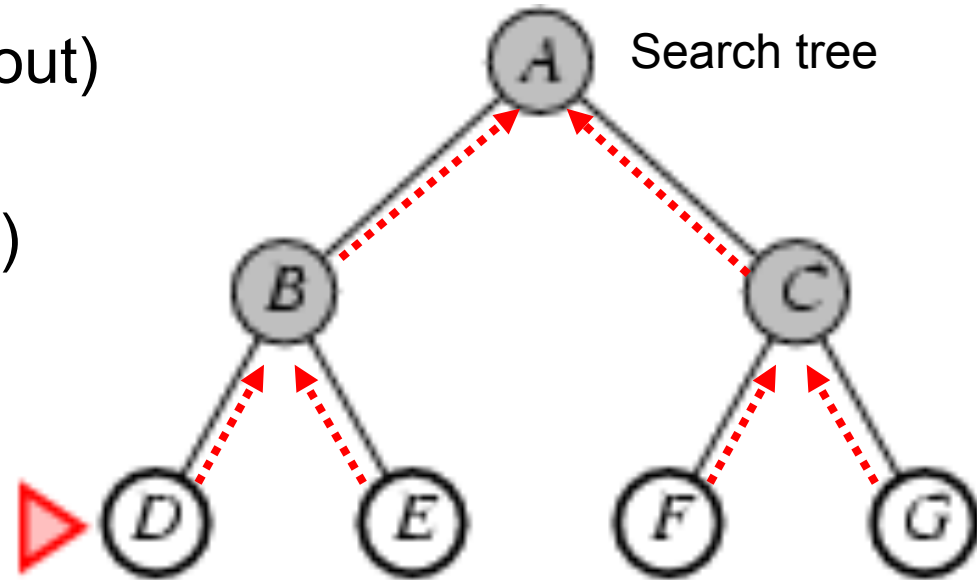
Goal state: **G**

If G is a goal, we've seen it, but we don't stop!

# Breadth-first search (BFS)

Use a **queue** (First-in First-out)

1. en\_queue(Initial states)
2. While (queue not empty)
3.   s = de\_queue()
4.   if (s==goal) success!
5.   T = succs(s)
6.   en\_queue(T)
7. endwhile



queue  
→[] →G

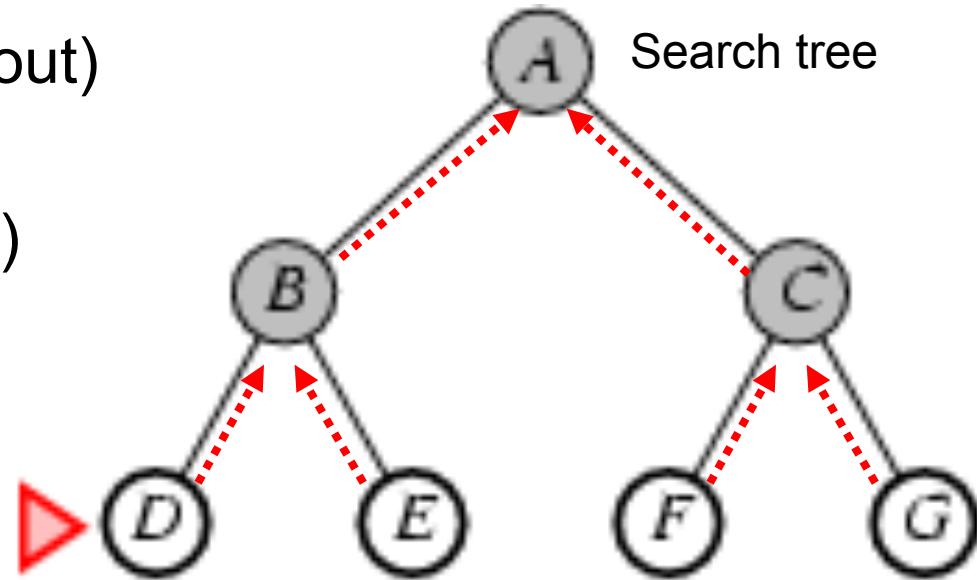
... until much later we pop G.

Looking foolish?  
Indeed. But let's be  
consistent...

# Breadth-first search (BFS)

Use a **queue** (First-in First-out)

1. en\_queue(Initial states)
2. While (queue not empty)
3.   s = de\_queue()
4.   if (s==goal) success!
5.   T = succs(s)
6.   en\_queue(T)
7. endwhile



queue  
→[] →G

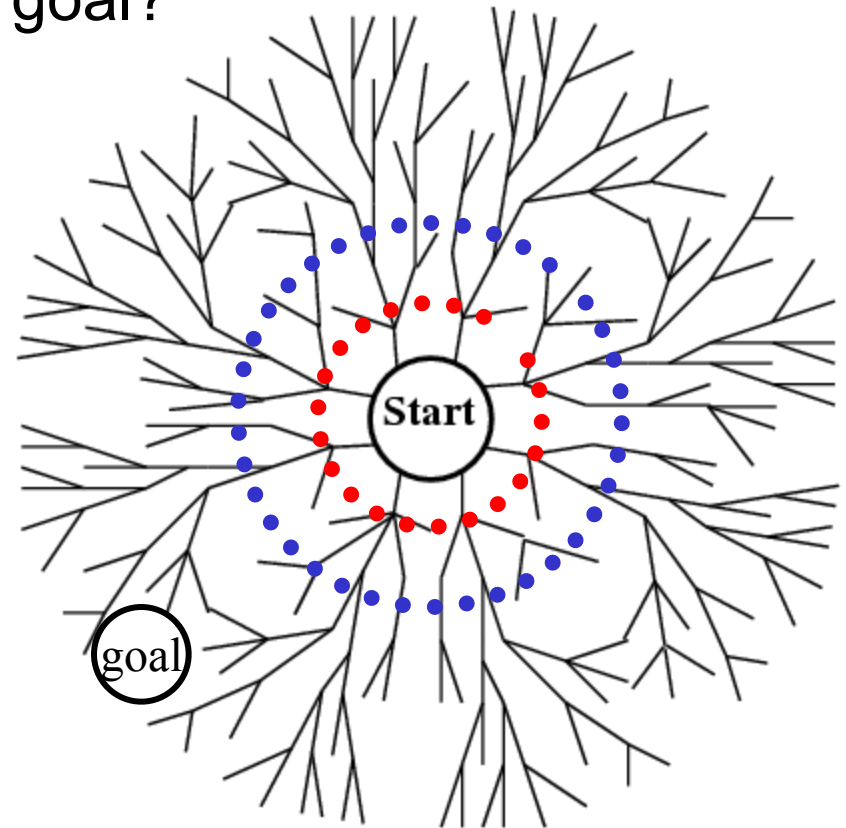
... until much later we pop G.

Looking foolish?  
Indeed. But let's be  
consistent...

We need **back pointers** to  
recover the solution path.

# Performance of BFS

- Assume:
  - the graph may be infinite.
  - Goal(s) exists and is only finite steps away.
- Will BFS find at least one goal?
- Will BFS find the least cost goal?
- Time complexity?
  - # states generated
  - Goal  $d$  edges away
  - Branching factor  $b$
- Space complexity?
  - # states stored



# Performance of BFS

# Performance of BFS

Four measures of search algorithms:



# Performance of BFS

Four measures of search algorithms:

- **Completeness** (**not** finding all goals): yes, BFS will find a goal.

# Performance of BFS

Four measures of search algorithms:

- **Completeness** (**not** finding all goals): yes, BFS will find a goal.
- **Optimality**: yes if edges cost 1 (more generally positive non-decreasing in depth), **no otherwise**.

# Performance of BFS

Four measures of search algorithms:

- **Completeness** (**not** finding all goals): yes, BFS will find a goal.
- **Optimality**: yes if edges cost 1 (more generally positive non-decreasing in depth), **no otherwise**.
- **Time** complexity (worst case): goal is the last node at radius  $d$ .

# Performance of BFS

Four measures of search algorithms:

- **Completeness** (**not** finding all goals): yes, BFS will find a goal.
- **Optimality**: yes if edges cost 1 (more generally positive non-decreasing in depth), **no otherwise**.
- **Time** complexity (worst case): goal is the last node at radius  $d$ .
  - Have to generate all nodes at radius  $d$ .

# Performance of BFS

Four measures of search algorithms:

- **Completeness** (**not** finding all goals): yes, BFS will find a goal.
- **Optimality**: yes if edges cost 1 (more generally positive non-decreasing in depth), **no otherwise**.
- **Time** complexity (worst case): goal is the last node at radius  $d$ .
  - Have to generate all nodes at radius  $d$ .
  - $b + b^2 + \dots + b^d \sim O(b^d)$

# Performance of BFS

Four measures of search algorithms:

- **Completeness** (**not** finding all goals): yes, BFS will find a goal.
- **Optimality**: yes if edges cost 1 (more generally positive non-decreasing in depth), **no otherwise**.
- **Time** complexity (worst case): goal is the last node at radius  $d$ .
  - Have to generate all nodes at radius  $d$ .
  - $b + b^2 + \dots + b^d \sim O(b^d)$
- **Space** complexity (**bad**)

# Performance of BFS

Four measures of search algorithms:

- **Completeness** (**not** finding all goals): yes, BFS will find a goal.
- **Optimality**: yes if edges cost 1 (more generally positive non-decreasing in depth), **no otherwise**.
- **Time** complexity (worst case): goal is the last node at radius  $d$ .
  - Have to generate all nodes at radius  $d$ .
  - $b + b^2 + \dots + b^d \sim O(b^d)$
- **Space** complexity (**bad**)
  - Back pointers for all generated nodes  $O(b^d)$

# Performance of BFS

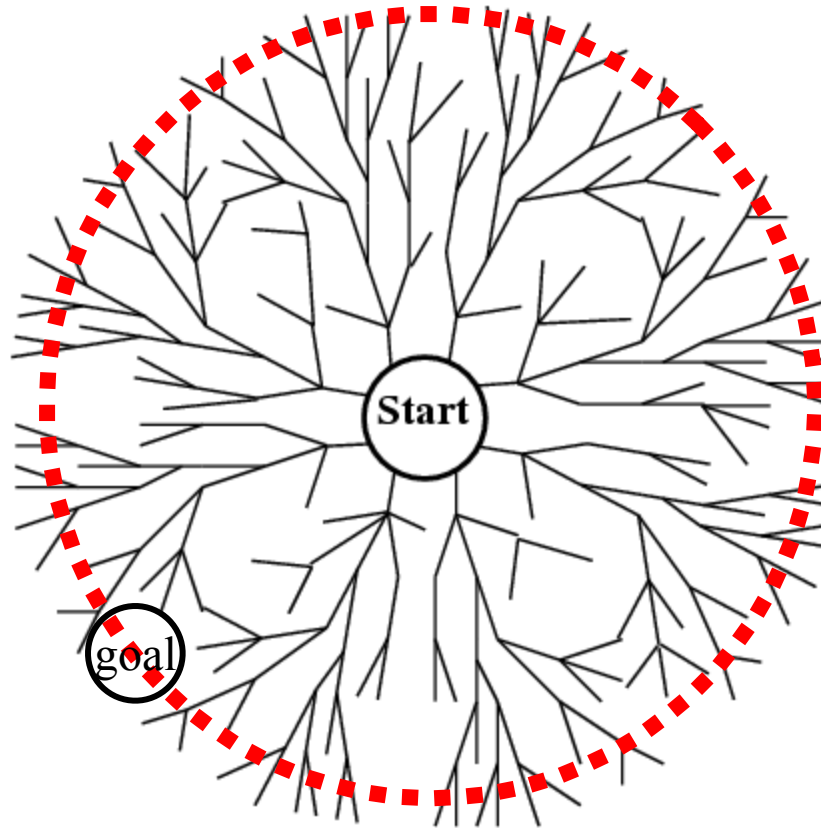
Four measures of search algorithms:

- **Completeness** (**not** finding all goals): yes, BFS will find a goal.
- **Optimality**: yes if edges cost 1 (more generally positive non-decreasing in depth), **no otherwise**.
- **Time** complexity (worst case): goal is the last node at radius  $d$ .
  - Have to generate all nodes at radius  $d$ .
  - $b + b^2 + \dots + b^d \sim O(b^d)$
- **Space** complexity (**bad**)
  - Back pointers for all generated nodes  $O(b^d)$
  - The queue / fringe (smaller, but still  $O(b^d)$ )



# What's in the fringe (queue) for BFS?

- Convince yourself this is  $O(b^d)$



# Performance of search algorithms on trees

b: branching factor (assume finite)    d: goal depth

	Complete	optimal	time	space
Breadth-first search	Y	Y, if 1	$O(b^d)$	$O(b^d)$

1. Edge cost constant, or positive non-decreasing in depth

**Q1-1: You are running BFS on a finite tree-structured state space graph that does not have a goal state. What is the behavior of BFS?**

1. Visit all  $N$  nodes, then return one at random
2. Visit all  $N$  nodes, then stop and return “failure”
3. Visit all  $N$  nodes, then return the node farthest from the initial state
4. Get stuck in an infinite loop

**Q1-1: You are running BFS on a finite tree-structured state space graph that does not have a goal state. What is the behavior of BFS?**

1. Visit all N nodes, then return one at random
2. Visit all N nodes, then stop and return “failure”
3. Visit all N nodes, then return the node farthest from the initial state
4. Get stuck in an infinite loop



# Performance of BFS

Four measures of search algorithms:

- **Completeness** (**not** finding all goals): yes, BFS will find a goal.
- **Optimality**: yes if edges cost 1 (more generally positive non-decreasing in depth), **no otherwise**.
- **Time** complexity (worst case): goal is the last node at radius  $d$ .
  - Have to generate all nodes at radius  $d$ .
  - $b + b^2 + \dots + b^d \sim O(b^d)$
- **Space** complexity (**bad**)
  - Back pointers for all generated nodes  $O(b^d)$
  - The queue / fringe (smaller, but still  $O(b^d)$ )

# Performance of BFS

Four measures of search algorithm

**Solution:**  
**Uniform-cost**  
**search**

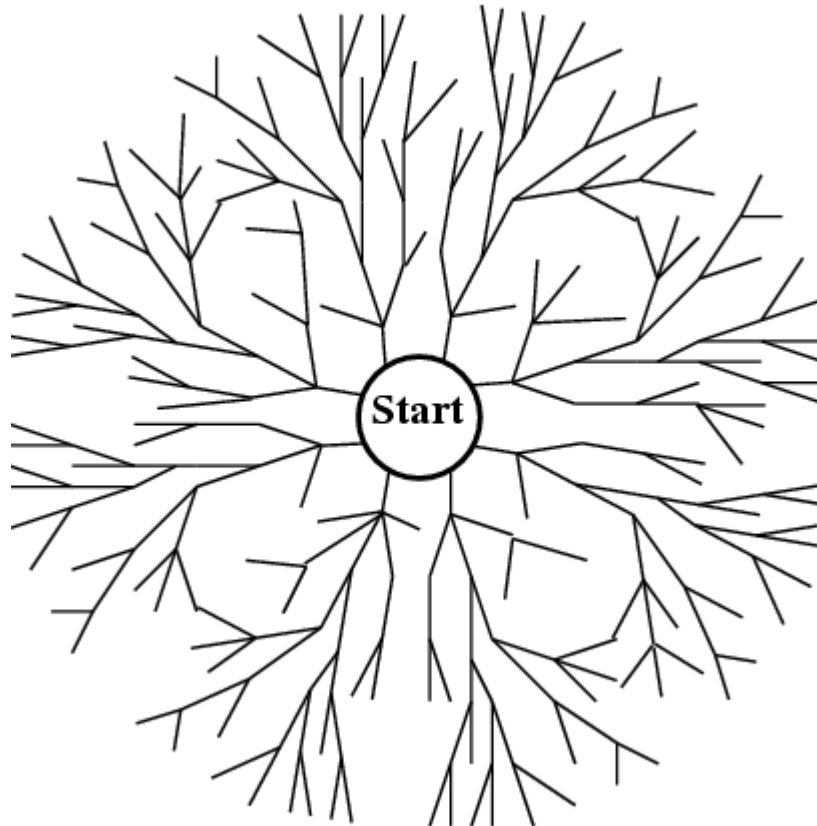
- **Completeness** (**not** finding all goals will find a goal).
- **Optimality**: yes if edges cost 1 (most generally positive non-decreasing in depth), **no otherwise**.
- **Time** complexity (worst case): goal is the last node at radius  $d$ .
  - Have to generate all nodes at radius  $d$ .
  - $b + b^2 + \dots + b^d \sim O(b^d)$
- **Space** complexity (**bad**)
  - Back pointers for all generated nodes  $O(b^d)$
  - The queue / fringe (smaller, but still  $O(b^d)$ )

# Uniform-cost search

- Find the least-cost goal
- Each node has a path cost from start (= sum of edge costs along the path).
- Expand the least cost node first.
- Use a **priority queue** instead of a normal queue
  - Always take out the least cost item

# Uniform-cost search (UCS)

- Complete and optimal (if edge costs  $\geq \epsilon > 0$ )
- Time and space: can be much worse than BFS
  - Let  $C^*$  be the cost of the least-cost goal
  - $O(b^{C^*/\epsilon})$



goal



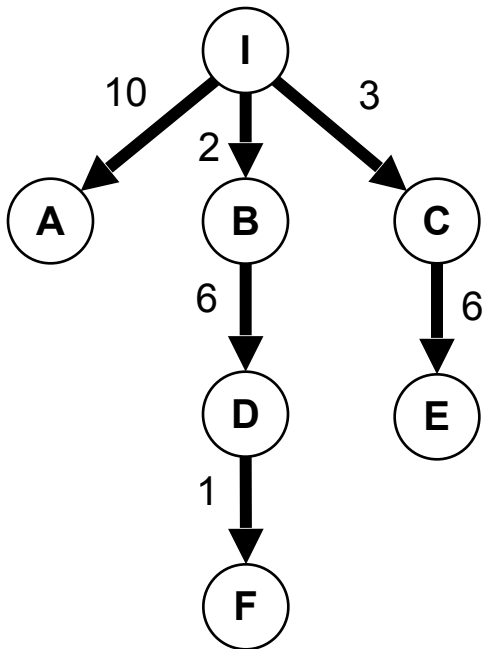
# Performance of search algorithms on trees

b: branching factor (assume finite)    d: goal depth

	Complete	optimal	time	space
Breadth-first search	Y	Y, if <sup>1</sup>	$O(b^d)$	$O(b^d)$
Uniform-cost search <sup>2</sup>	Y	Y	$O(b^{C^*/\epsilon})$	$O(b^{C^*/\epsilon})$

1. edge cost constant, or positive non-decreasing in depth
2. edge costs  $\geq \epsilon > 0$ .  $C^*$  is the best goal path cost.

**Q1-2: You are running UCS in the state space graph below. You just called the successor function on node D. What is the cost of node F?**



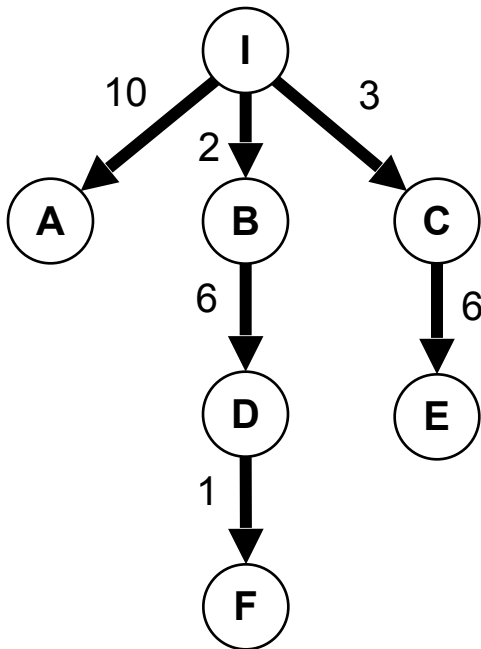
1. 2

2. 7

3. 8

4. 9

**Q1-2: You are running UCS in the state space graph below. You just called the successor function on node D. What is the cost of node F?**



1. 2

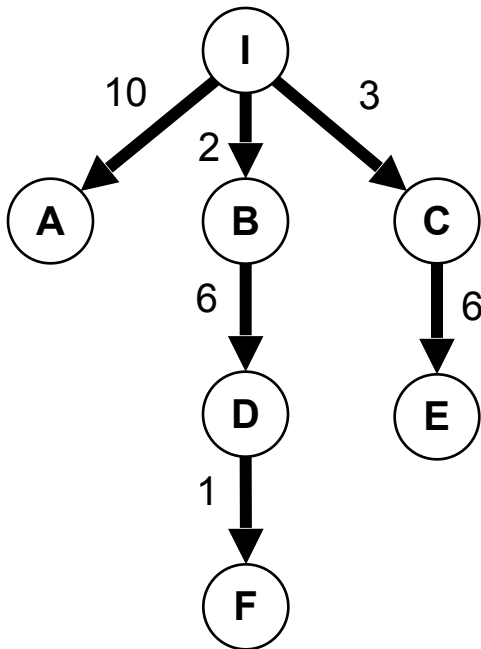
2. 7

3. 8

4. 9



**Q1-3: You are running UCS in the state space graph below. You just expanded (visited) node C. What node will the search expand next?**



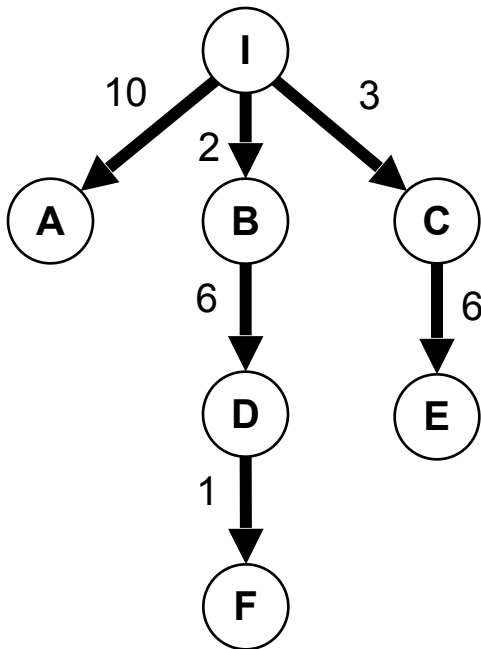
1. A

2. D

3. E

4. F

**Q1-3: You are running UCS in the state space graph below. You just expanded (visited) node C. What node will the search expand next?**



1. A
- 2. D
3. E
4. F

# General State-Space Search Algorithm

```
function general-search(problem, QUEUEING-FUNCTION)
  # problem describes the start state, operators, goal test, and
  # operator costs
  # queueing-function is a comparator function that ranks two states
  # general-search returns either a goal node or "failure"

  nodes = MAKE-QUEUE(MAKE-NODE(problem.INITIAL-STATE))
  loop
    if EMPTY(nodes) then return "failure"
    node = REMOVE-FRONT(nodes)
    if problem.GOAL-TEST(node.STATE) succeeds then return node
    nodes = QUEUEING-FUNCTION(nodes, EXPAND(node,
                                              problem.OPERATORS))
  # succ(s)=EXPAND(s, OPERATORS)
  # Note: The goal test is NOT done when nodes are generated
  # Note: This algorithm does not detect loops
end
```

# Performance of BFS

Four measures of search algorithm

**Solution:**  
**Uniform-cost**  
**search**

- **Completeness** (**not** finding all goals will find a goal).
- **Optimality**: yes if edges cost 1 (most generally positive non-decreasing in depth), **no otherwise**.
- **Time** complexity (worst case): goal is the last node at radius  $d$ .
  - Have to generate all nodes at radius  $d$ .
  - $b + b^2 + \dots + b^d \sim O(b^d)$
- **Space** complexity (**bad**)
  - Back pointers for all generated nodes  $O(b^d)$
  - The queue / fringe (smaller, but still  $O(b^d)$ )

# Performance of BFS

Four measures of search algorithm

- **Completeness** (not finding all goals will find a goal.
- **Optimality**: yes if edges cost 1 (most generally positive non-decreasing in depth), **no otherwise**.
- **Time** complexity: goal is the last node at radius  $d$ .
  - Have to generate all nodes at radius  $d$ .
  - $b + b^2 + \dots + b^d \sim O(b^d)$
- **Space** complexity (bad)
  - Back pointers for all generated nodes  $O(b^d)$
  - The queue / fringe (smaller, but still  $O(b^d)$ )

**Solution:**  
**Uniform-cost**  
**search**

**Solution:**  
**Depth-first**  
**search**

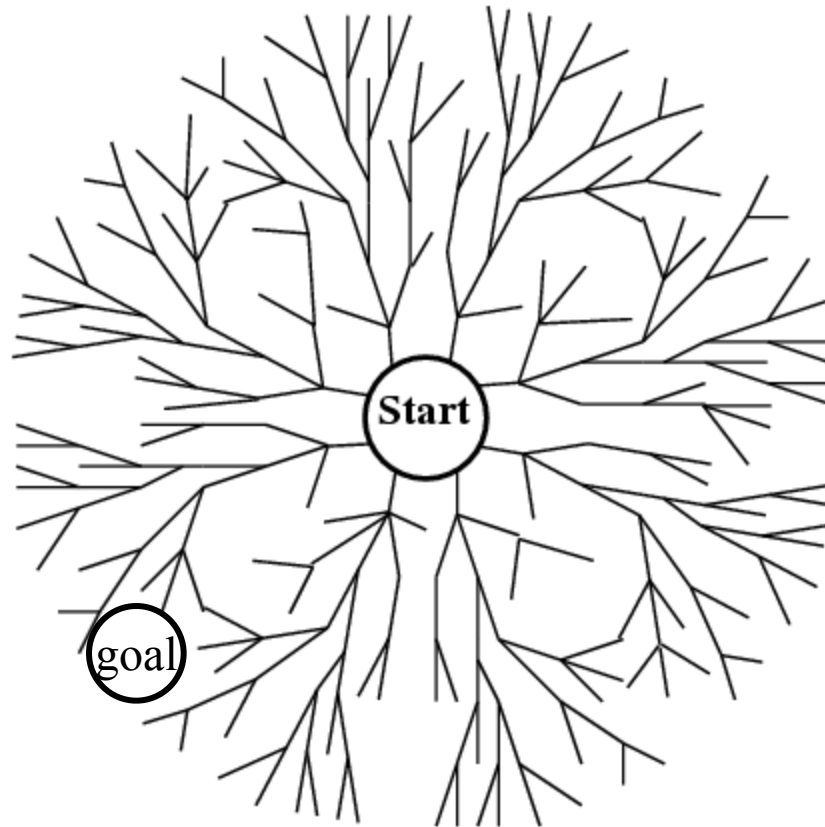


# Depth-first search

Expand the deepest node first

1. Select a direction, go deep to the end ————
2. Slightly change the end ————
3. Slightly change the end some more... ————

fan

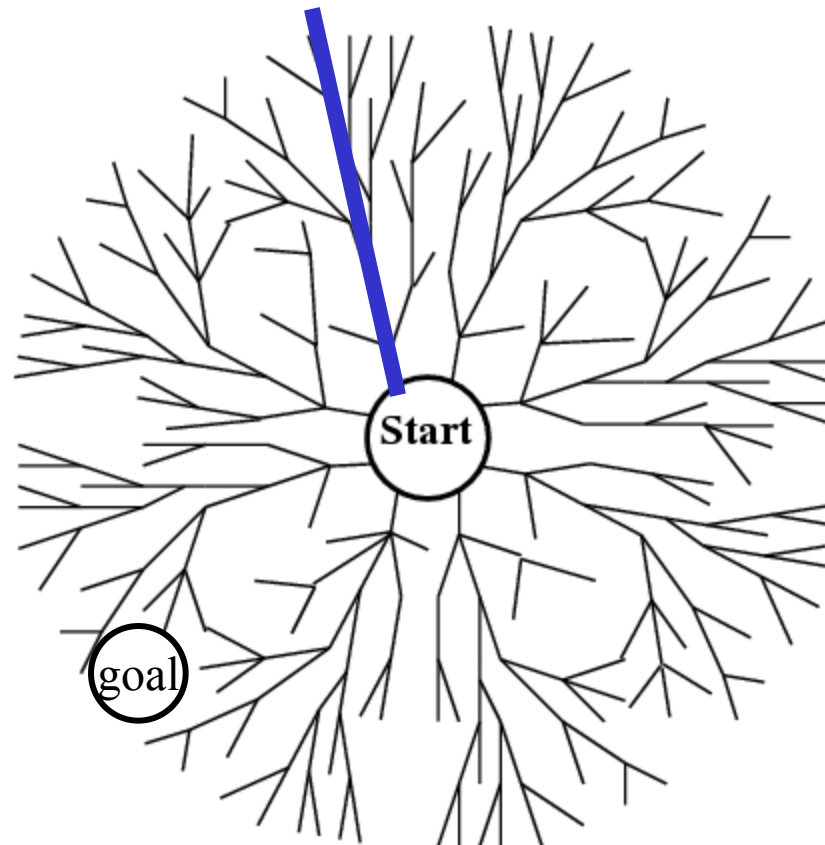


# Depth-first search

Expand the deepest node first

1. Select a direction, go deep to the end ————
2. Slightly change the end ————
3. Slightly change the end some more... ————

fan

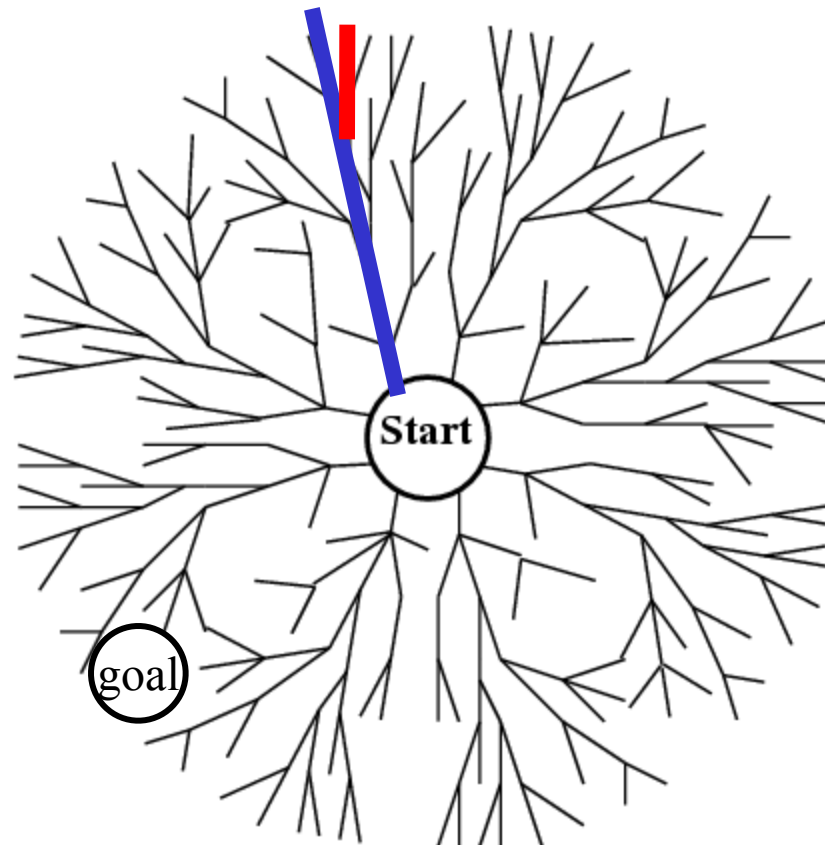


# Depth-first search

Expand the deepest node first

1. Select a direction, go deep to the end ————
2. Slightly change the end ————
3. Slightly change the end some more... ————

fan

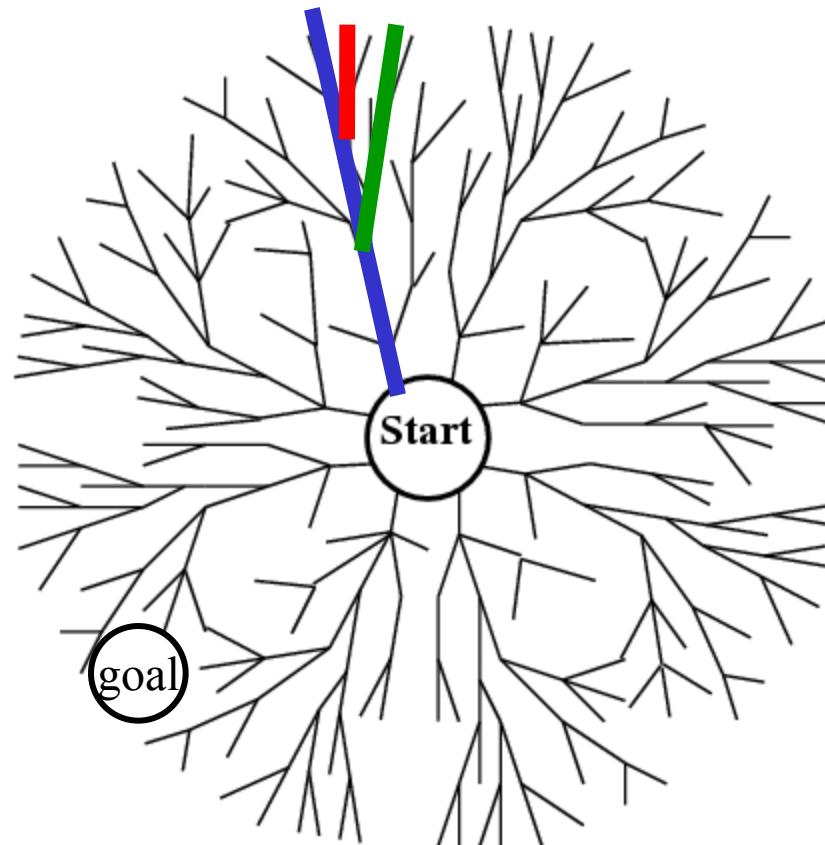


# Depth-first search

Expand the deepest node first

1. Select a direction, go deep to the end ————
2. Slightly change the end ————
3. Slightly change the end some more... ————

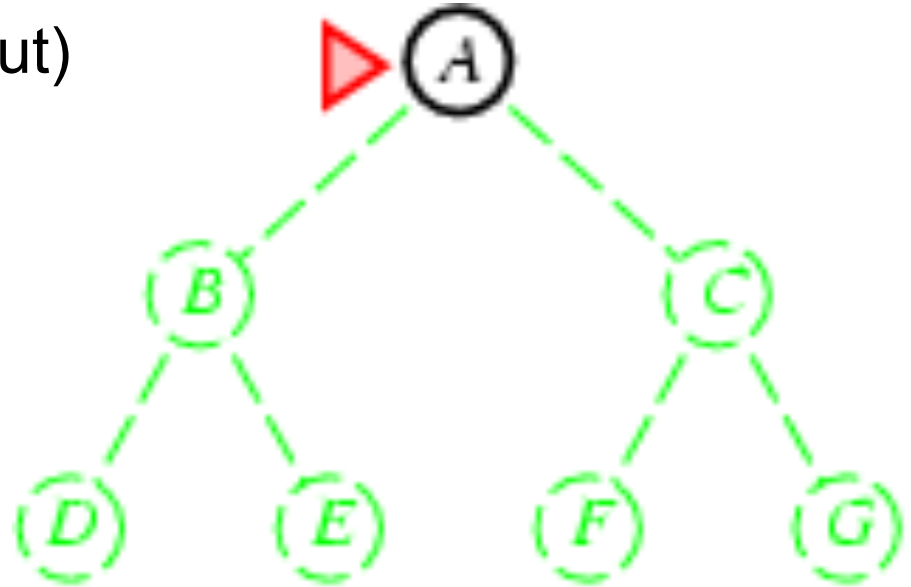
fan



# Depth-first search (DFS)

Use a **stack** (First-in Last-out)

1. push(Initial states)
2. While (stack not empty)
3.   s = pop()
4.   if (s==goal) success!
5.   T = succs(s)
6.   push(T)
7. endwhile

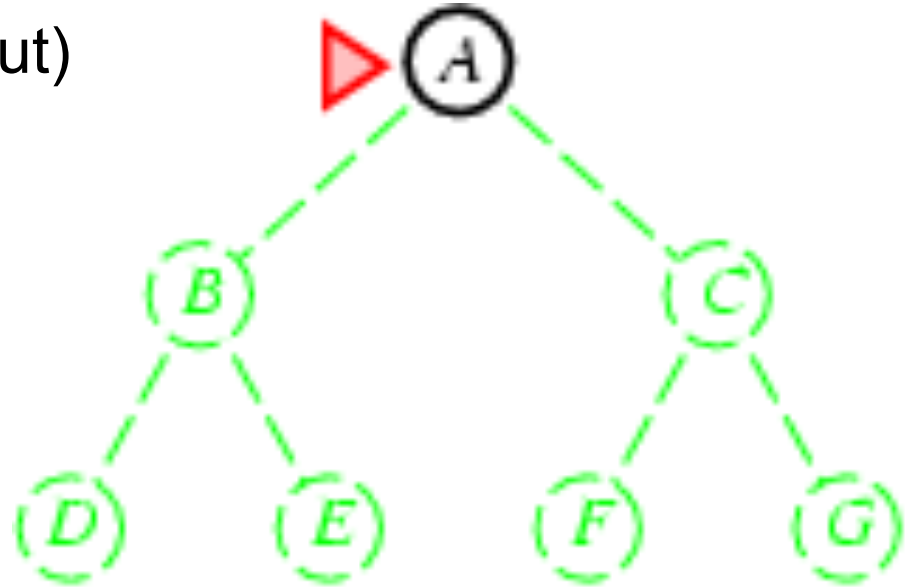


stack (**fringe**)

# Depth-first search (DFS)

Use a **stack** (First-in Last-out)

1. push(Initial states)
2. While (stack not empty)
3.   s = pop()
4.   if (s==goal) success!
5.   T = succs(s)
6.   push(T)
7. endwhile



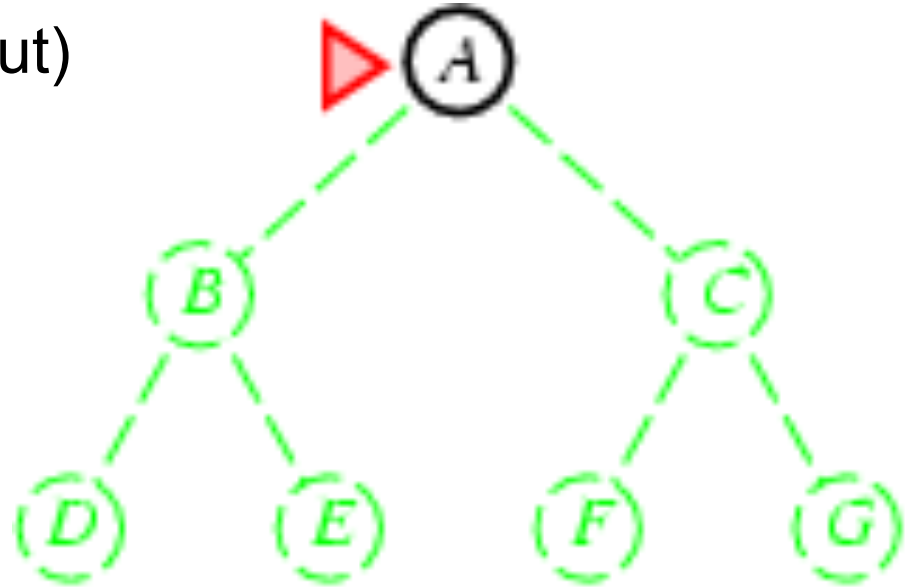
stack (**fringe**)

1. A, [B, C]

# Depth-first search (DFS)

Use a **stack** (First-in Last-out)

1. push(Initial states)
2. While (stack not empty)
3.   s = pop()
4.   if (s==goal) success!
5.   T = succs(s)
6.   push(T)
7. endwhile



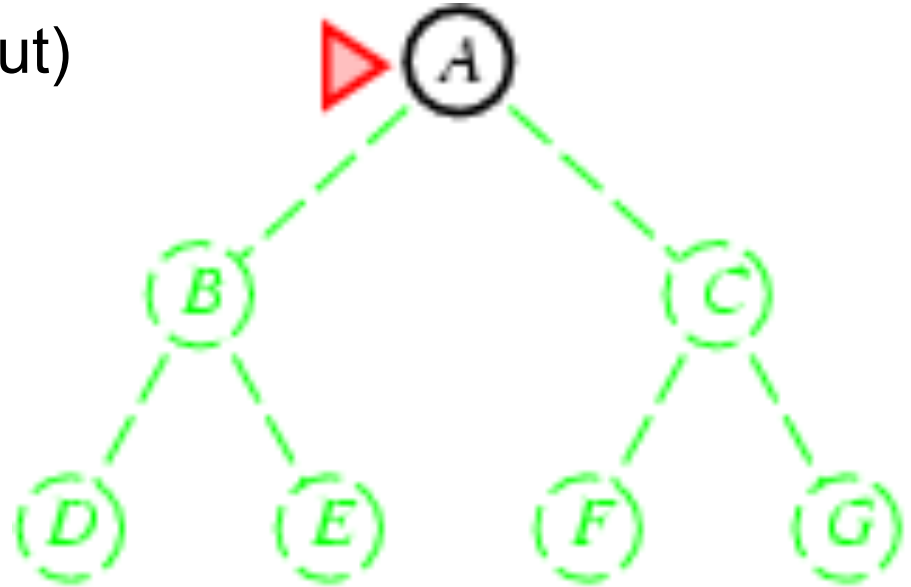
stack (**fringe**)

1. A, [B, C]
2. B, [D, E, C]

# Depth-first search (DFS)

Use a **stack** (First-in Last-out)

1. push(Initial states)
2. While (stack not empty)
3.   s = pop()
4.   if (s==goal) success!
5.   T = succs(s)
6.   push(T)
7. endwhile



stack (**fringe**)

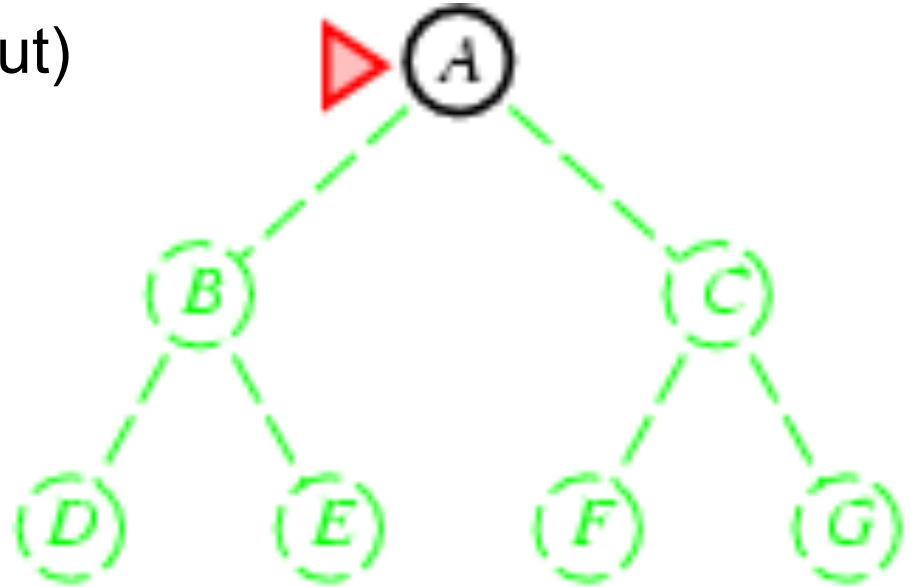
1. A, [B, C]
2. B, [D, E, C]
3. D, [E, C]



# Depth-first search (DFS)

Use a **stack** (First-in Last-out)

1. push(Initial states)
2. While (stack not empty)
3.   s = pop()
4.   if (s==goal) success!
5.   T = succs(s)
6.   push(T)
7. endwhile



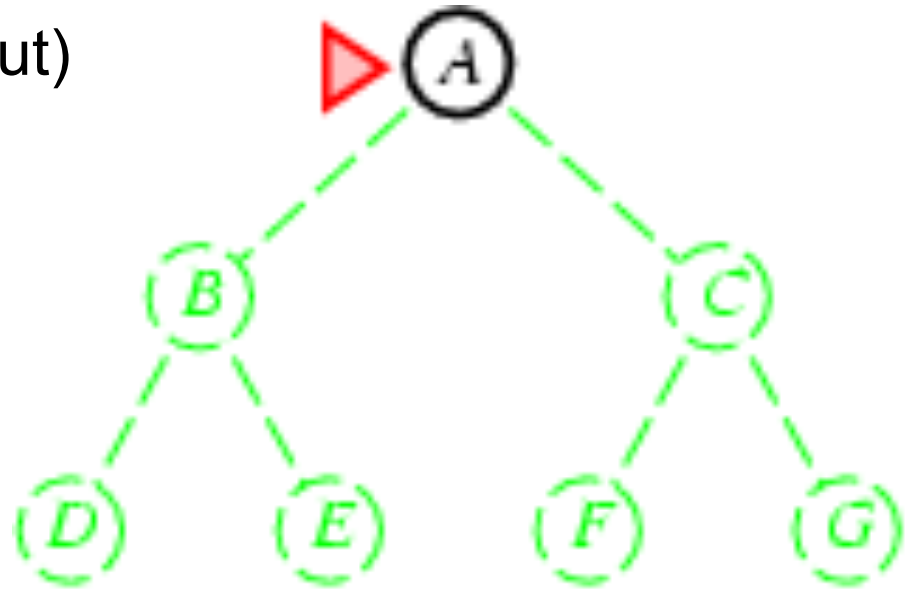
stack (**fringe**)

1. A, [B, C]
2. B, [D, E, C]
3. D, [E, C]
4. E, [C]

# Depth-first search (DFS)

Use a **stack** (First-in Last-out)

1. push(Initial states)
2. While (stack not empty)
3.   s = pop()
4.   if (s==goal) success!
5.   T = succs(s)
6.   push(T)
7. endwhile



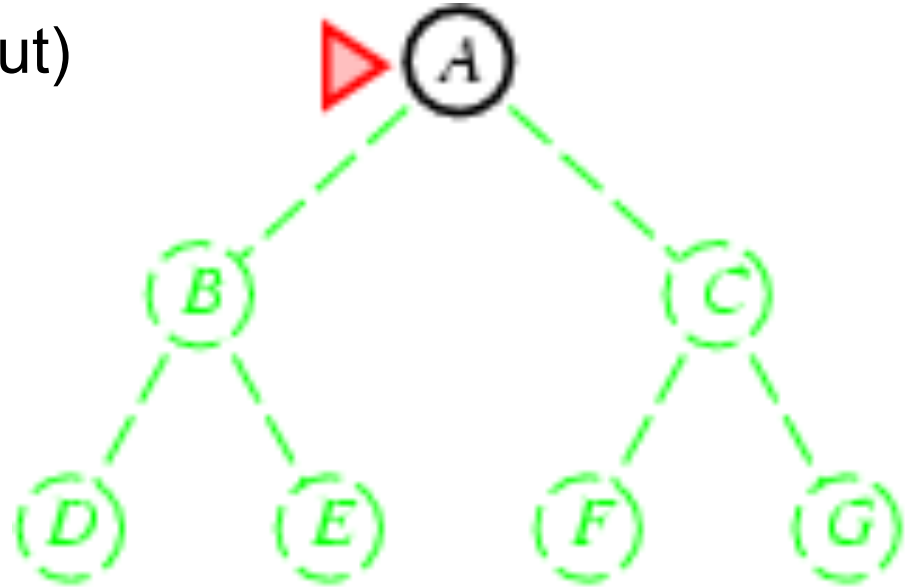
stack (**fringe**)

1. A, [B, C]
2. B, [D, E, C]
3. D, [E, C]
4. E, [C]
5. C, [F, G]

# Depth-first search (DFS)

Use a **stack** (First-in Last-out)

1. push(Initial states)
2. While (stack not empty)
3.   s = pop()
4.   if (s==goal) success!
5.   T = succs(s)
6.   push(T)
7. endwhile



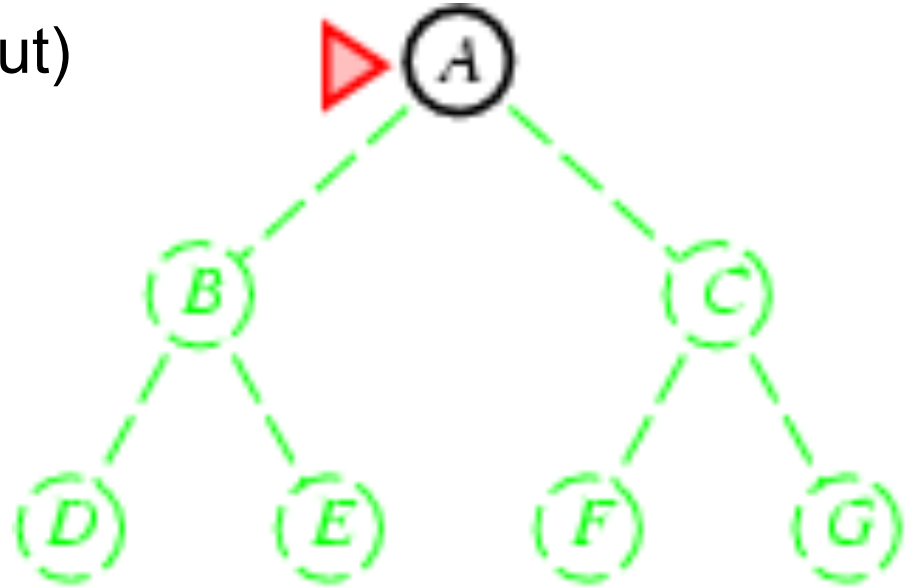
stack (**fringe**)

1. A, [B, C]
2. B, [D, E, C]
3. D, [E, C]
4. E, [C]
5. C, [F, G]
6. F, [G]

# Depth-first search (DFS)

Use a **stack** (First-in Last-out)

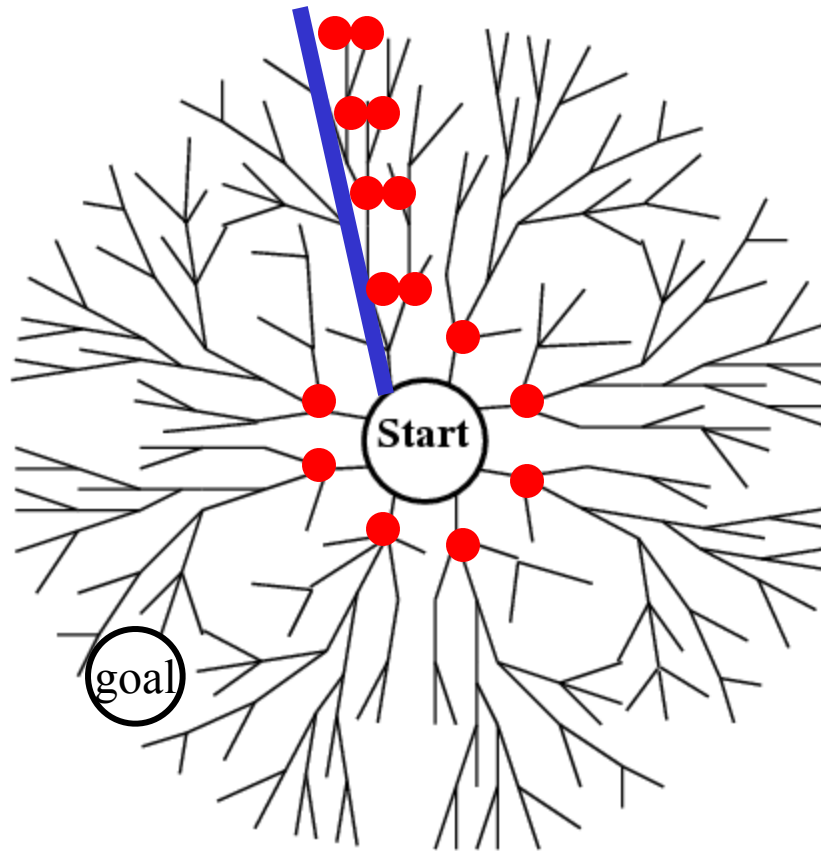
1. push(Initial states)
2. While (stack not empty)
3.   s = pop()
4.   if (s==goal) success!
5.   T = succs(s)
6.   push(T)
7. endwhile



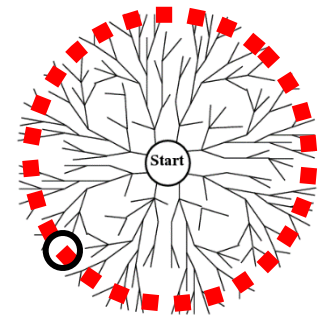
stack (**fringe**)

1. A, [B, C]
2. B, [D, E, C]
3. D, [E, C]
4. E, [C]
5. C, [F, G]
6. F, [G]
7. G

# What's in the fringe for DFS?

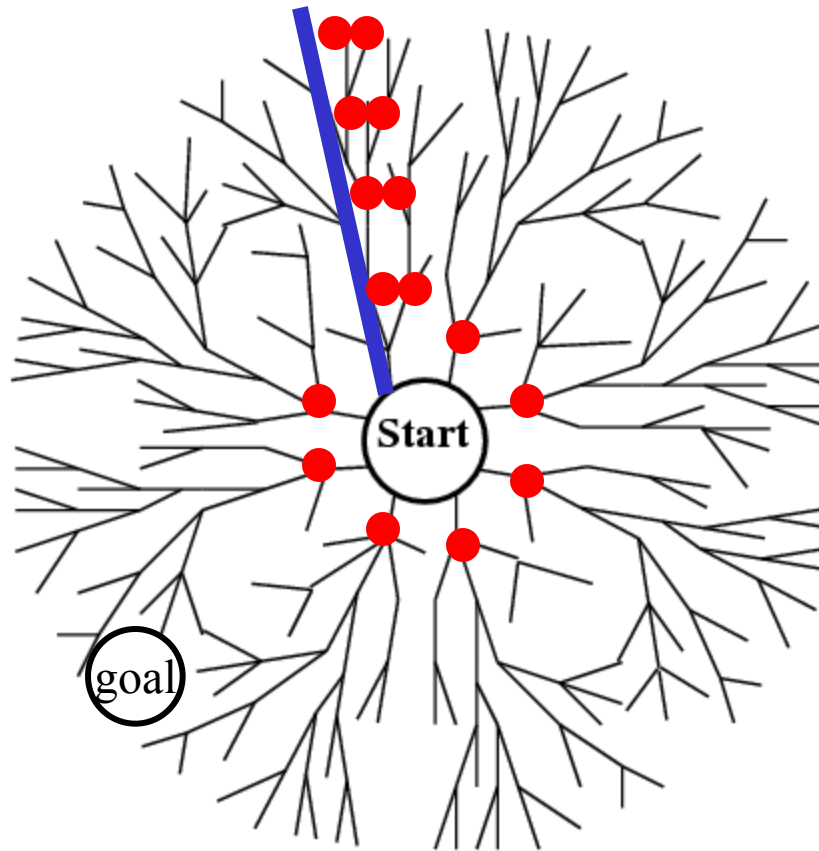


c.f. BFS  $O(b^d)$

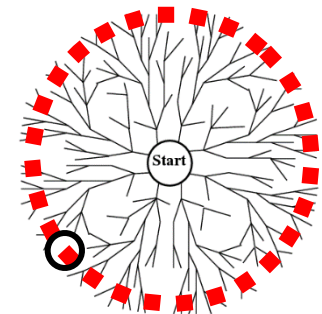


# What's in the fringe for DFS?

- m = maximum depth of graph from start

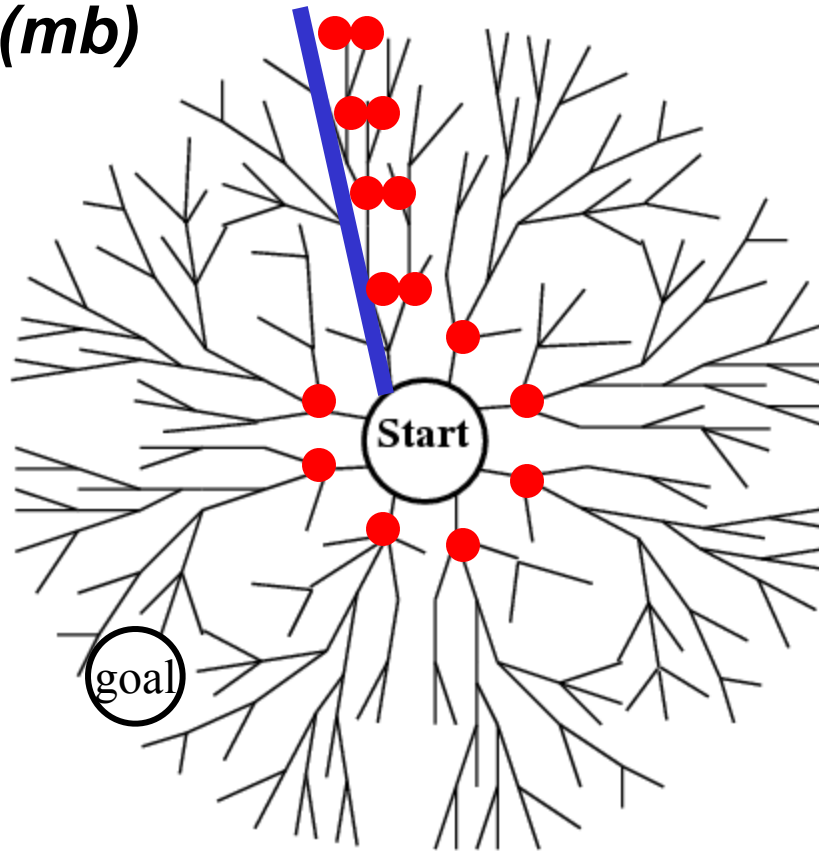


c.f. BFS  $O(b^d)$

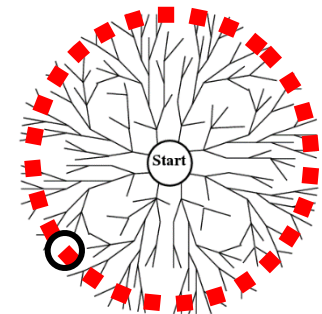


# What's in the fringe for DFS?

- $m$  = maximum depth of graph from start
- $m(b-1) \sim O(mb)$

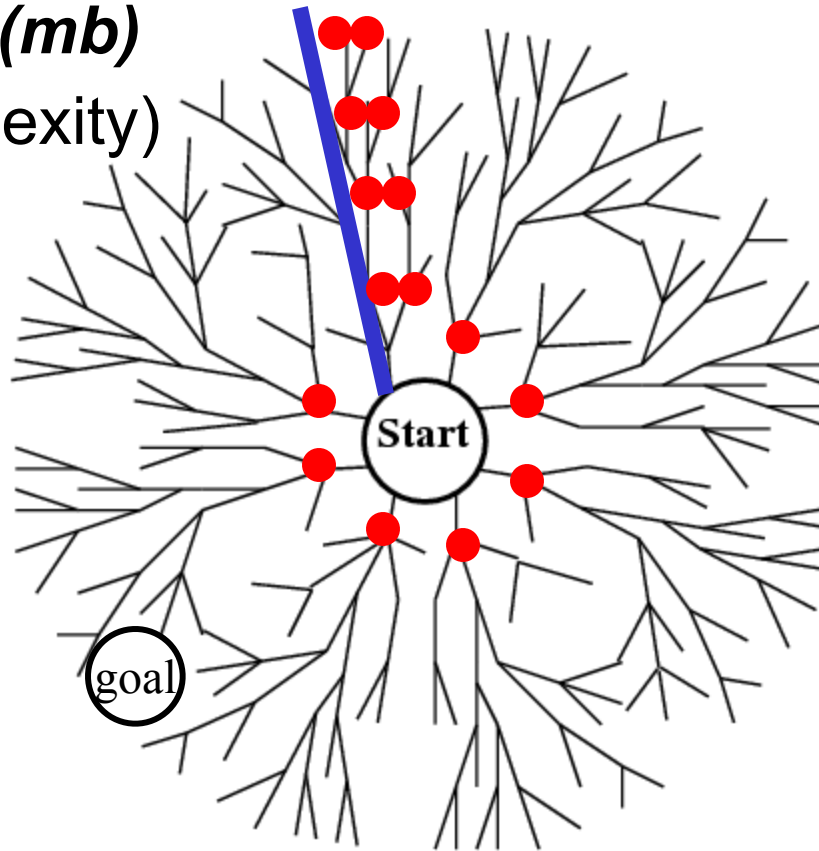


c.f. BFS  $O(b^d)$

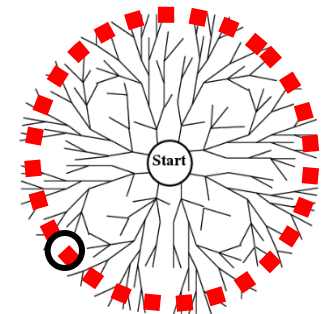


# What's in the fringe for DFS?

- $m$  = maximum depth of graph from start
- $m(b-1) \sim O(mb)$   
(Space complexity)



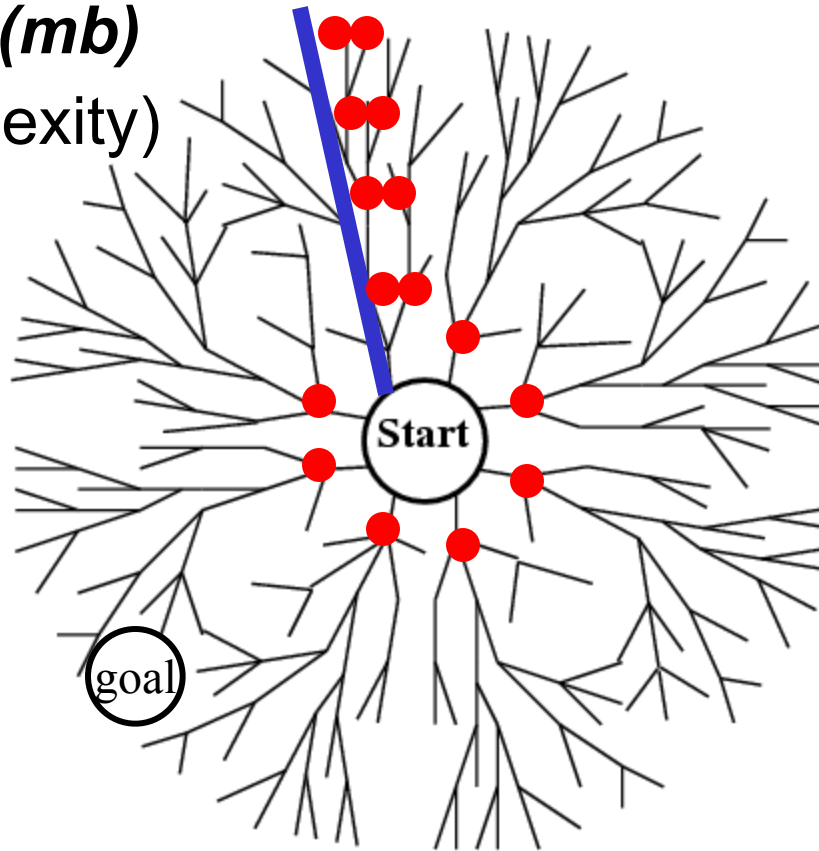
c.f. BFS  $O(b^d)$





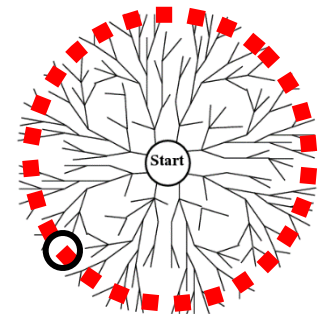
# What's in the fringe for DFS?

- $m$  = maximum depth of graph from start
- $m(b-1) \sim O(mb)$   
(Space complexity)



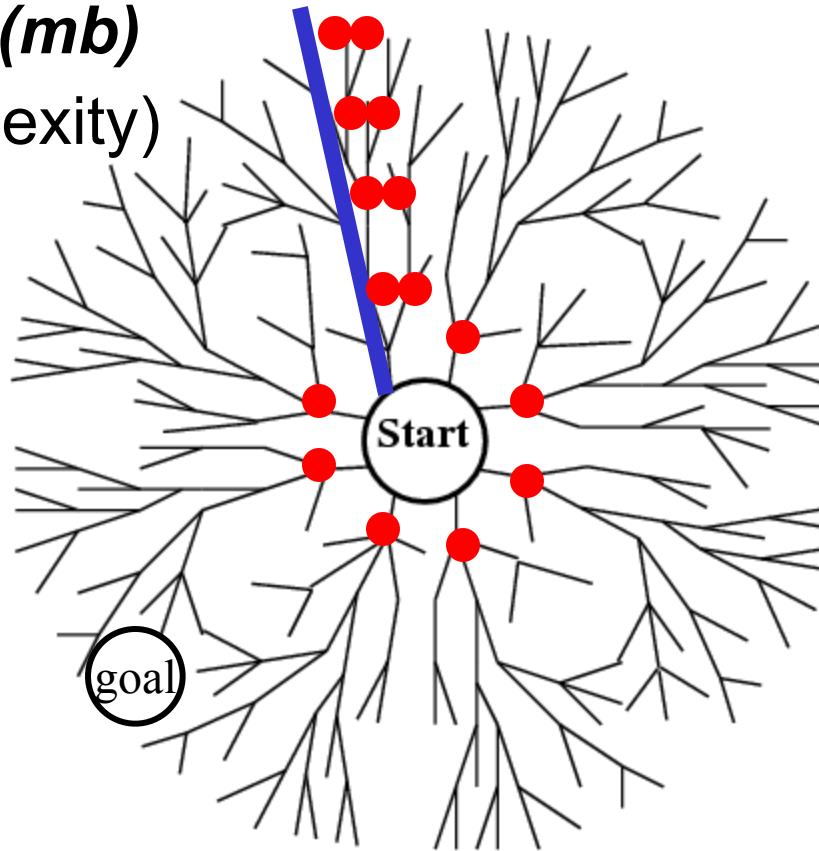
- “backtracking search” even less space

c.f. BFS  $O(b^d)$

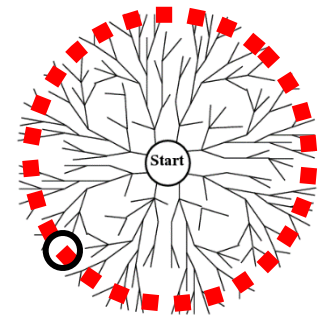


# What's in the fringe for DFS?

- $m$  = maximum depth of graph from start
- $m(b-1) \sim O(mb)$   
(Space complexity)



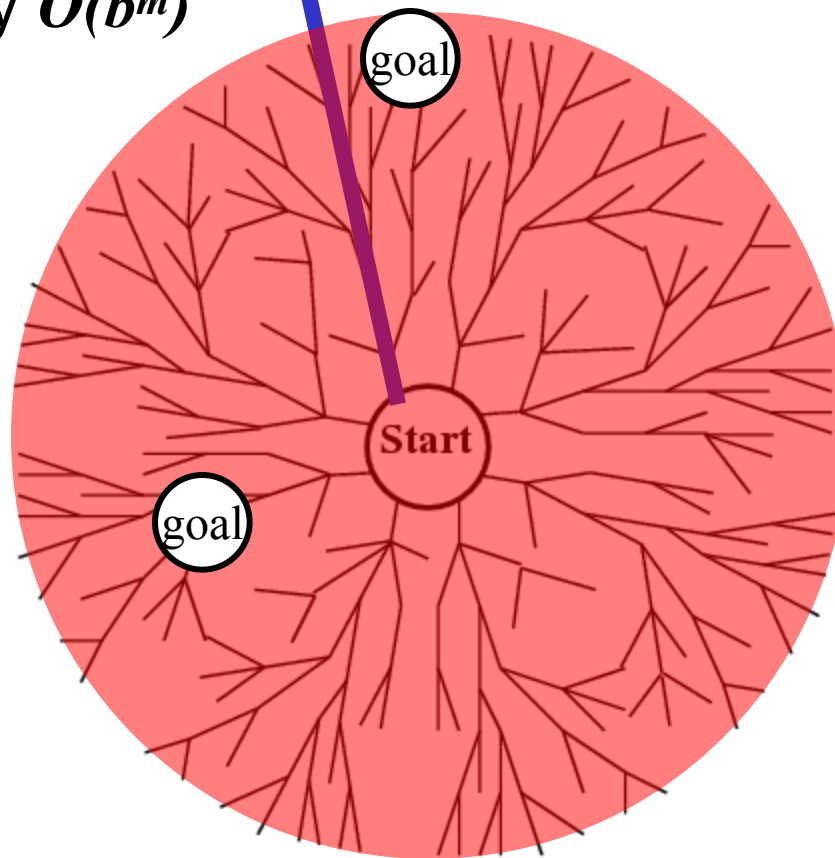
c.f. BFS  $O(b^d)$



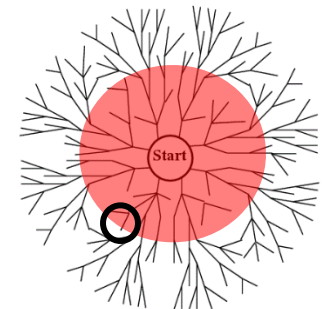
- “backtracking search” even less space
  - generate siblings (if applicable)

# What's wrong with DFS?

- Infinite tree: may not find goal (incomplete)
- May not be optimal
- Finite tree: may visit almost all nodes, time complexity  $O(b^m)$



c.f. BFS  $O(b^d)$



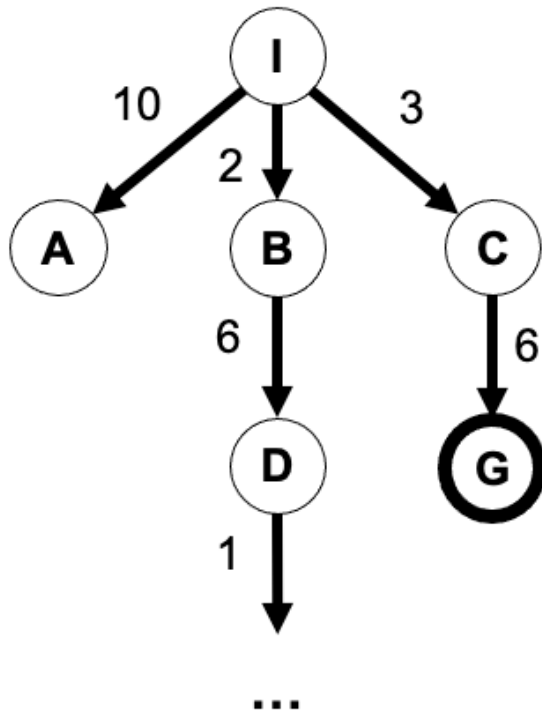
# Performance of search algorithms on trees

b: branching factor (assume finite)    d: goal depth    m: graph depth

	Complete	optimal	time	space
Breadth-first search	Y	Y, if <sup>1</sup>	$O(b^d)$	$O(b^d)$
Uniform-cost search <sup>2</sup>	Y	Y	$O(b^{C^*/\epsilon})$	$O(b^{C^*/\epsilon})$
Depth-first search	N	N	$O(b^m)$	$O(bm)$

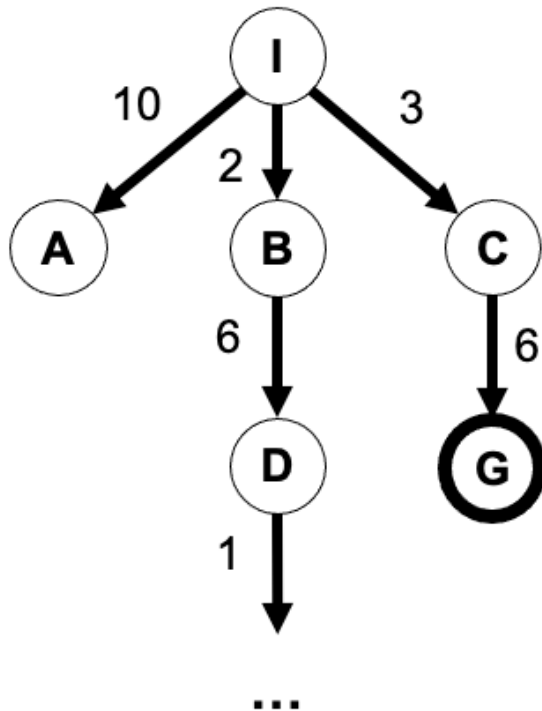
1. edge cost constant, or positive non-decreasing in depth
2. edge costs  $\geq \epsilon > 0$ .  $C^*$  is the best goal path cost.

**Q2-1: You are running DFS in the state space graph below. DFS expands nodes left to right. G is the goal state. The state space graph is infinite (the path after D does not terminate). What is the behavior of DFS?**



1. Get stuck in an infinite loop
2. Return A
3. Return G
4. Return "failure"

**Q2-1: You are running DFS in the state space graph below. DFS expands nodes left to right. G is the goal state. The state space graph is infinite (the path after D does not terminate). What is the behavior of DFS?**



1. Get stuck in an infinite loop
2. Return A
3. Return G
4. Return "failure"

**Q2-2: You need to search a randomly generated state space graph with one goal, uniform edges costs,  $d=2$ , and  $m=100$ . Considering worst case behavior, do you select BFS or DFS for your search?**

1. BFS

2. DFS

**Q2-2: You need to search a randomly generated state space graph with one goal, uniform edges costs,  $d=2$ , and  $m=100$ . Considering worst case behavior, do you select BFS or DFS for your search?**

1. BFS 

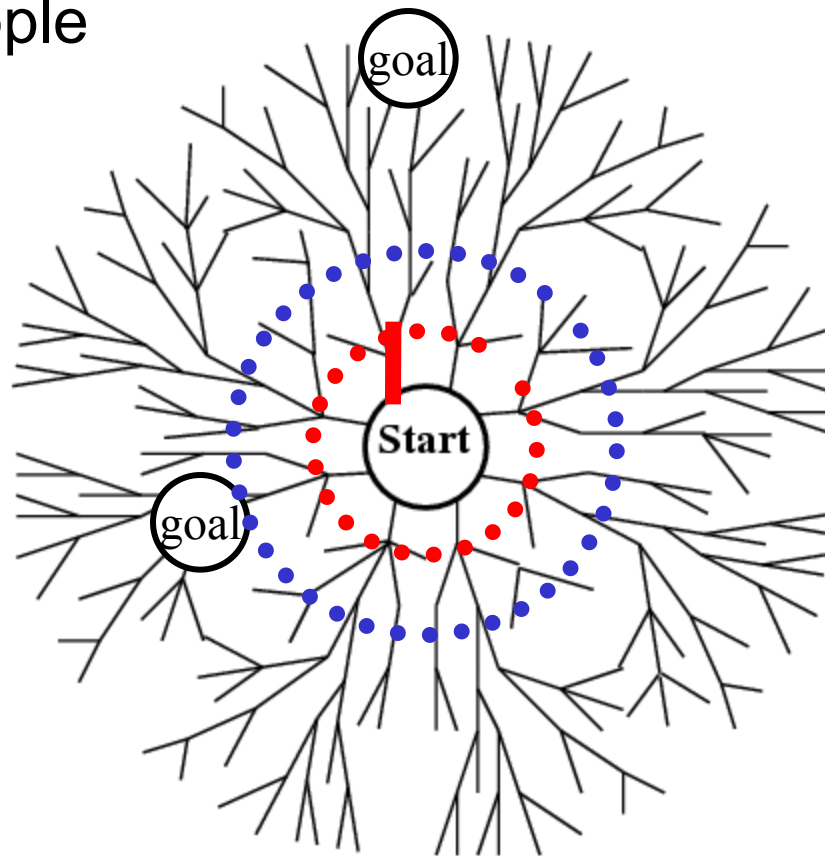
2. DFS



# How about this?

1. DFS, but stop if path length  $> 1$ .
2. If goal not found, repeat DFS, stop if path length  $> 2$ .
3. And so on...

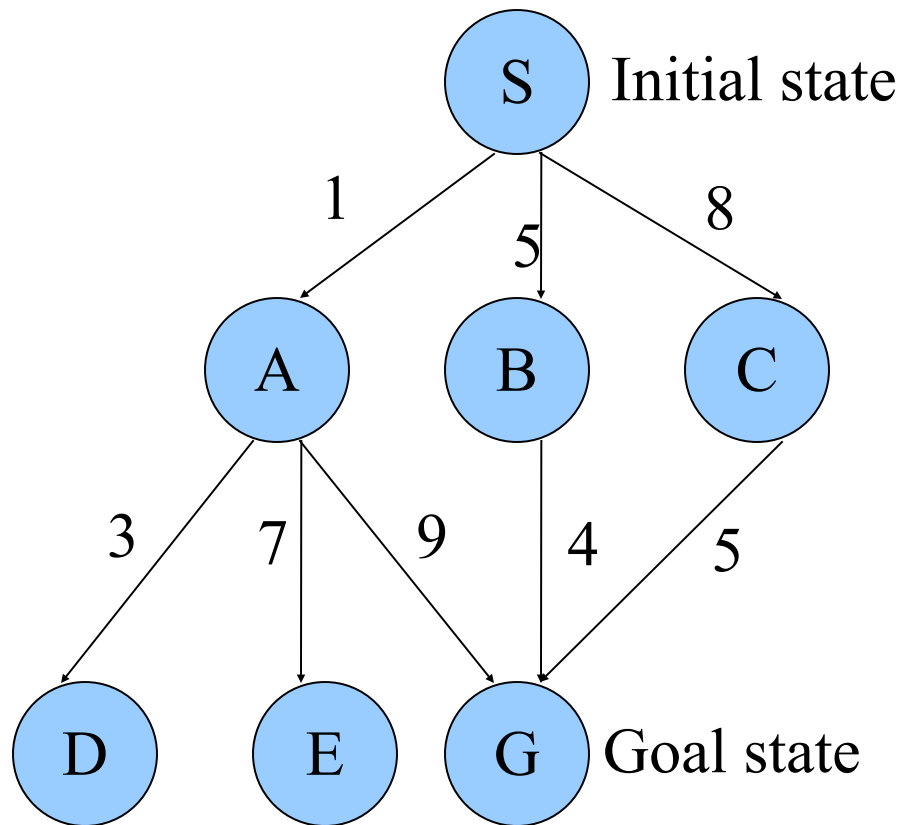
fan within ripple



## Iterative deepening

- Search proceeds like BFS, but fringe is like DFS
  - Complete, optimal like BFS
  - Small space complexity like DFS
  - Time complexity like BFS
- Preferred uninformed search method

# Example



(All edges are directed, pointing downwards)

## Nodes expanded by:

- Breadth-First Search: S A B C D E G

Solution found: S A G

- Uniform-Cost Search: S A D B C E G

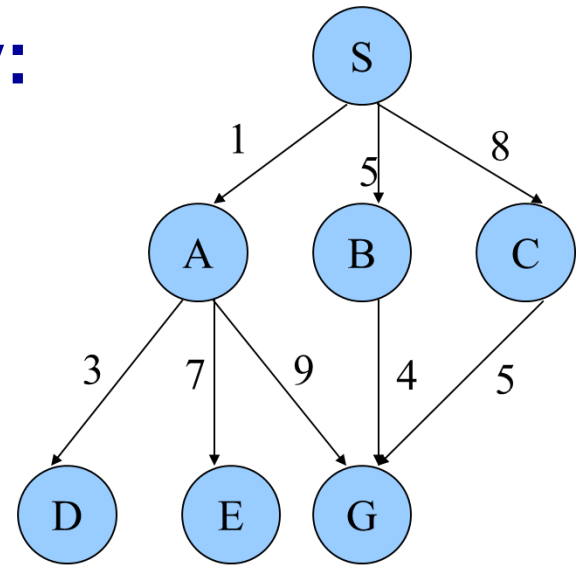
Solution found: S B G (This is the only uninformed search that worries about costs.)

- Depth-First Search: S A D E G

Solution found: S A G

- Iterative-Deepening Search: S A B C S A D E G

Solution found: S A G



# Performance of search algorithms on trees

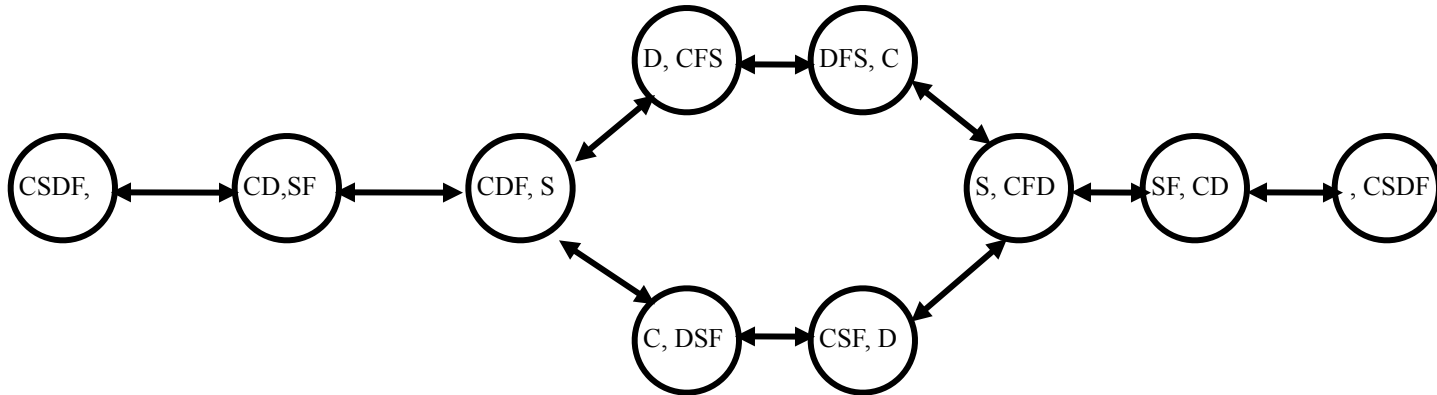
b: branching factor (assume finite)    d: goal depth    m: graph depth

	Complete	optimal	time	space
Breadth-first search	Y	Y, if <sup>1</sup>	$O(b^d)$	$O(b^d)$
Uniform-cost search <sup>2</sup>	Y	Y	$O(b^{C^*/\epsilon})$	$O(b^{C^*/\epsilon})$
Depth-first search	N	N	$O(b^m)$	$O(bm)$
Iterative deepening	Y	Y, if <sup>1</sup>	$O(b^d)$	$O(bd)$

1. edge cost constant, or positive non-decreasing in depth
2. edge costs  $\geq \epsilon > 0$ .  $C^*$  is the best goal path cost.

# If state space graph is not a tree

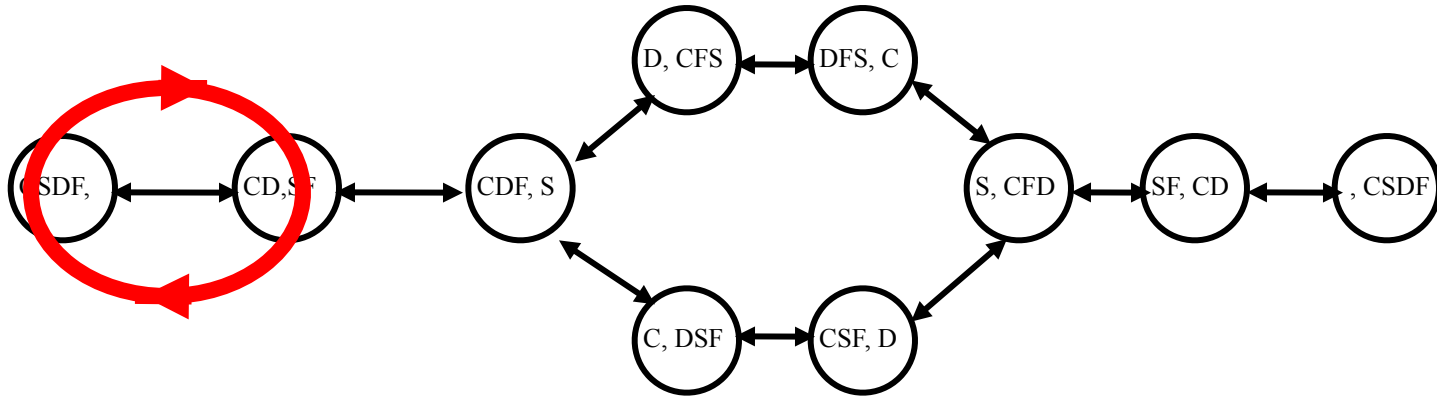
- The problem: repeated states



- Ignore the danger of repeated states: wasteful (BFS) or impossible (DFS). Can you see why?
- How to prevent it?

# If state space graph is not a tree

- The problem: repeated states



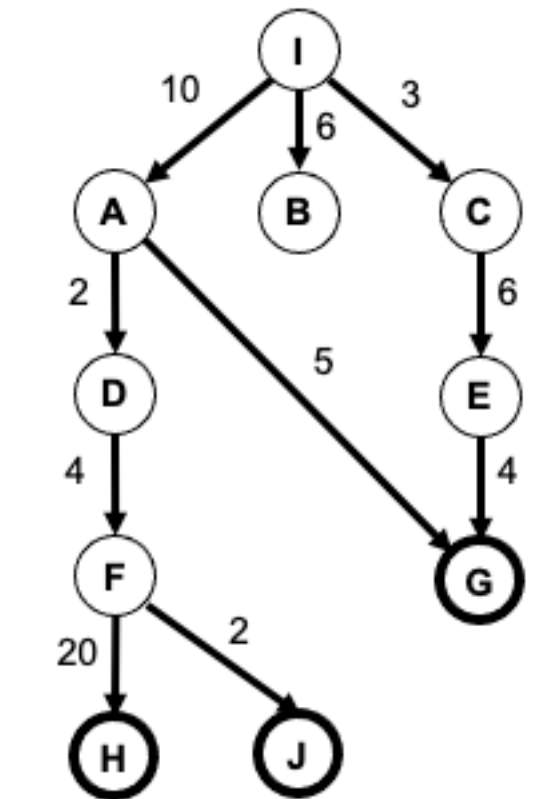
- Ignore the danger of repeated states: wasteful (BFS) or impossible (DFS). Can you see why?
- How to prevent it?

## If state space graph is not a tree

- We have to remember already-expanded states (**CLOSED**).
- When we take out a state from the fringe (OPEN), check whether it is in CLOSED (already expanded).
  - If yes, throw it away.
  - If no, expand it (add successors to OPEN), and move it to CLOSED.



**Q3-1: Consider the state space graph below. Goal states have bold borders. Nodes are expanded left to right when there are ties. What solution path is returned by BFS?**



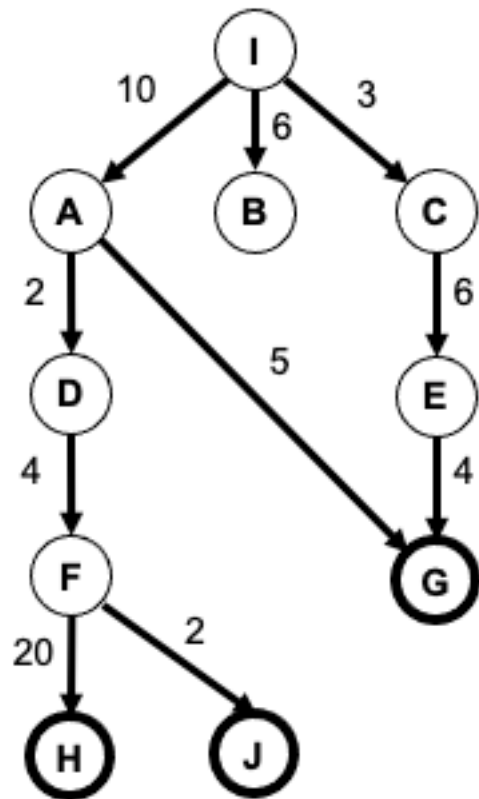
1. IADFH

2. IADFJ

3. IAG

4. ICEG

**Q3-1: Consider the state space graph below. Goal states have bold borders. Nodes are expanded left to right when there are ties. What solution path is returned by BFS?**



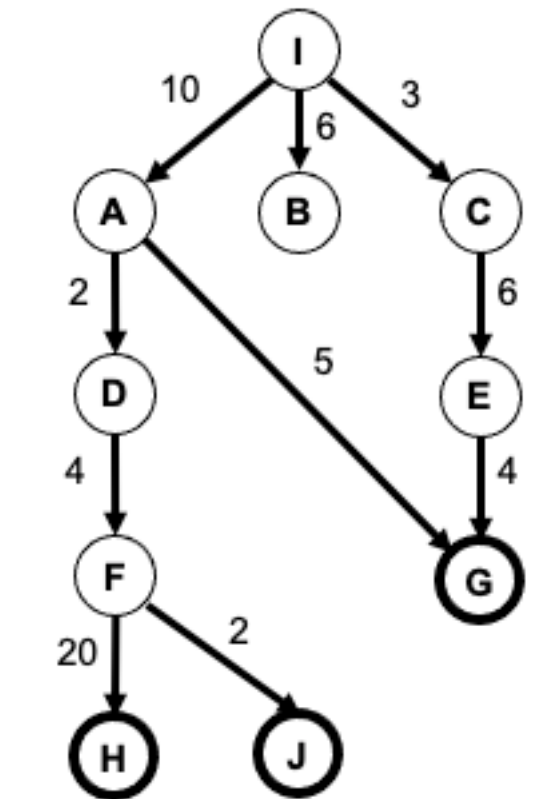
1. IADFH

2. IADFJ

3. IAG

4. ICEG

**Q3-2: Consider the state space graph below. Goal states have bold borders. Nodes are expanded left to right when there are ties. What solution path is returned by UCS?**



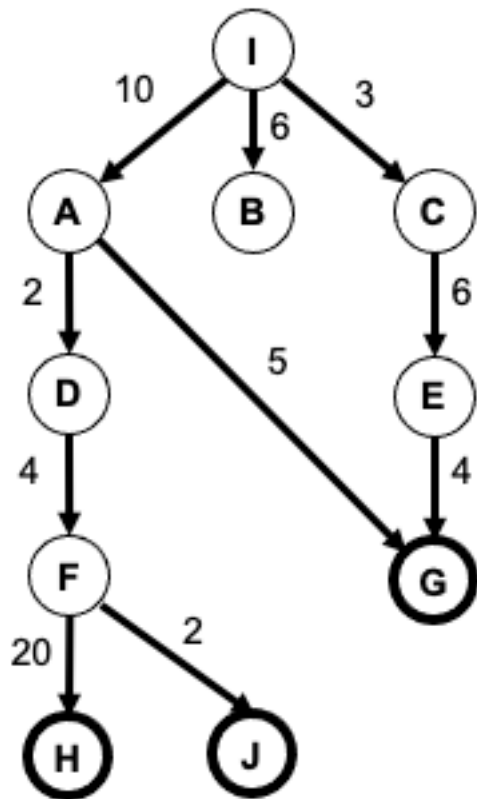
1. IADFH

2. IADFJ

3. IAG

4. ICEG

**Q3-2: Consider the state space graph below. Goal states have bold borders. Nodes are expanded left to right when there are ties. What solution path is returned by UCS?**



1. IADFH

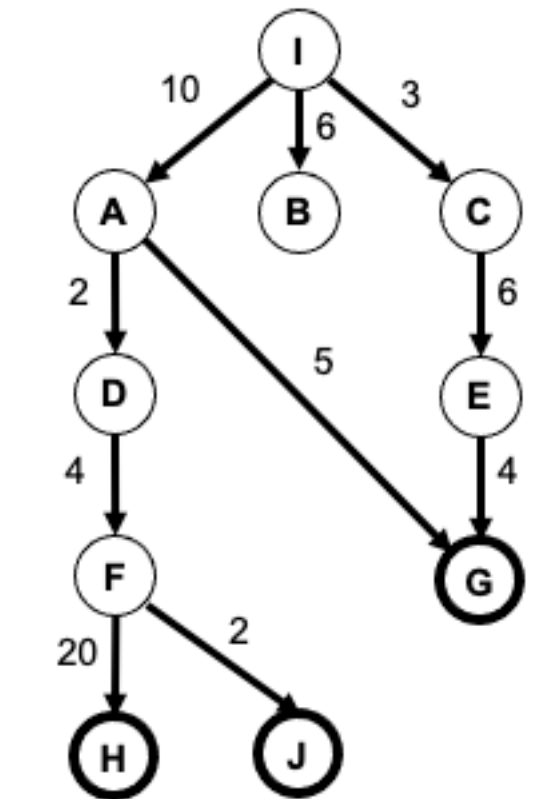
2. IADFJ

3. IAG



4. ICEG

**Q3-3: Consider the state space graph below. Goal states have bold borders. Nodes are expanded left to right when there are ties. What solution path is returned by DFS?**



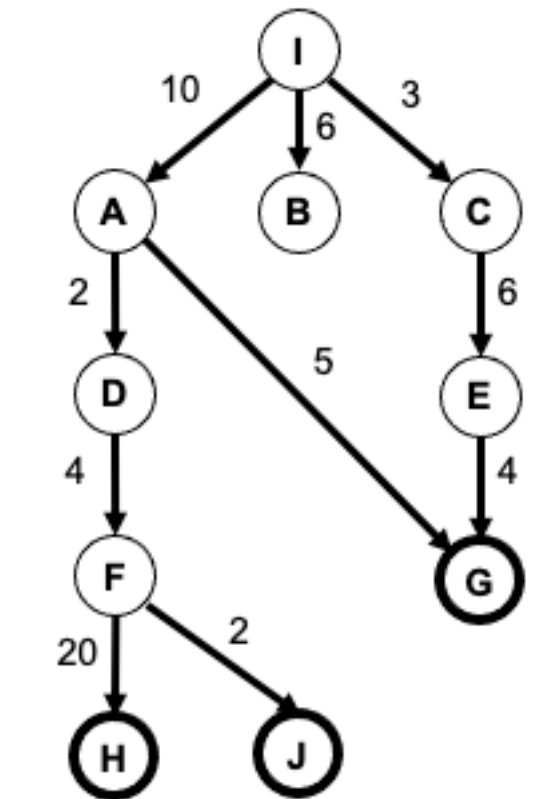
1. IADFH

2. IADFJ

3. IAG

4. ICEG

**Q3-3: Consider the state space graph below. Goal states have bold borders. Nodes are expanded left to right when there are ties. What solution path is returned by DFS?**



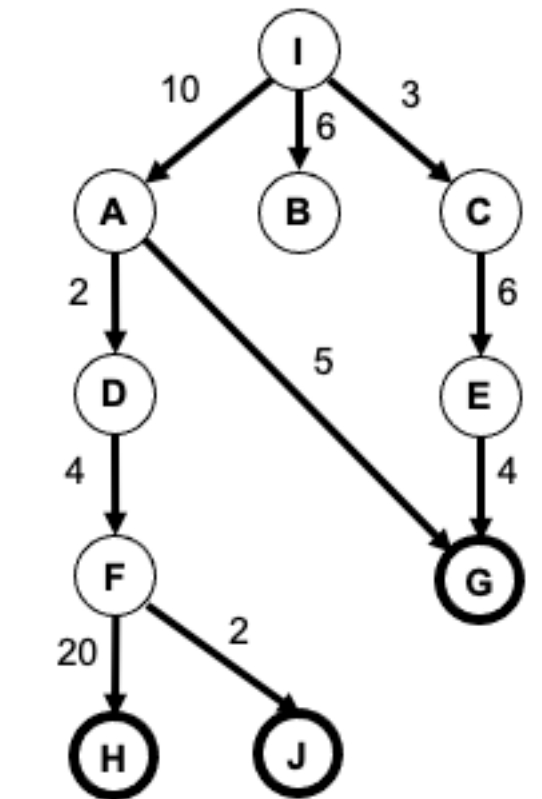
1. IADFH

2. IADFJ

3. IAG

4. ICEG

**Q3-4: Consider the state space graph below. Goal states have bold borders. Nodes are expanded left to right when there are ties. What solution path is returned by IDS?**



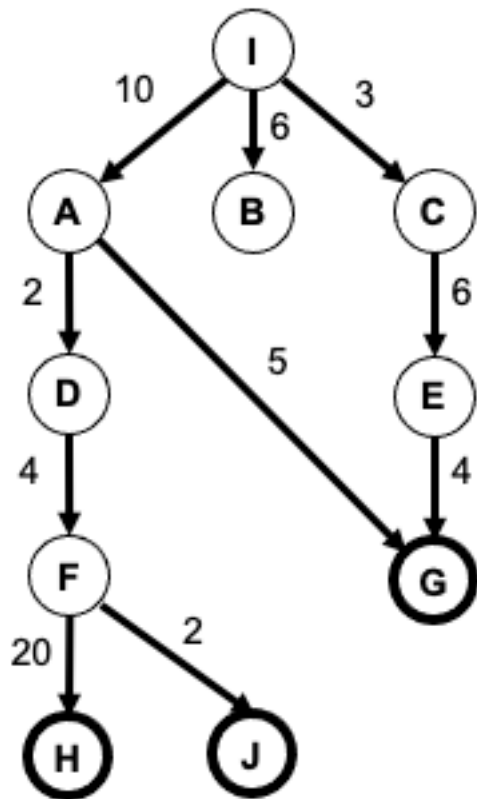
1. IADFH

2. IADFJ

3. IAG

4. ICEG

**Q3-4: Consider the state space graph below. Goal states have bold borders. Nodes are expanded left to right when there are ties. What solution path is returned by IDS?**



1. IADFH

2. IADFJ

3. IAG

4. ICEG



# What you should know

- Problem solving as search: state, successors, goal test
- Uninformed search
  - Breadth-first search
    - Uniform-cost search
  - Depth-first search
  - Iterative deepening



- Can you unify them using the same algorithm, with different priority functions?
- Performance measures
  - Completeness, optimality, time complexity, space complexity