CS540 Intro to AI
Uninformed Search

Josiah Hanna
University of Wisconsin-Madison

Slides created by Xiaojin Zhu (UW-Madison)
lightly edited by Anthony Gitter, Josiah Hanna

Announcements

- Homework 7 due Tuesday.
- Midterm grade revision requests due today.

Search

Search

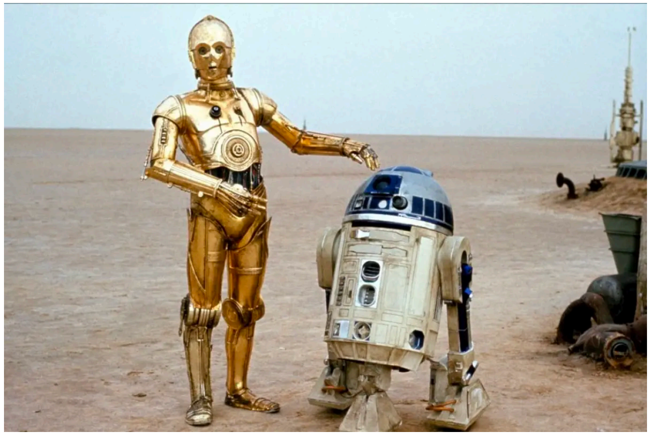
- So far we have discussed fundamentals, logic, natural language processing, unsupervised learning, and supervised learning.

Search

- So far we have discussed fundamentals, logic, natural language processing, unsupervised learning, and supervised learning.
- How should an intelligent agent take action?

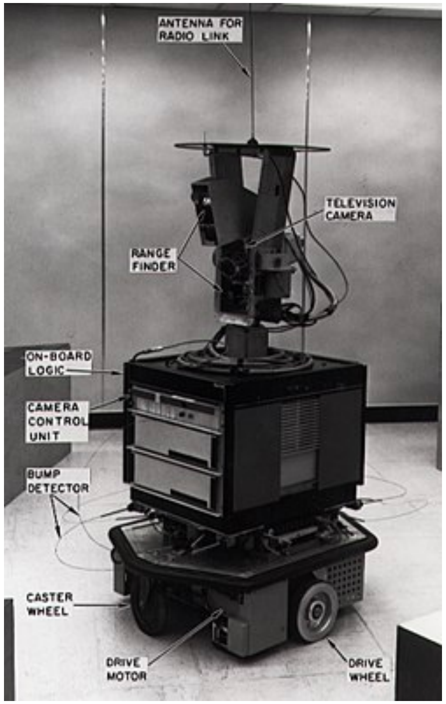
Search

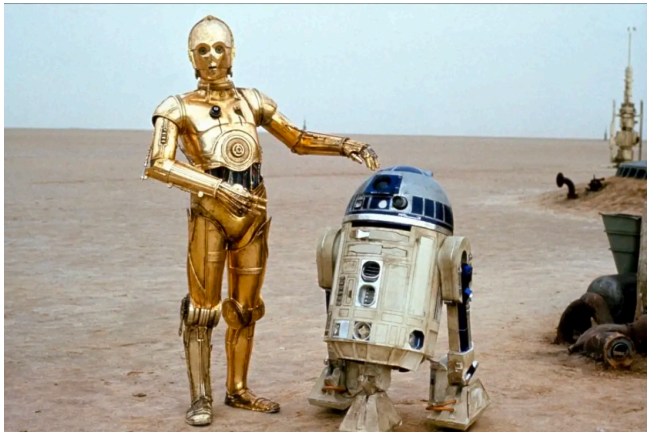
- So far we have discussed fundamentals, logic, natural language processing, unsupervised learning, and supervised learning.
- How should an intelligent agent take action?



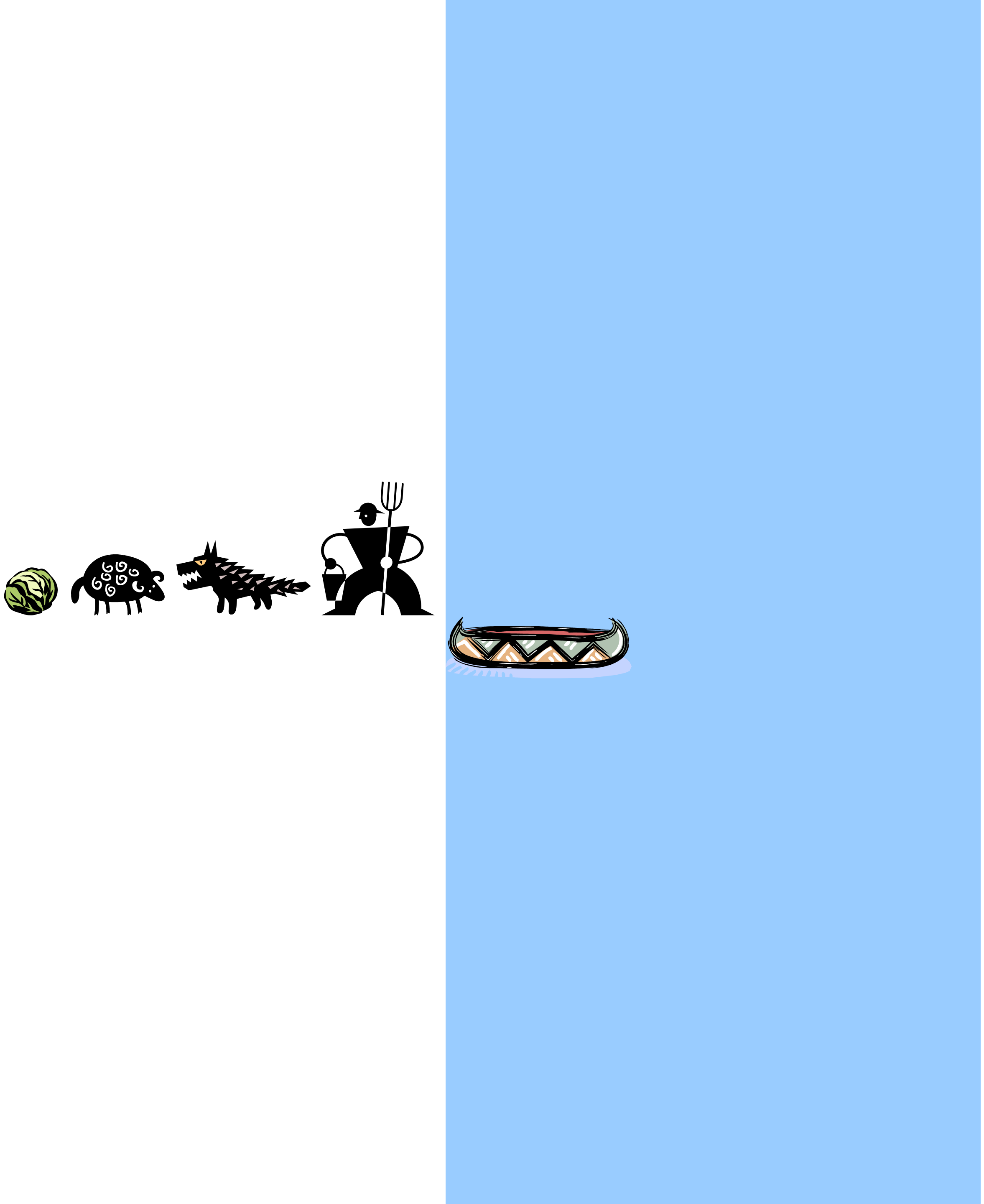
Search

- So far we have discussed fundamentals, logic, natural language processing, unsupervised learning, and supervised learning.
- How should an intelligent agent take action?





Many AI problems can be formulated as search.

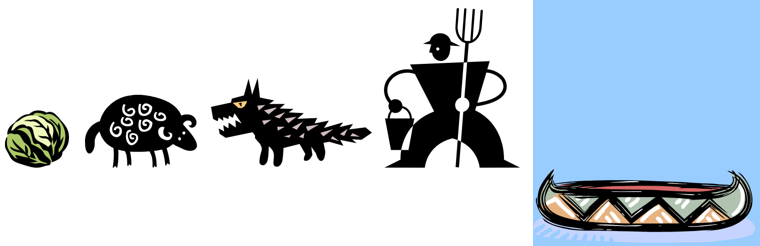


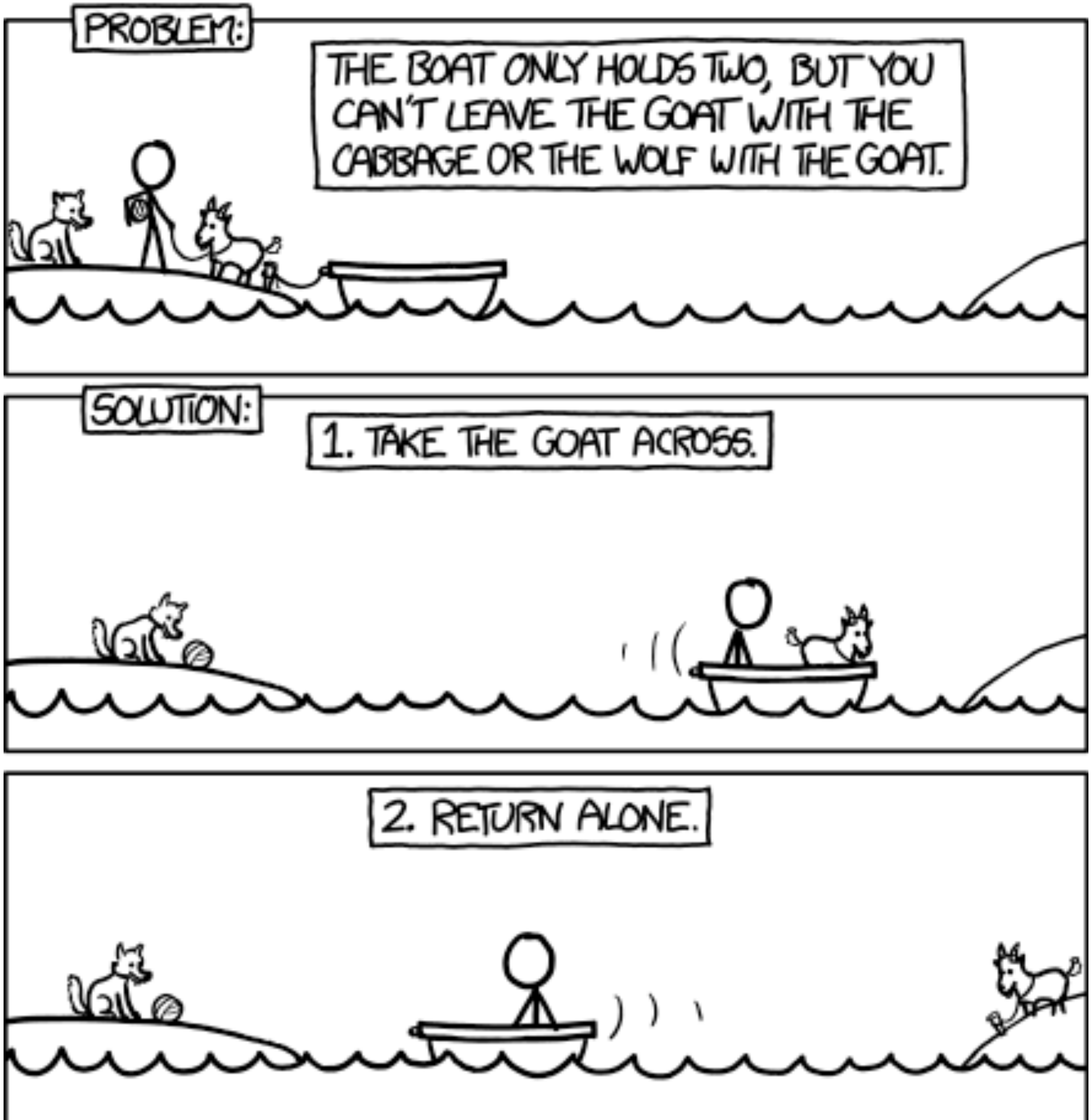
Boat can only hold two at a time.

Sheep and cabbage cannot be alone together.

Wolf and Sheep cannot be alone together.

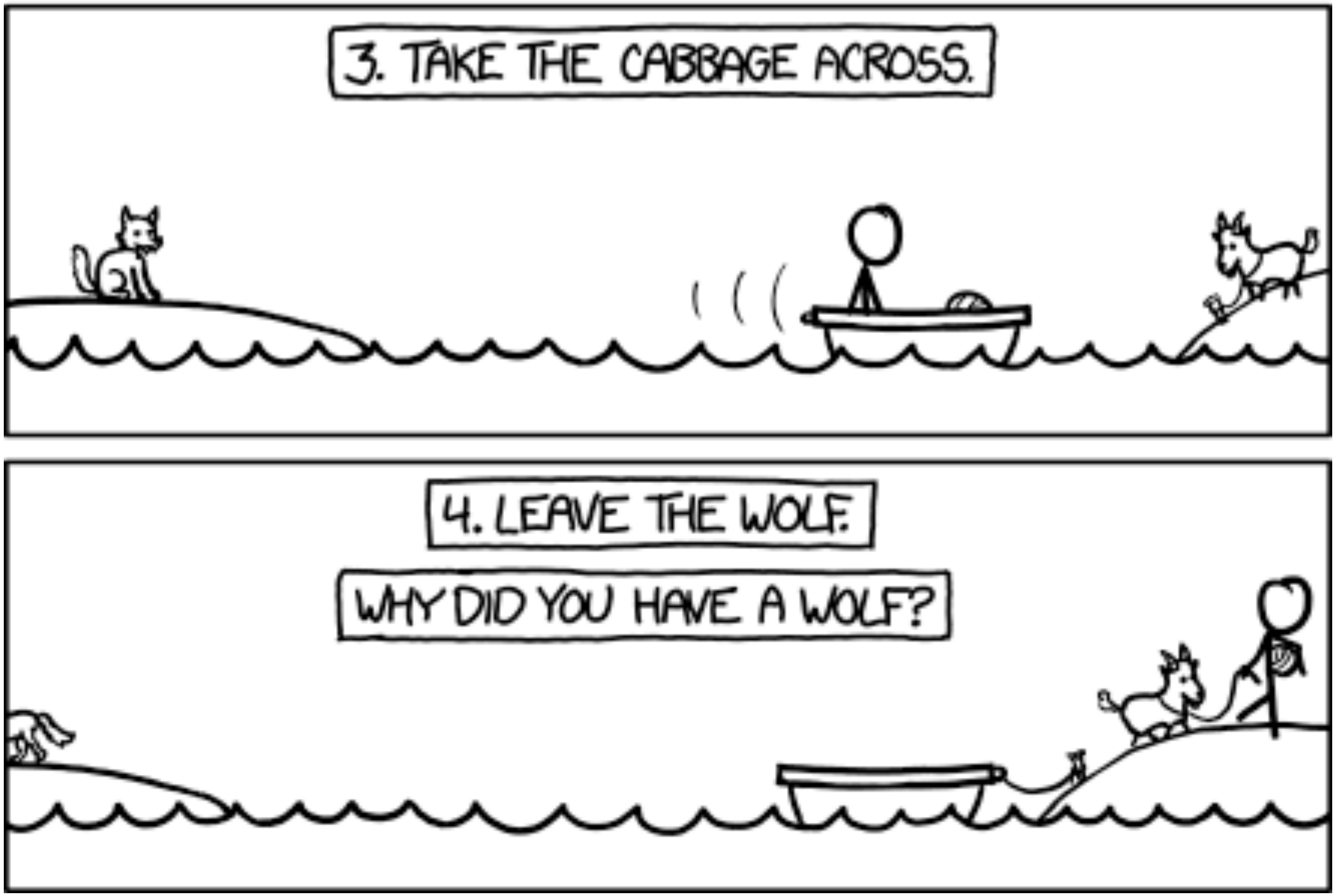
Farmer must go in the boat.



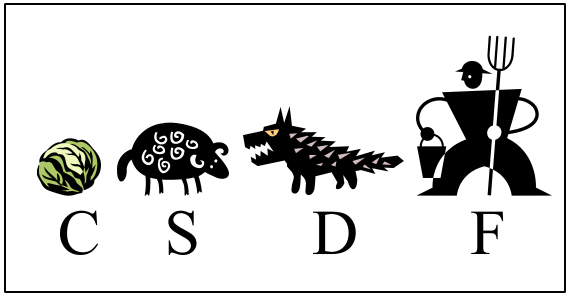


http://xkcd.com/1134/

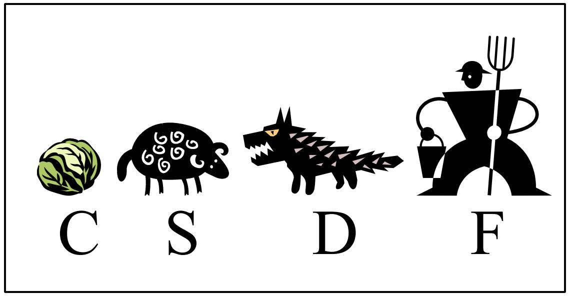
http://xkcd.com/1134/



The search problem

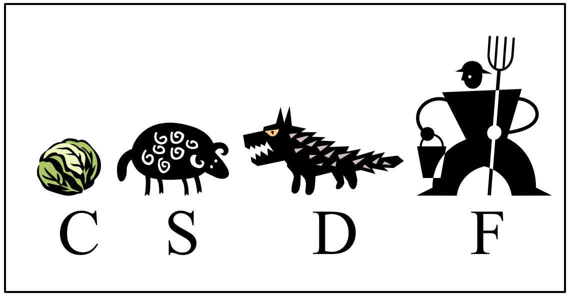


The search problem



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

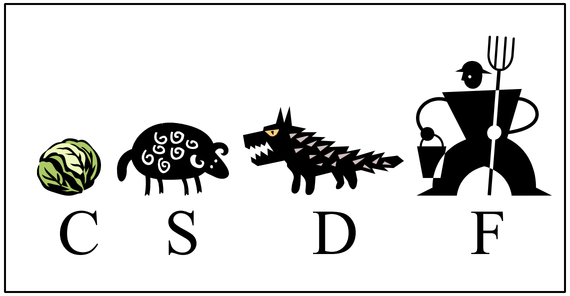
The search problem



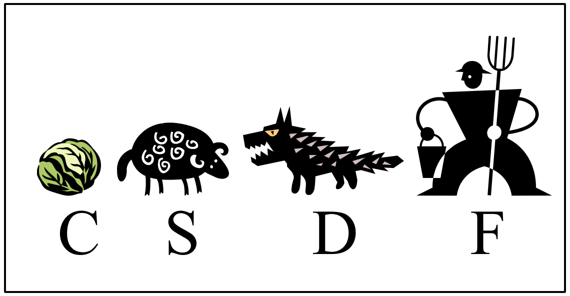
- State space ***S*** : all valid configurations
- Initial state ***I***={(CSDF,)} ⊆ ***S***

The search problem

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

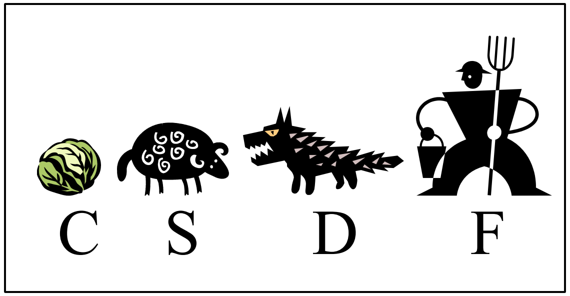


The search problem



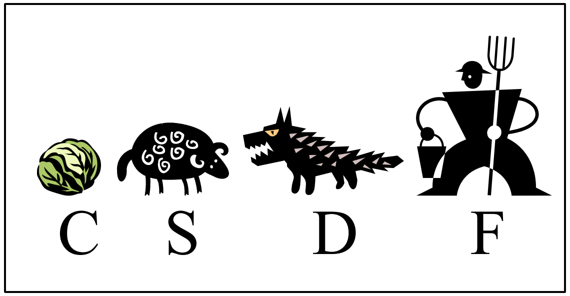
- State space ***S*** : all valid configurations
- Initial state ***I***={(CSDF,)} ⊆ ***S***
- Goal state ***G***={(,CSDF)} ⊆ ***S***
- Successor function ***succs***(s)⊆ ***S*** : states reachable in one step from state s
  - ***succs***((CSDF,)) = {(CD, SF)}
  - ***succs***((CDF,S)) = {(CD,FS), (D,CFS),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

The search problem



- State space ***S*** : all valid configurations
- Initial state ***I***={(CSDF,)} ⊆ ***S***
- Goal state ***G***={(,CSDF)} ⊆ ***S***
- Successor function ***succs***(s)⊆ ***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).

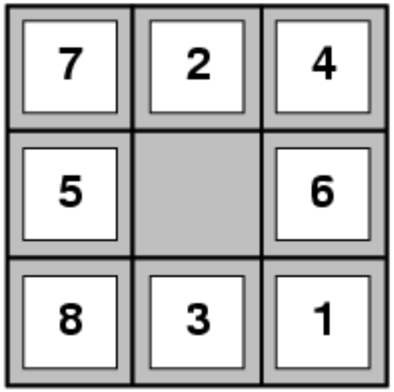
The search problem



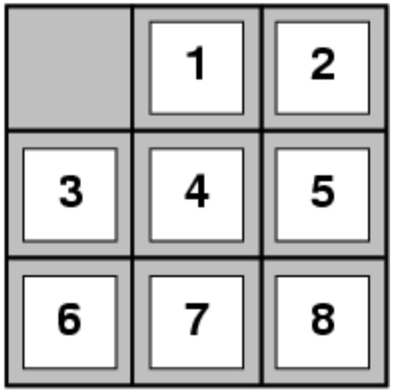
- State space ***S*** : all valid configurations
- Initial state ***I***={(CSDF,)} ⊆ ***S***
- Goal state ***G***={(,CSDF)} ⊆ ***S***
- Successor function ***succs***(s)⊆ ***S*** : states reachable in one step from state s
  - ***succs***((CSDF,)) = {(CD, SF)}
  - ***succs***((CDF,S)) = {(CD,FS), (D,CFS),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

Search examples

- 8-puzzle



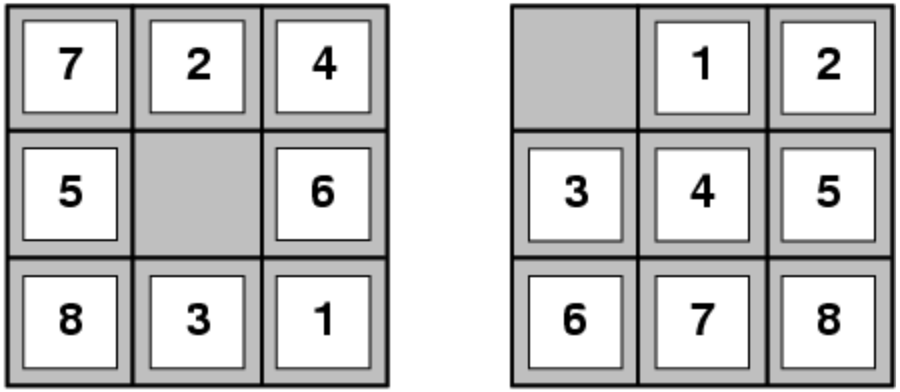
Start State



Goal State

Search examples

- 8-puzzle

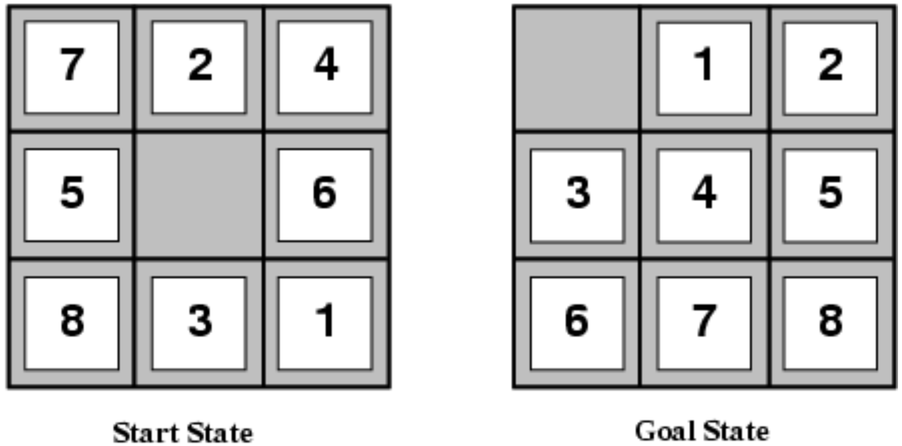


Start State Goal State

- States = 3x3 array configurations

Search examples

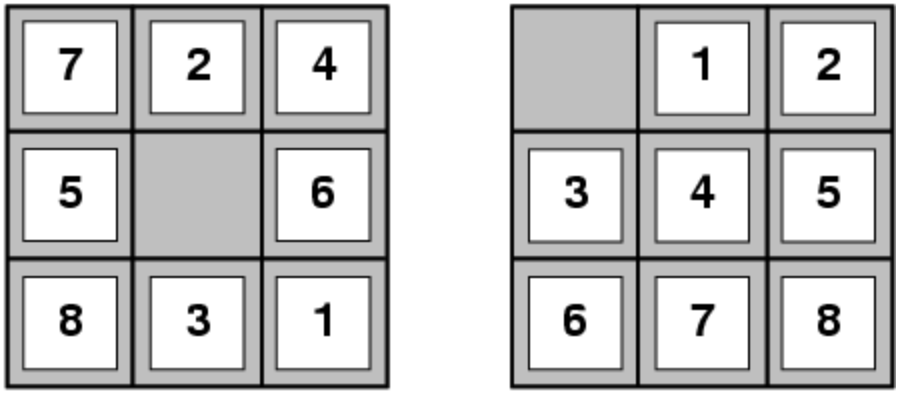
- 8-puzzle



- States = 3x3 array configurations
- Action = Move one of adjacent numbers to empty cell

Search examples

- 8-puzzle

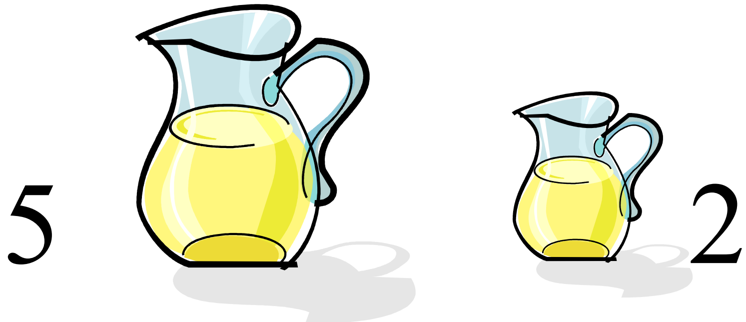


Start State Goal State

- States = 3x3 array configurations
- Action = Move one of adjacent numbers to empty cell
- Cost = 1 for each move

Search examples

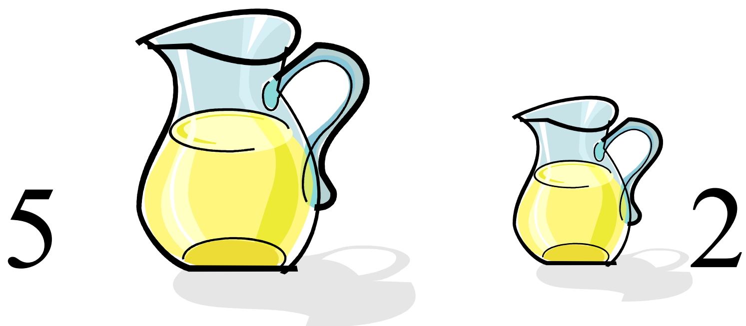
- Water jugs: how to get 1?



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

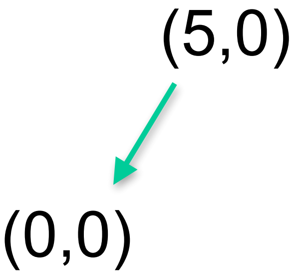
Operators / Actions

(x,y) -> (0,y) ; empty 5-gal jug
(x,y) -> (x,0) ; empty 2-gal jug
(x,2) and x<=3 -> (x+2,0) ; pour 2-gal into 5-gal
(x,0) and x>=2 -> (x-2,2) ; pour 5-gal into 2-gal
(1,0) -> (0,1) ; empty 5-gal into 2-gal

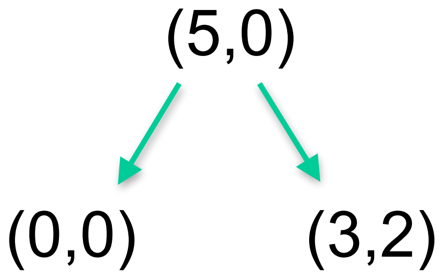
Search examples

(5,0)

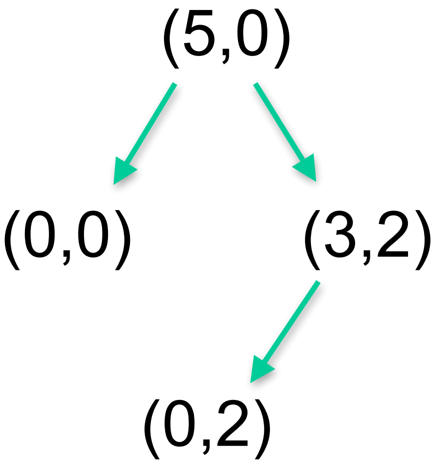
Search examples



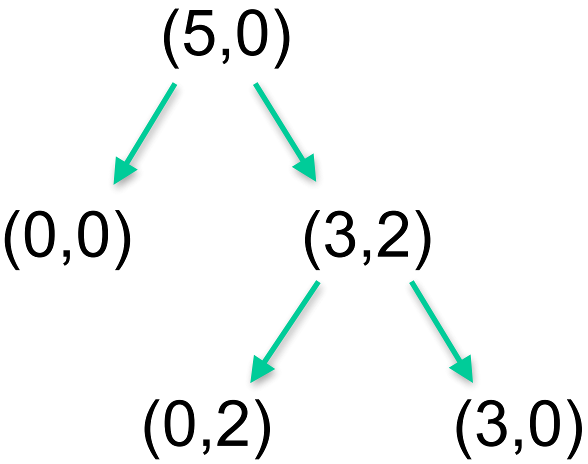
Search examples



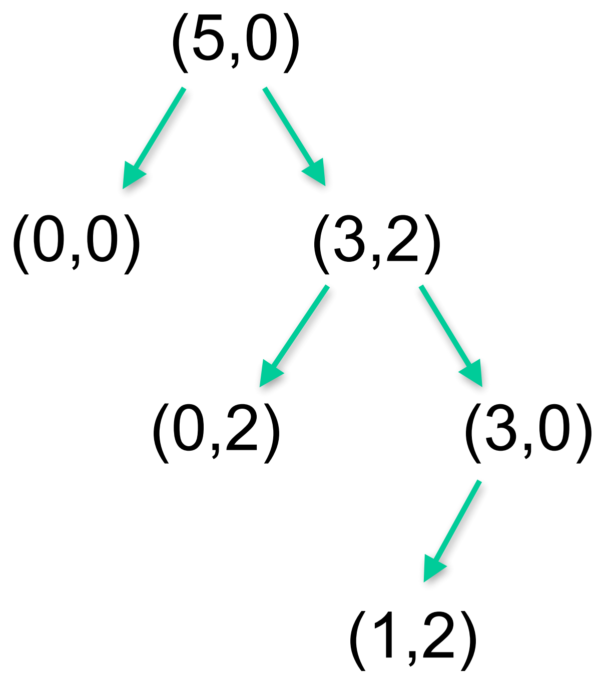
Search examples



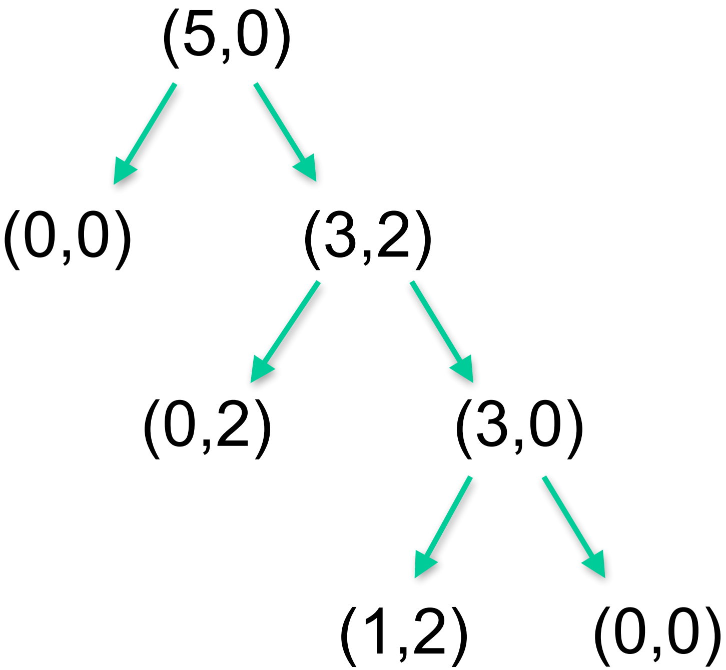
Search examples



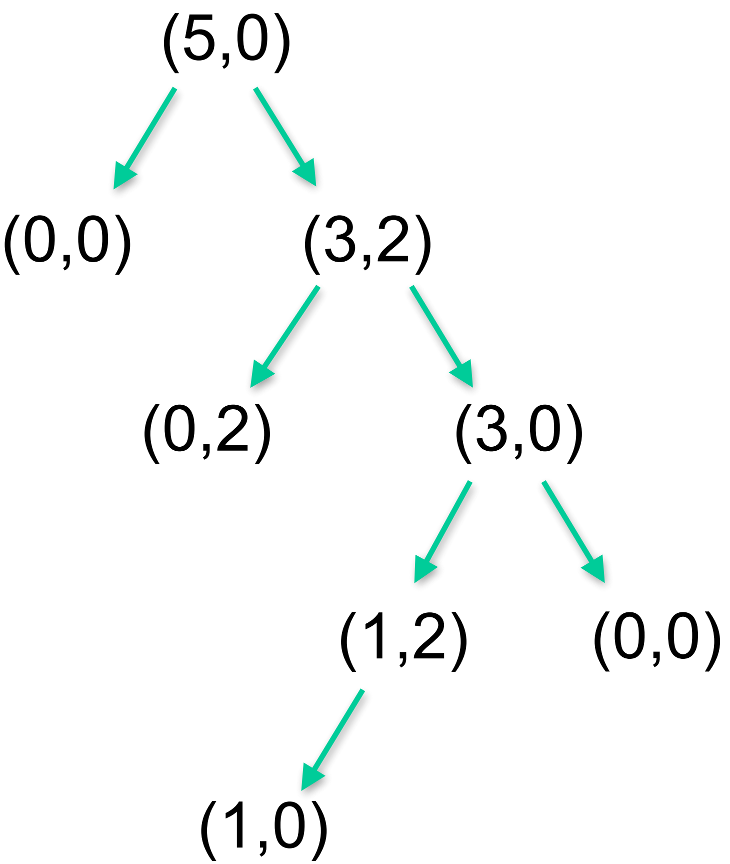
Search examples



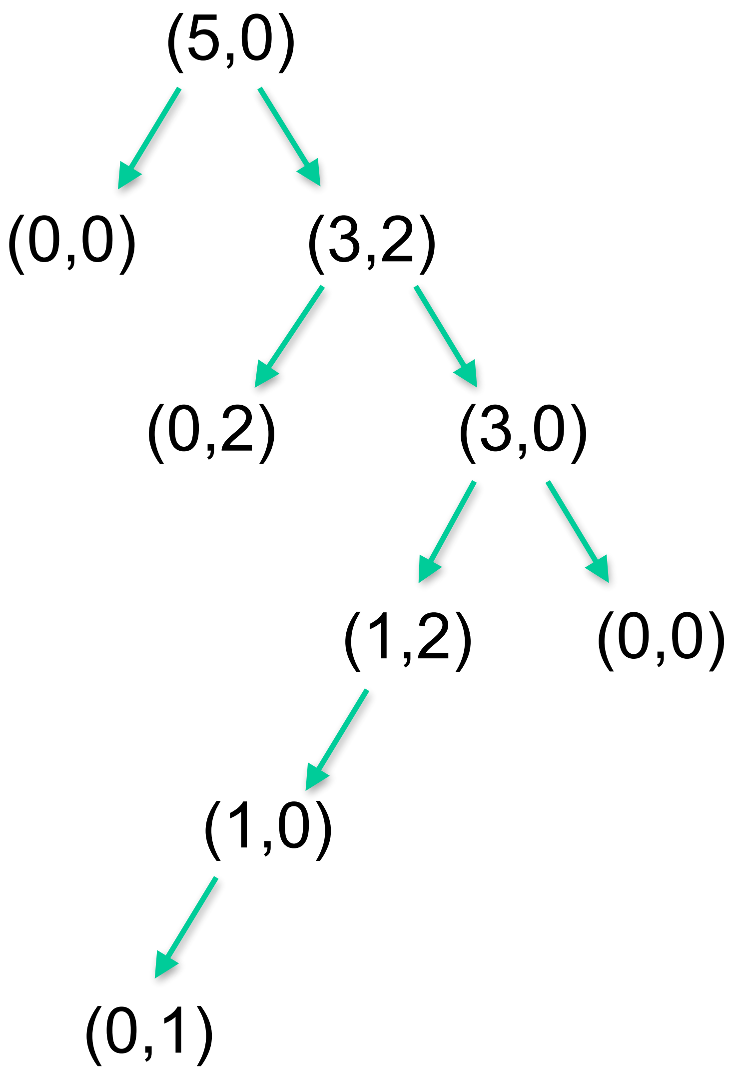
Search examples



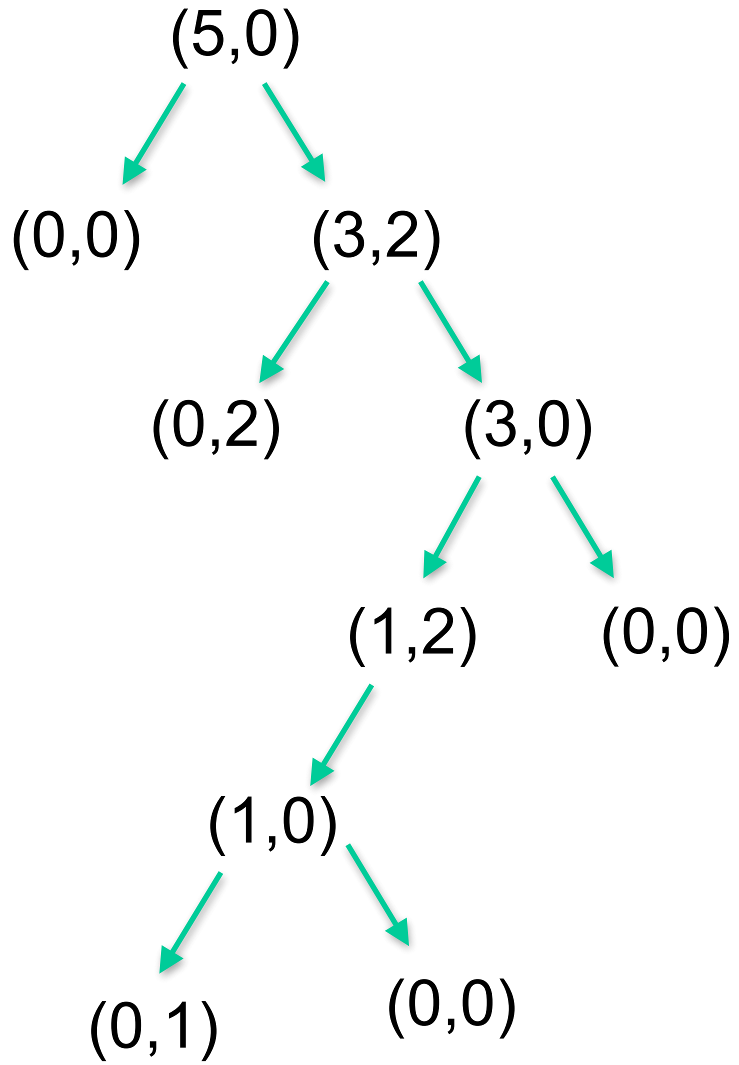
Search examples



Search examples

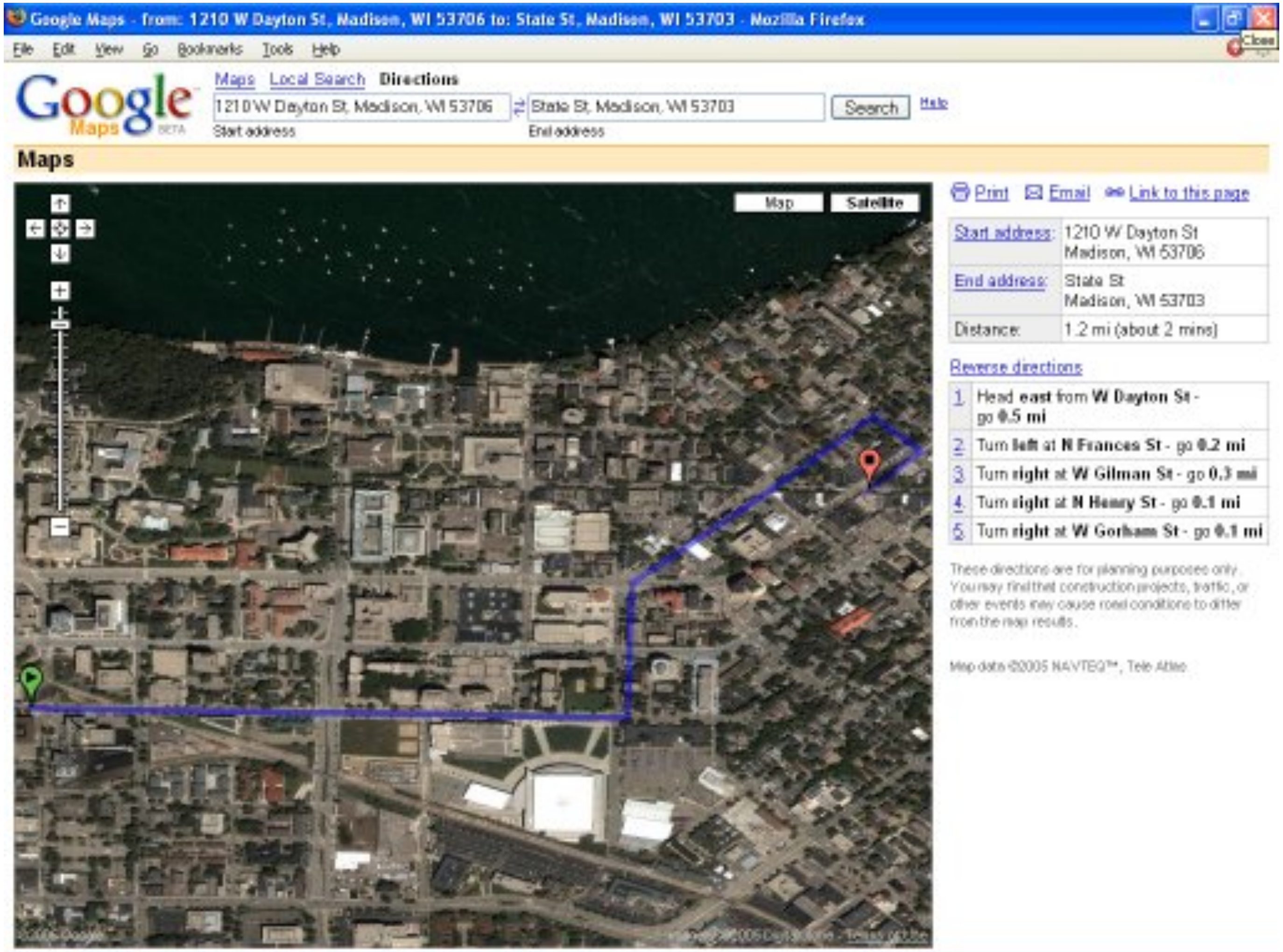


Search examples

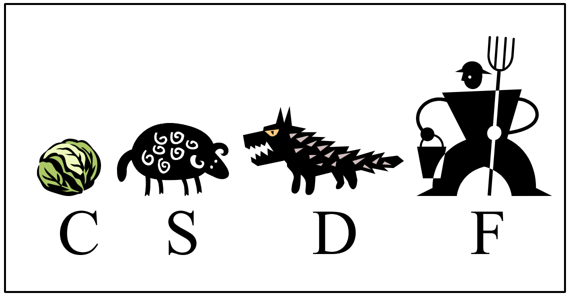


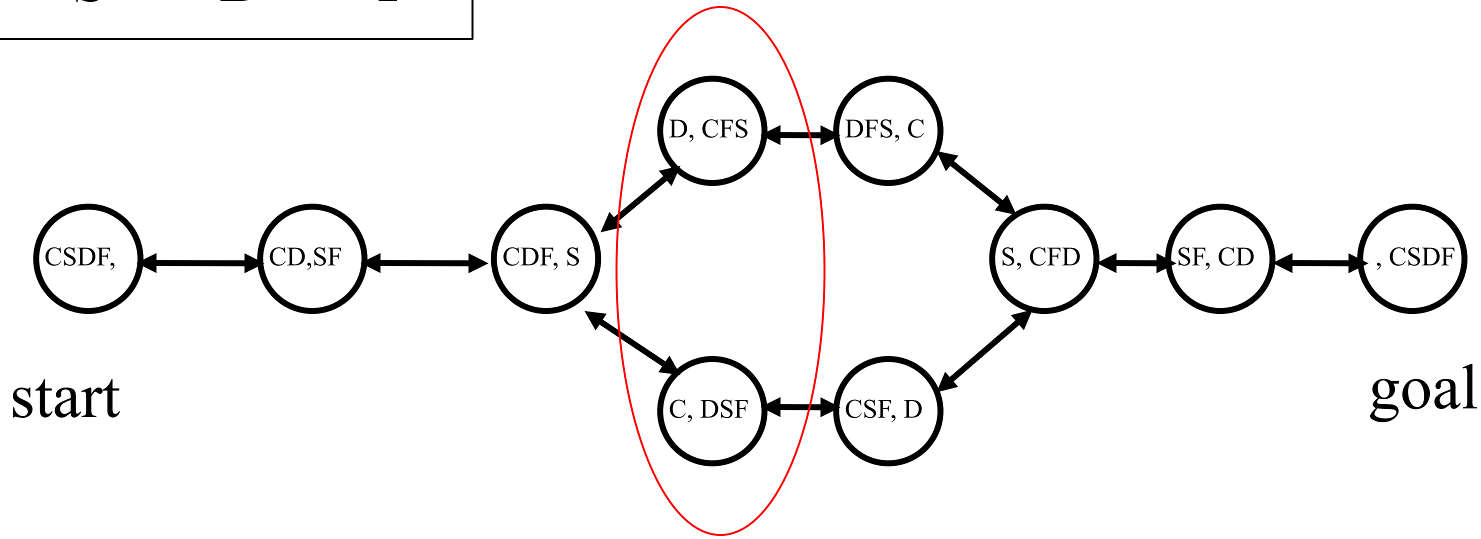
Search examples

- Route finding (State? Successors? Cost weighted)



A directed graph in state space

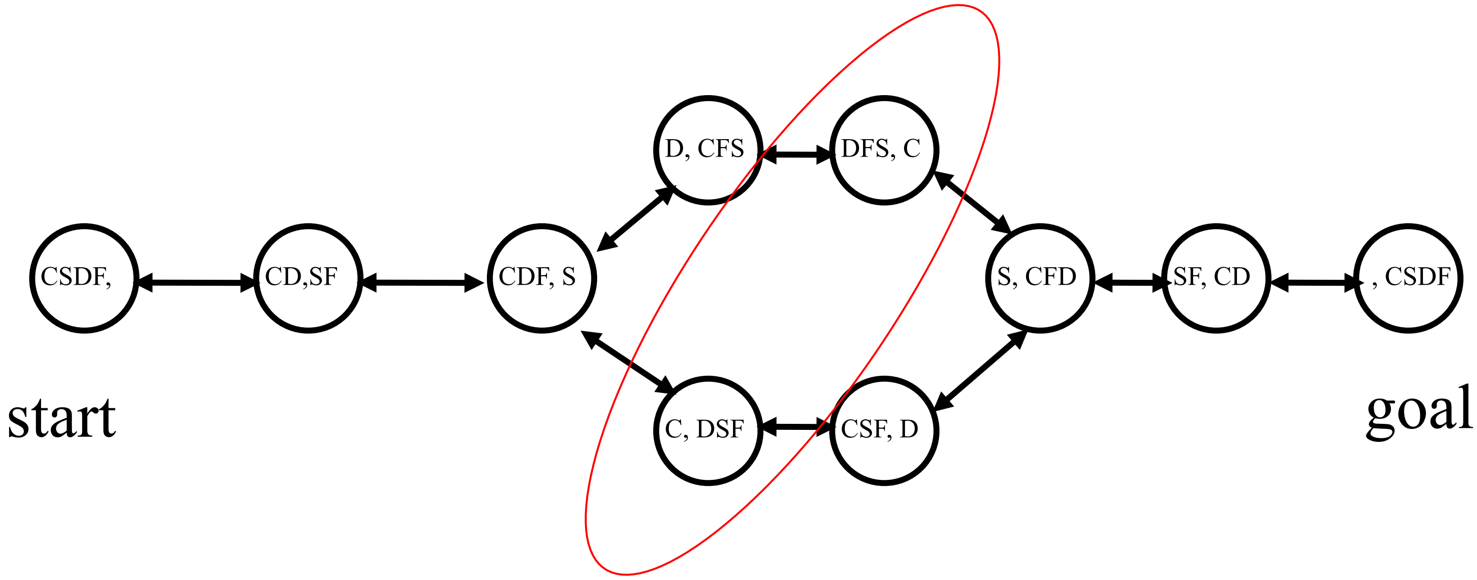




- In general there will be many generated, but un-expanded states at any given time
- One has to choose which one to expand next

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

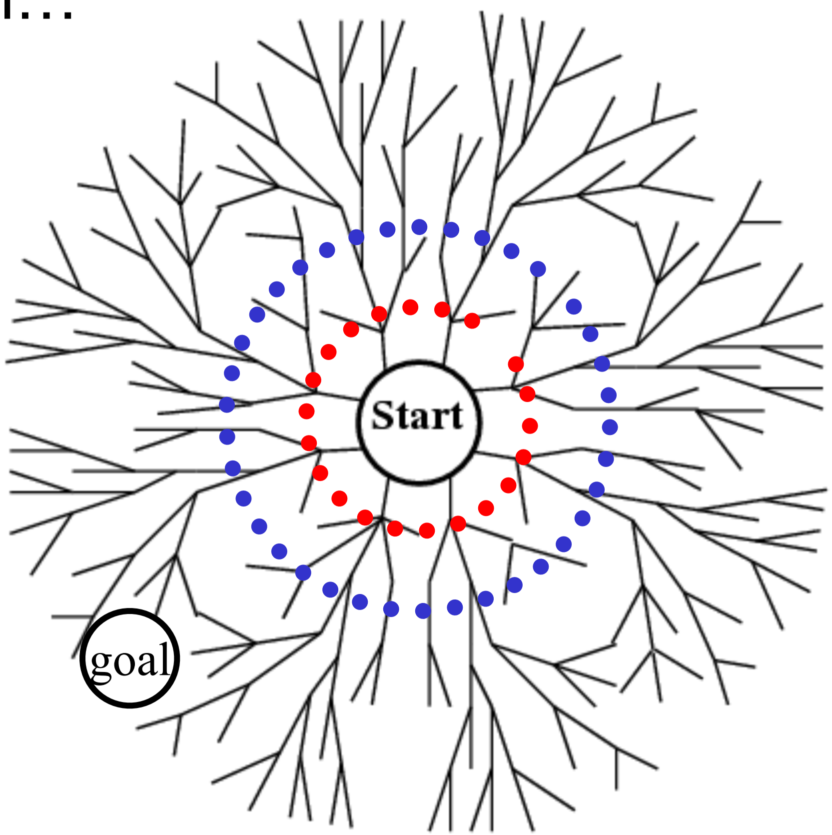
- Uninformed means we only know:
  - The goal test
  - The ***succs***() function
- But not which non-goal states are better: that would be informed search (next topic).
- For now, we also assume ***succs***() graph is a tree.
  - Won't encounter repeated states.
  - We will relax it later.
- Search strategies:
  - Breadth-first Search (BFS)
  - Uniform Cost Search (UCS)
  - Depth-first Search (DFS)
  - Iterative Deepening Search (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…

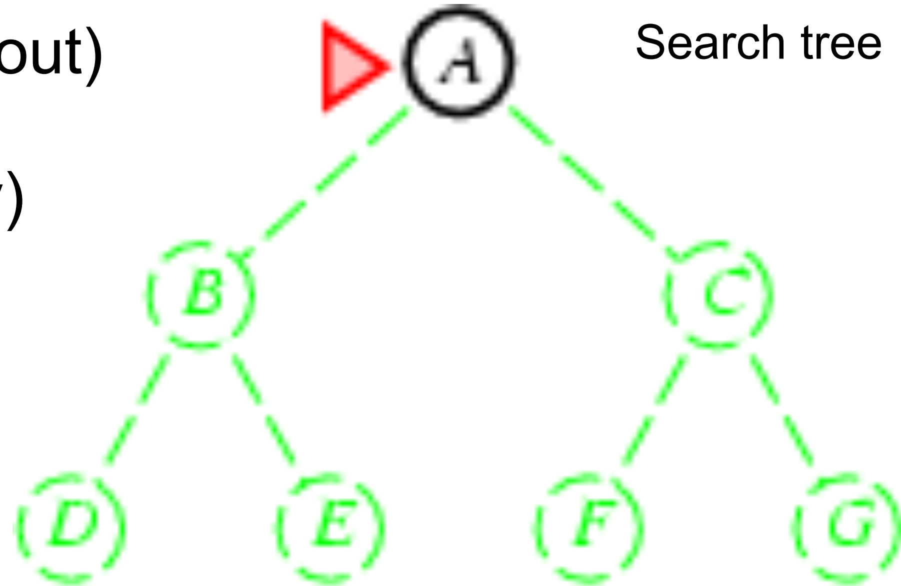
ripple



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



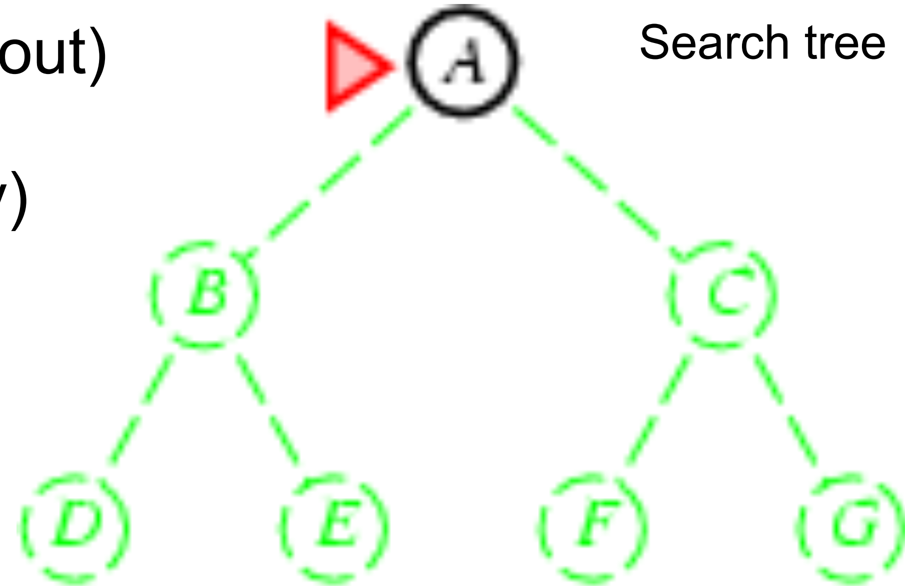
Initial state: **A**
Goal state: **G**

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

Search tree



queue (fringe, OPEN)
→ [A] →

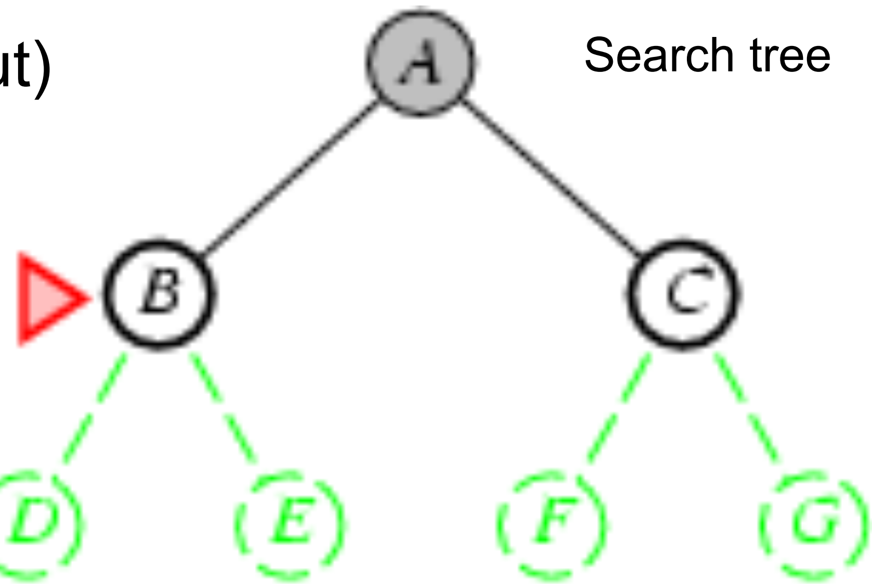
Initial state: **A**
Goal state: **G**

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

Search tree



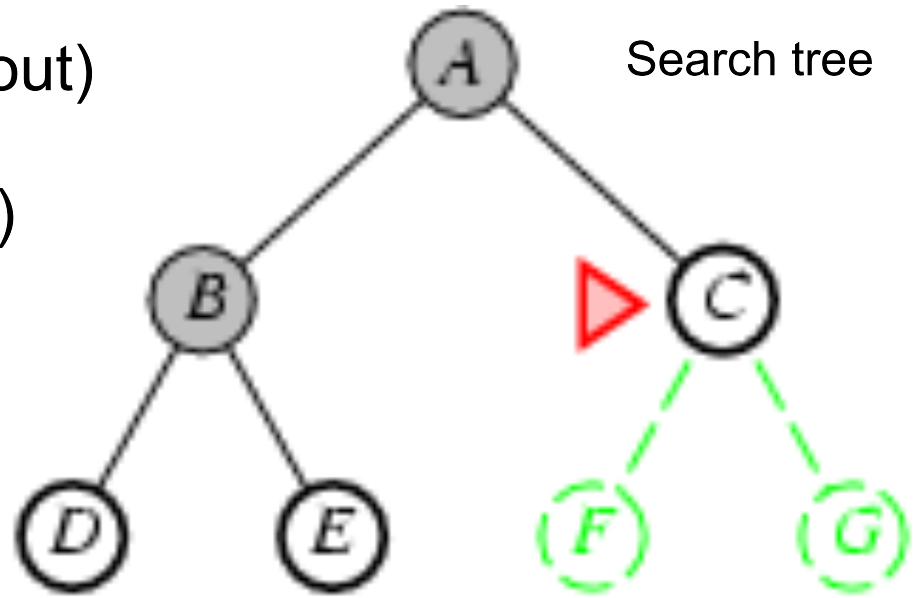
queue (fringe, OPEN)
→ [CB] → A

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



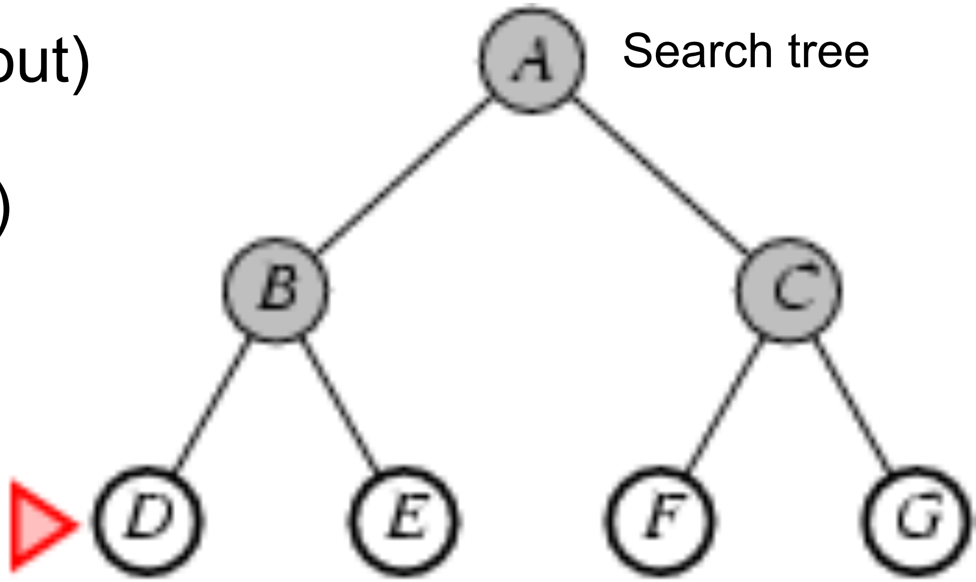
queue (fringe, OPEN)
→ [EDC] → B

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

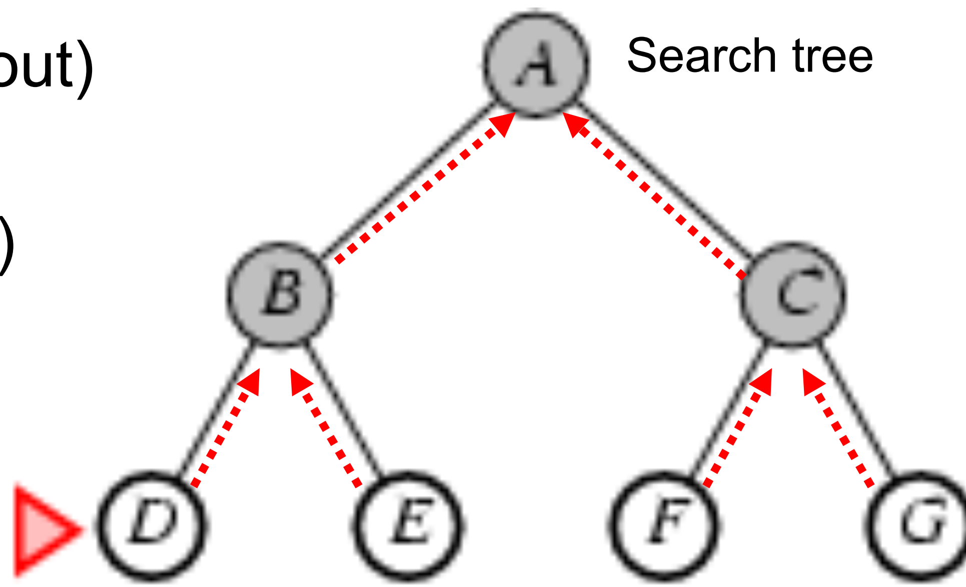
Initial state: **A**
Goal state: **G**

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

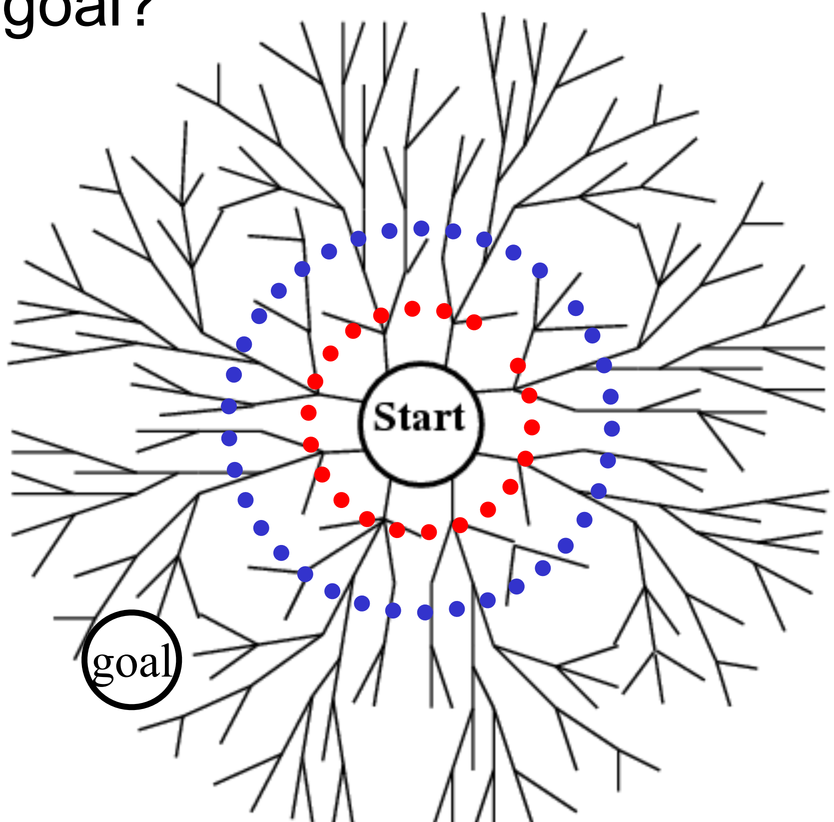
Looking foolish? Indeed. But let's be consistent…

... until much later we pop G.

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



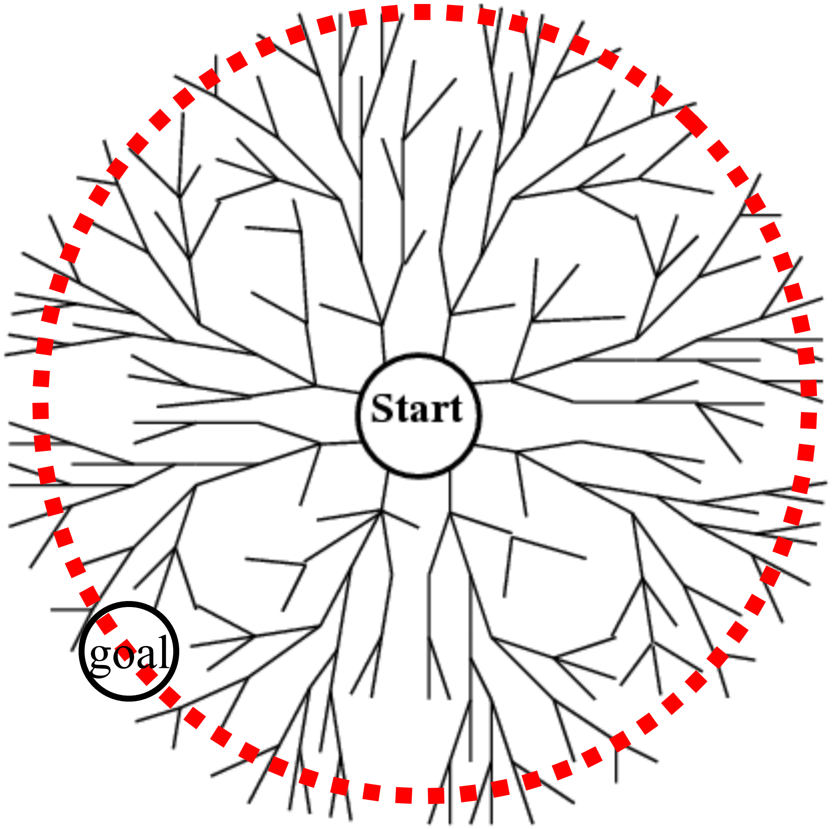
Performance of BFS

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***.
  - $b + b^2 + \ldots + b^d \sim O(b^d)$
- 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 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 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 with depth), no otherwise.
- Time complexity (worst case): goal is the last node at radius ***d***.
  - Have to generate all nodes at radius ***d***.
  - $b + b^2 + \ldots + b^d \sim O(b^d)$
- Space complexity (bad)
  - Back points for all generated nodes $O(b^d)$
  - The queue (smaller, but still $O(b^d)$)

Performance of BFS

Four measures of search algorithms:

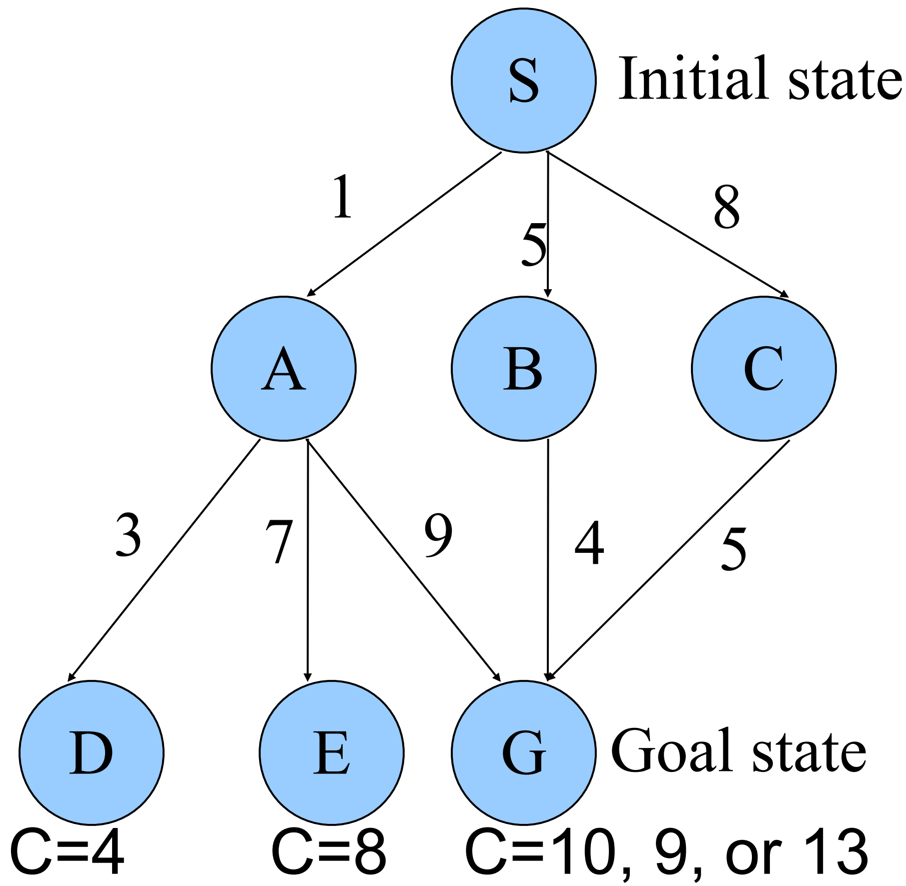
- Completeness (not finding all goals): yes, BFS will find a goal.
- Optimality: yes if edges cost 1 (more generally positive non-decreasing with depth), no otherwise.
- Time complexity (worst case): goal is the last node at radius ***d***.
  - Have to generate all nodes at radius ***d***.
  - $b + b^2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

Solution: Uniform-cost search

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

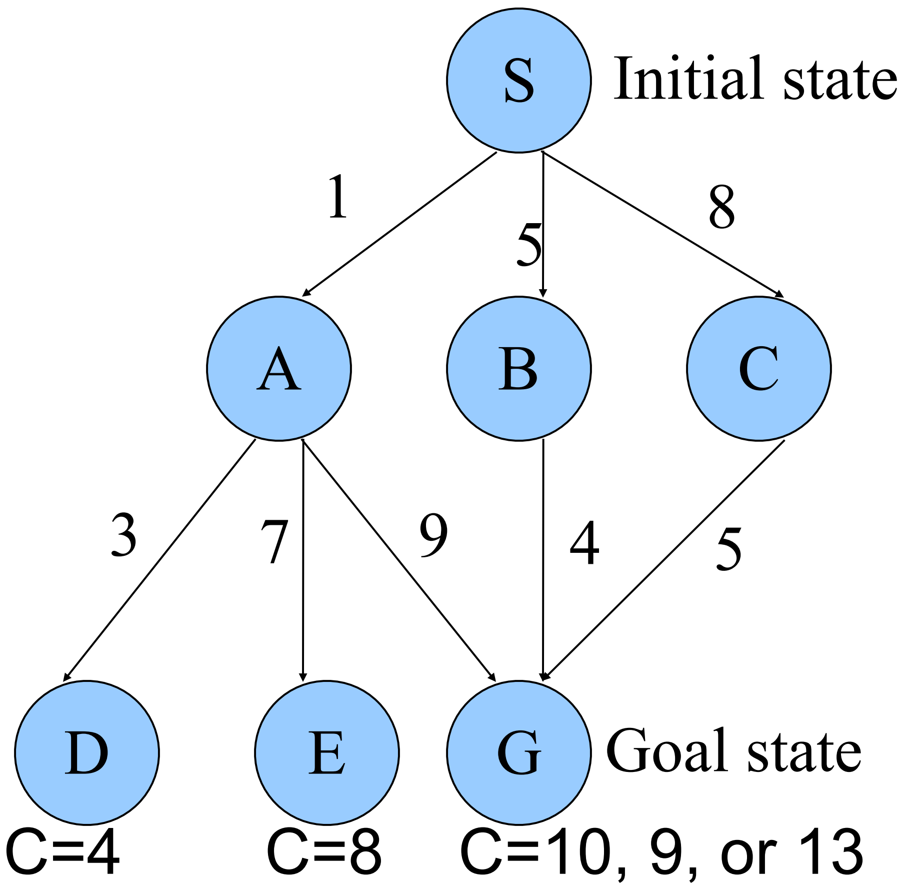
Example



(All edges are directed, pointing downwards)

Example

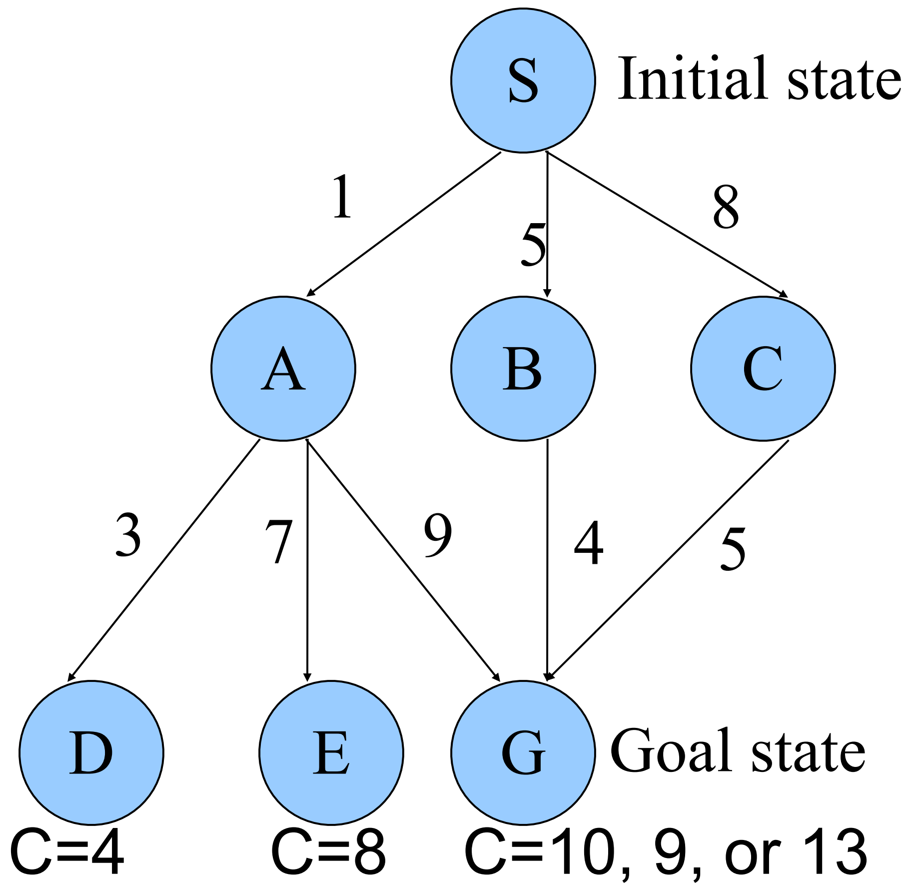
Expanded (Fringe):



(All edges are directed, pointing downwards)

Example

Expanded (Fringe):
1. S (A,B,C)

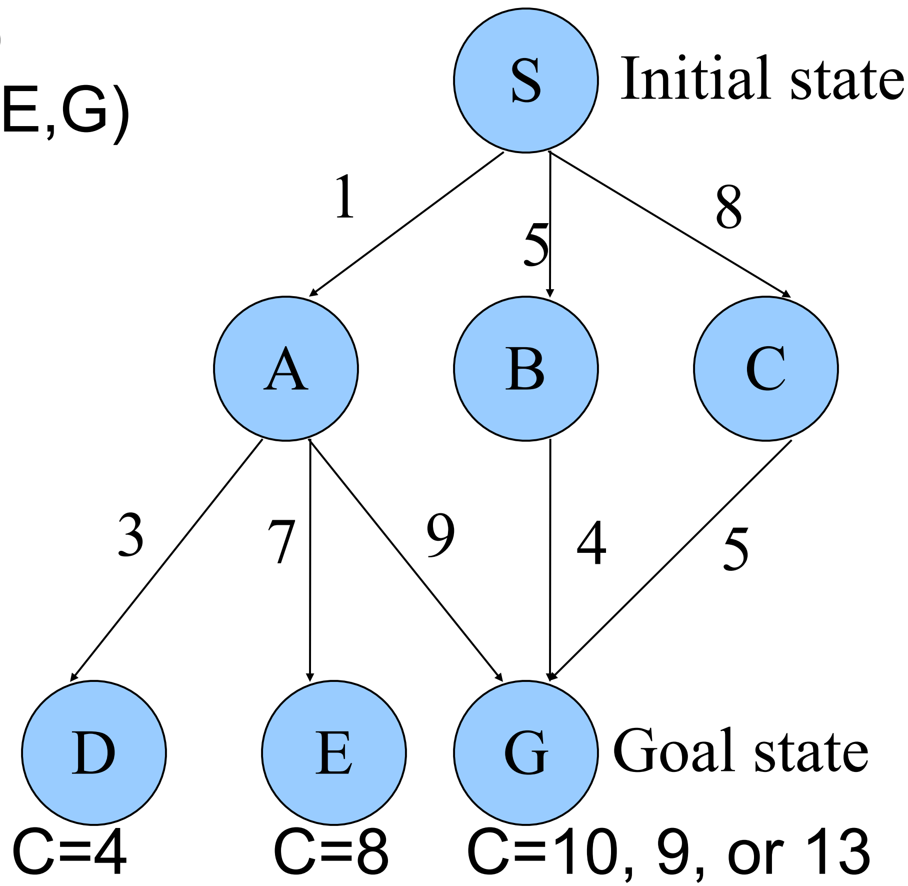


(All edges are directed, pointing downwards)

Example

Expanded (Fringe):

1. S (A,B,C)
2. A (D,B,C,E,G)

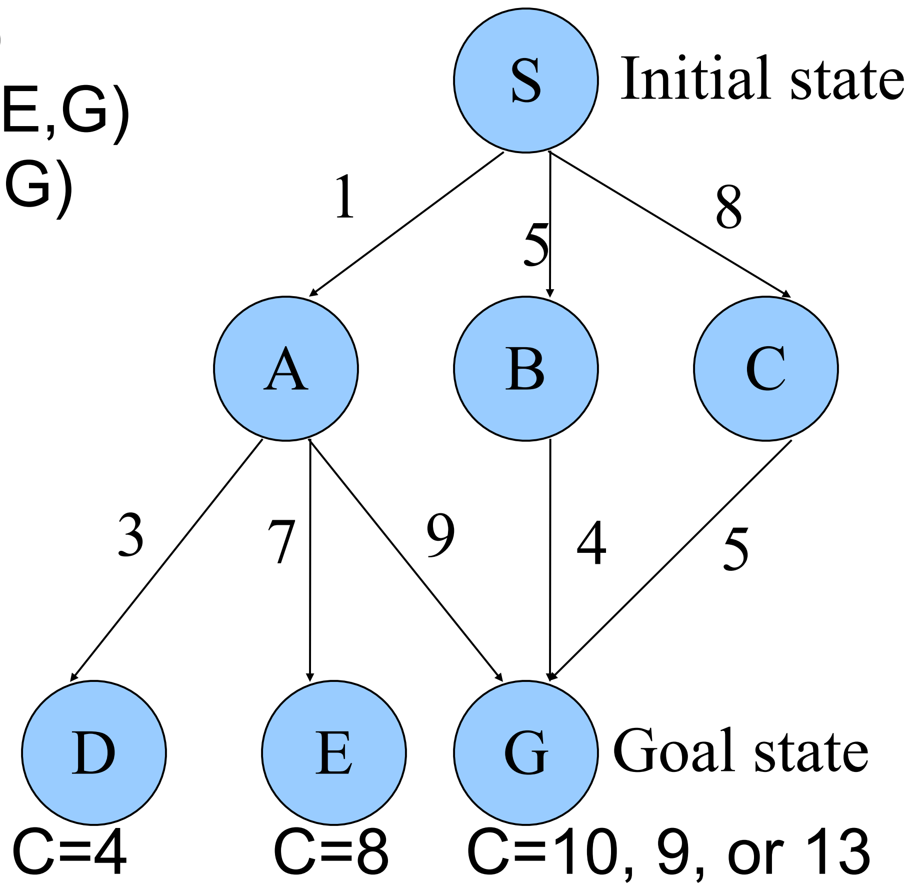


(All edges are directed, pointing downwards)

Example

Expanded (Fringe):

1. S (A,B,C)
2. A (D,B,C,E,G)
3. D (B,C,E,G)

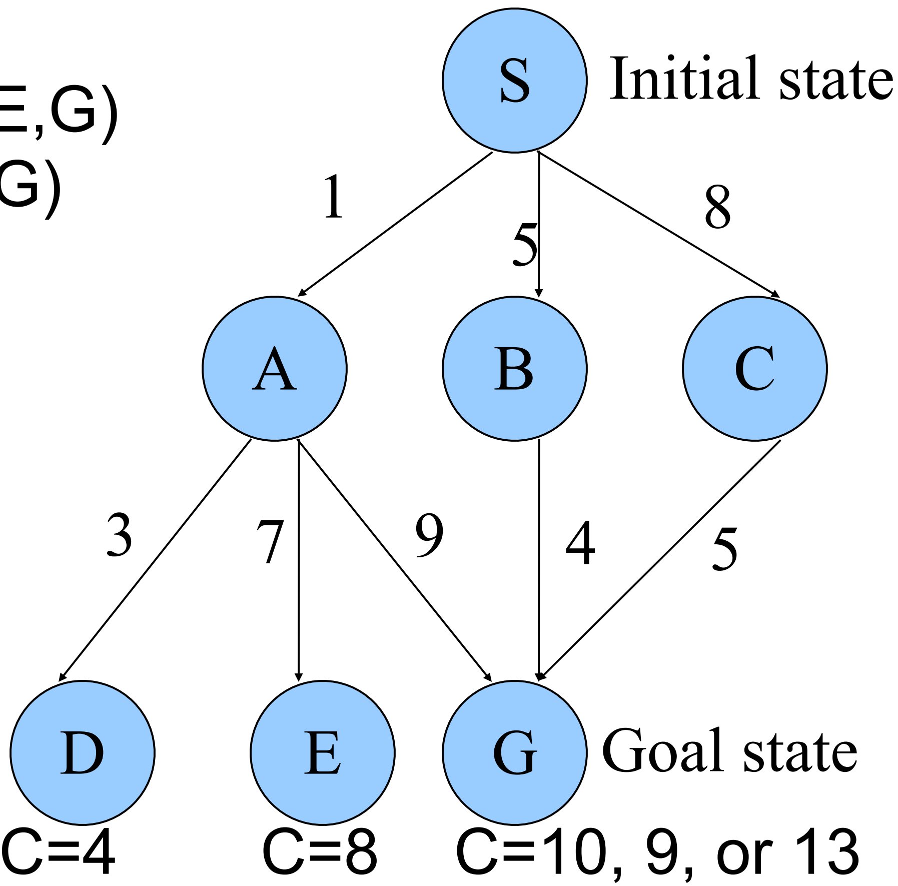


(All edges are directed, pointing downwards)

Example

Expanded (Fringe):

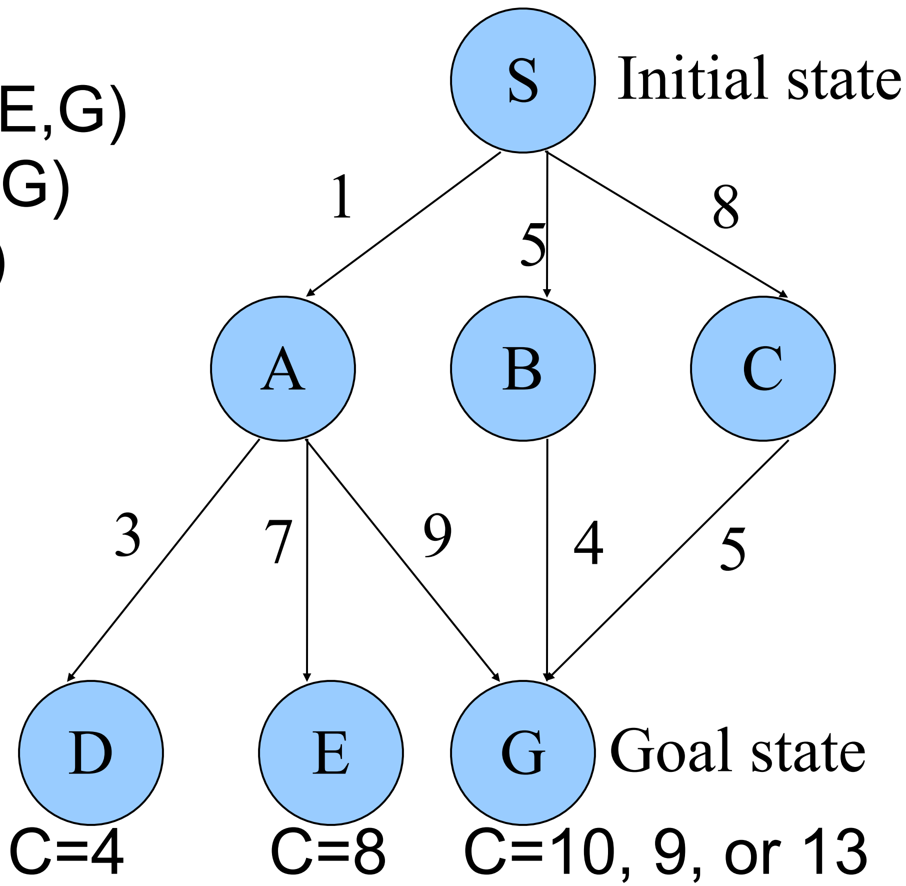
1. S (A,B,C)
2. A (D,B,C,E,G)
3. D (B,C,E,G)
4. B (C,E,G)



(All edges are directed, pointing downwards)

Example

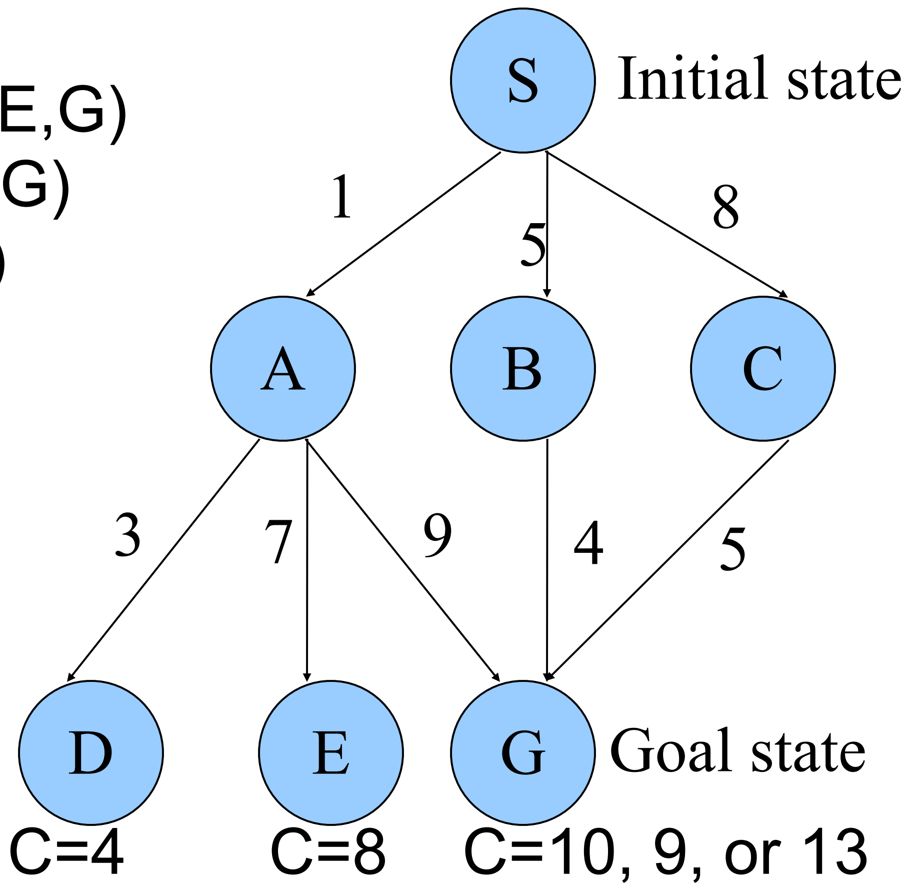
Expanded (Fringe):
1. S (A,B,C)
2. A (D,B,C,E,G)
3. D (B,C,E,G)
4. B (C,E,G)
5. C (E,G)



(All edges are directed, pointing downwards)

Example

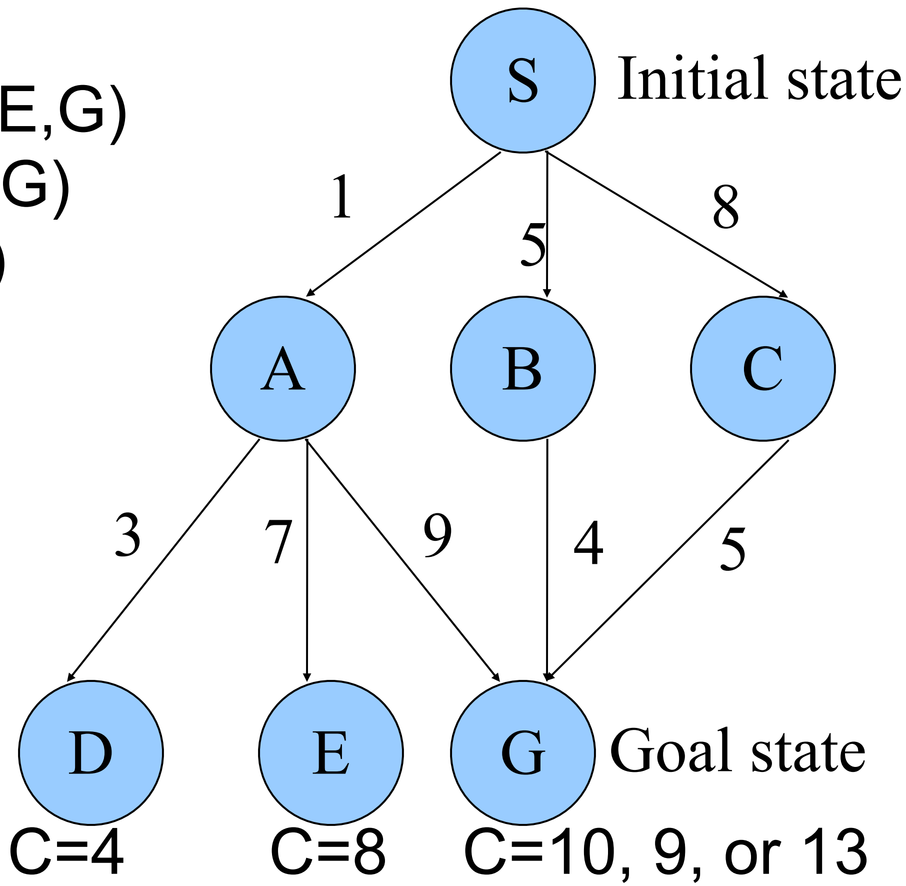
Expanded (Fringe):
1. S (A,B,C)
2. A (D,B,C,E,G)
3. D (B,C,E,G)
4. B (C,E,G)
5. C (E,G)
6. E (G)



(All edges are directed, pointing downwards)

Example

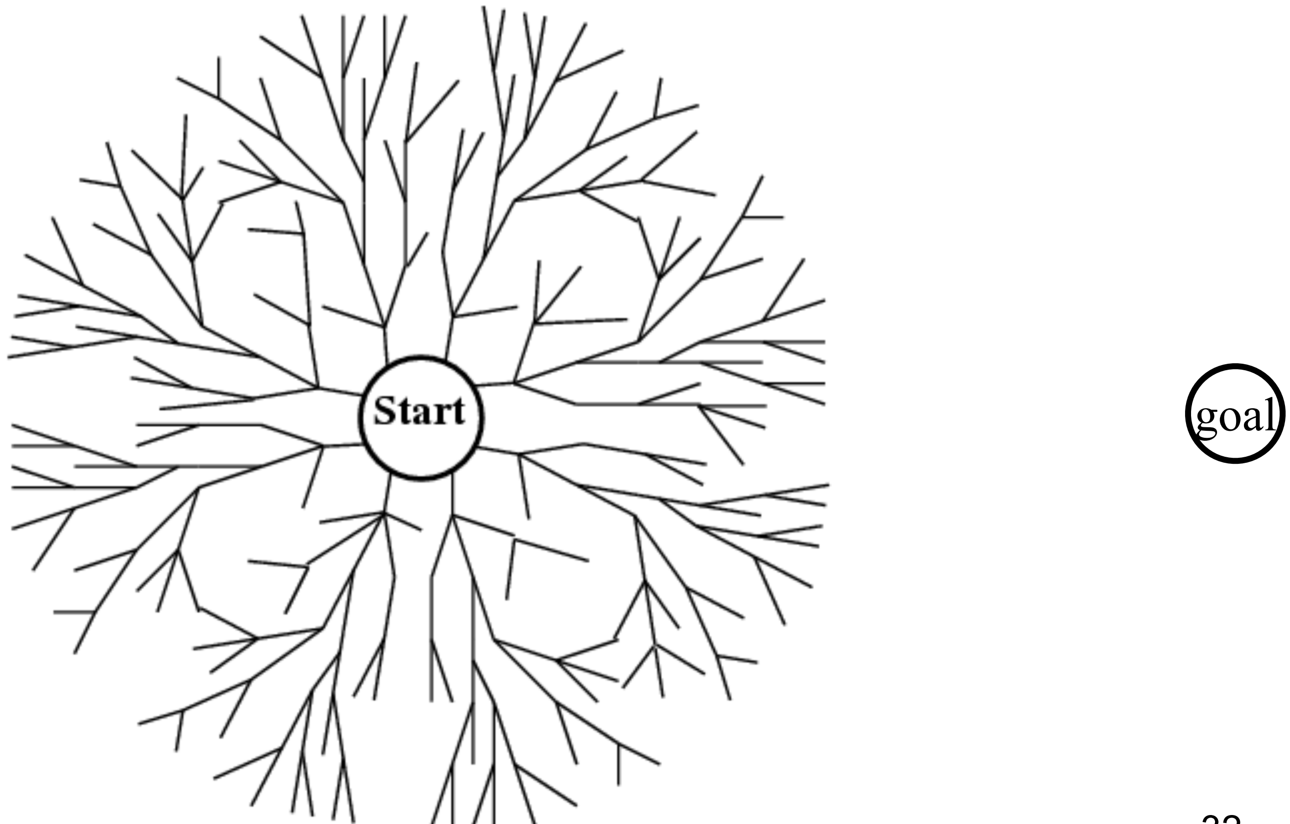
Expanded (Fringe):
1. S (A,B,C)
2. A (D,B,C,E,G)
3. D (B,C,E,G)
4. B (C,E,G)
5. C (E,G)
6. E (G)
7. G



(All edges are directed, pointing downwards)

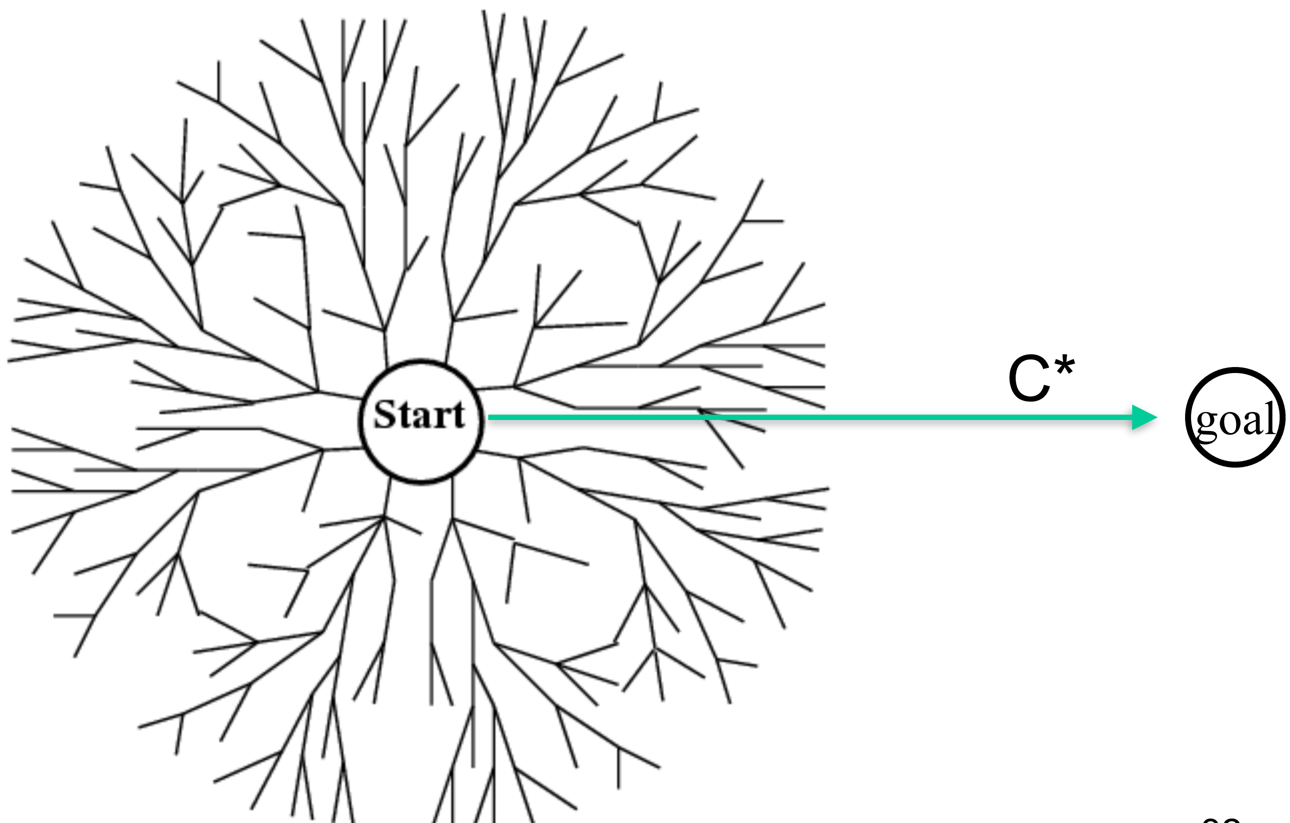
Uniform-cost search (UCS)

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



Uniform-cost search (UCS)

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



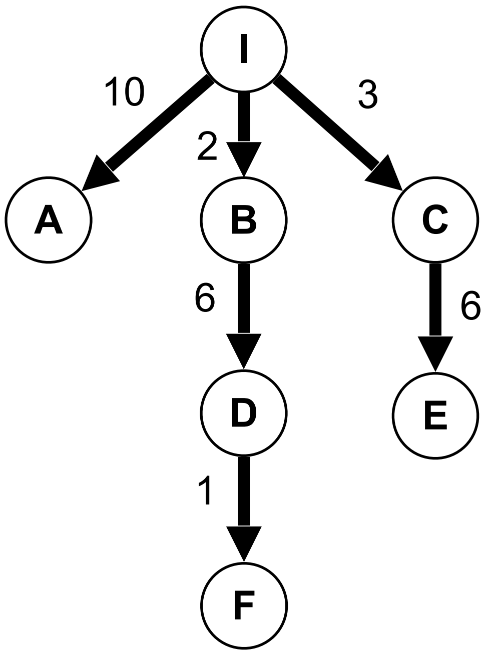
Performance of search algorithms on trees

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

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

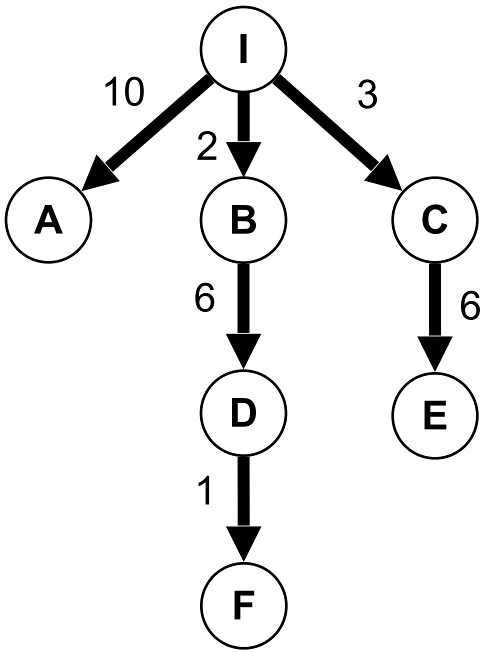
1. edge cost constant, or positive non-decreasing in depth
2. edge costs $\geq \varepsilon > 0$. C^* is the best goal path cost.

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?



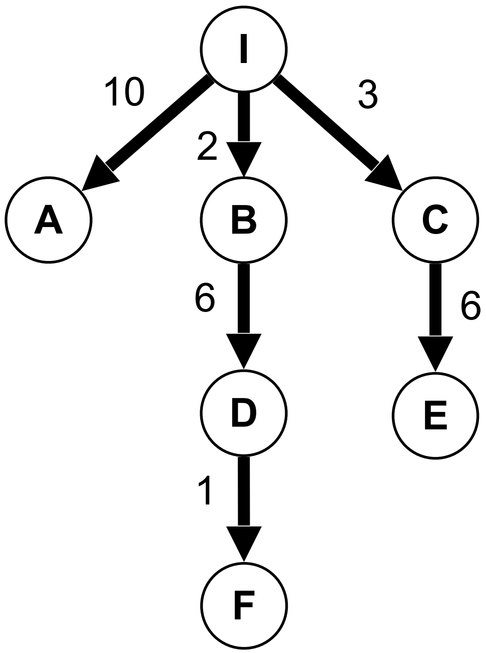
1. 2
2. 7
3. 8
4. 9

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?



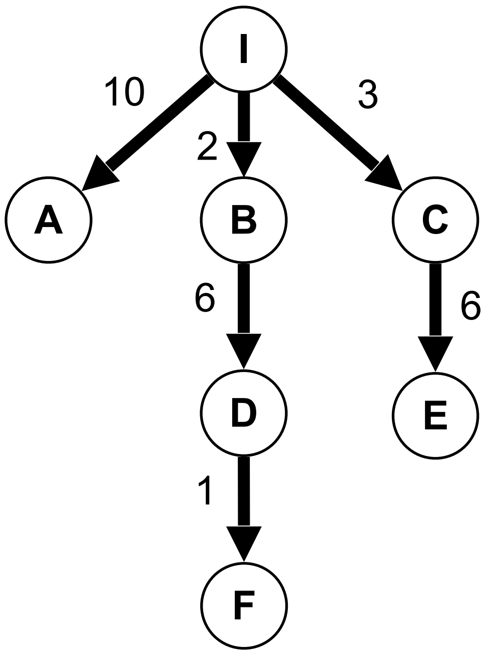
1. 2
2. 7
3. 8
4. 9

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?



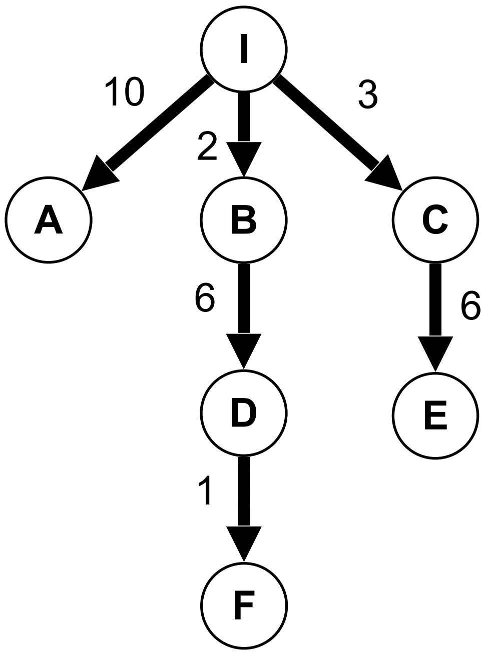
1. A
2. D
3. E
4. F

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

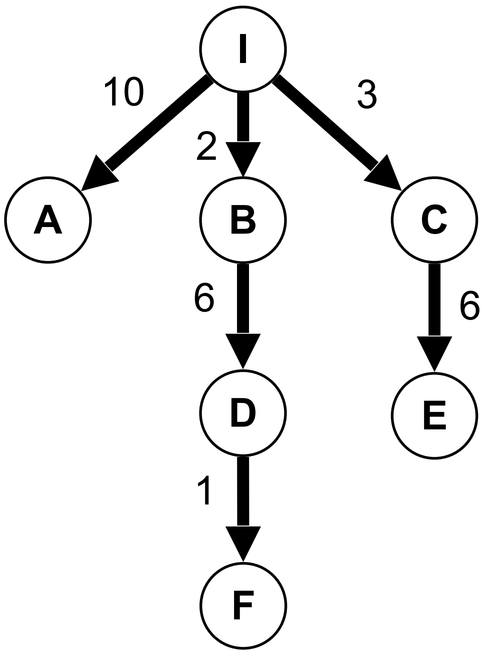
Q1-3: You are running UCS in the state space graph below. You just expanded (visited) node C. What node will the search expand next?



Expanded (Fringe):

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?

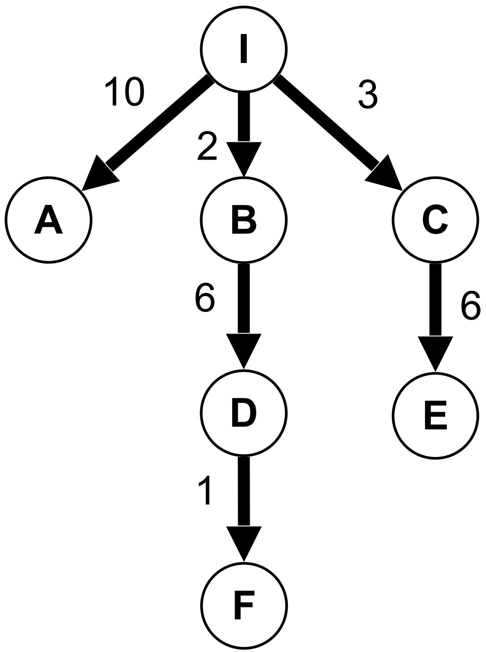


Expanded (Fringe):

1. I (A,B,C)

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

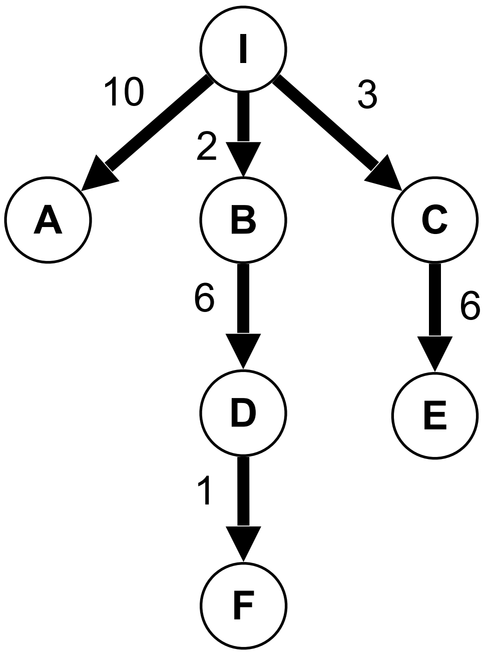
Q1-3: You are running UCS in the state space graph below. You just expanded (visited) node C. What node will the search expand next?



Expanded (Fringe):
1. I (A,B,C)
2. B (A,C,D)

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

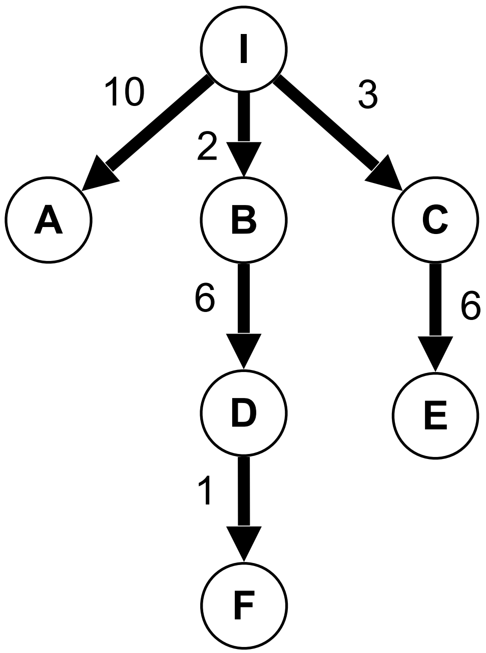
Q1-3: You are running UCS in the state space graph below. You just expanded (visited) node C. What node will the search expand next?



Expanded (Fringe):
1. I (A,B,C)
2. B (A,C,D)
3. C (A,D,E)

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?



Expanded (Fringe):

1. I (A,B,C)
2. B (A,C,D)
3. C (A,D,E)
4. D (A,E)

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"

 fringe = MAKE-QUEUE( MAKE-NODE( problem.INITIAL-STATE ) )
 loop
   if EMPTY( fringe ) then return "failure"
   node = REMOVE-FRONT(fringe)
   if problem.GOAL-TEST(node.STATE) succeeds:
     return node
   fringe = QUEUEING-FUNCTION(fringe, 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
```

Recall the bad space complexity 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 with depth), no otherwise.
- Time complexity (worst case): goal is the last node at radius ***d***.
  - Have to generate all nodes at radius ***d***.
  - $b + b^2 + \ldots + b^d \sim O(b^d)$
- Space complexity (bad, Figure 3.11)
  - Back points for all generated nodes $O(b^d)$
  - The queue (smaller, but still $O(b^d)$)

Solution: Uniform-cost search

Recall the bad space complexity 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 with depth), no otherwise.
- Time complexity (worst case): goal is the last node at radius ***d***.
  - Have to generate all nodes at radius ***d***.
  - $b + b^2 + \ldots + b^d \sim O(b^d)$
- Space complexity (bad, Figure 3.11)
  - Back points for all generated nodes $O(b^d)$
  - The queue (smaller, but still $O(b^d)$)

Solution: Uniform-cost search

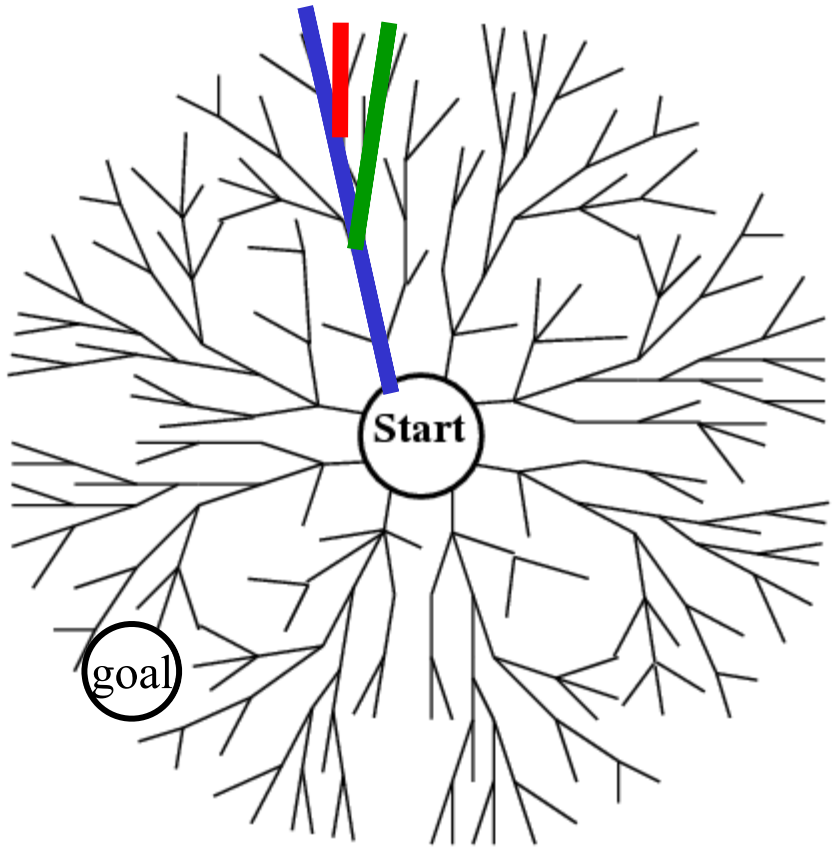
Solution: Depth-first search

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…

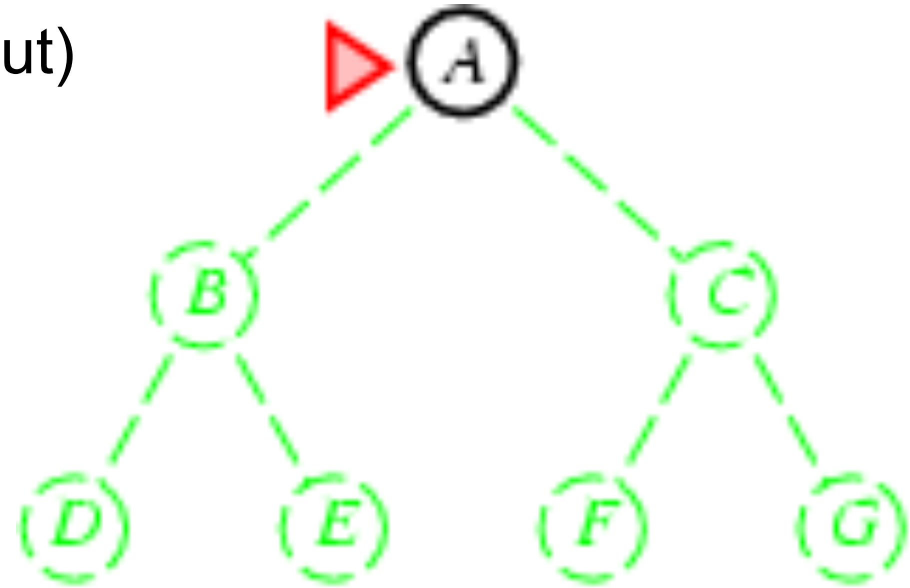
fan



Depth-first search (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



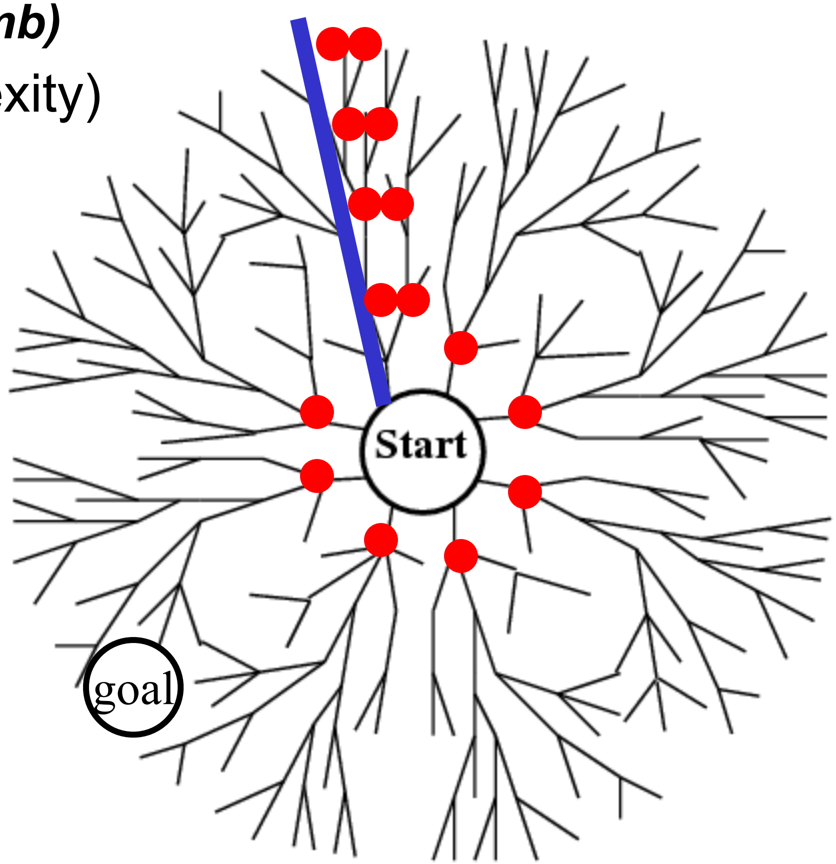
stack (fringe)

[] ⇔

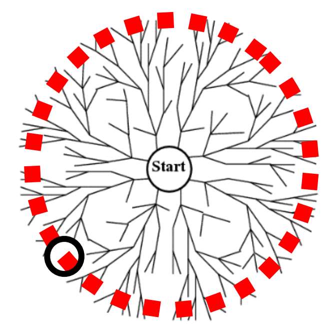
What's in the fringe for DFS?

- m = maximum depth of graph from start
- ***m(b-1) ~ O(mb)***

(Space complexity)



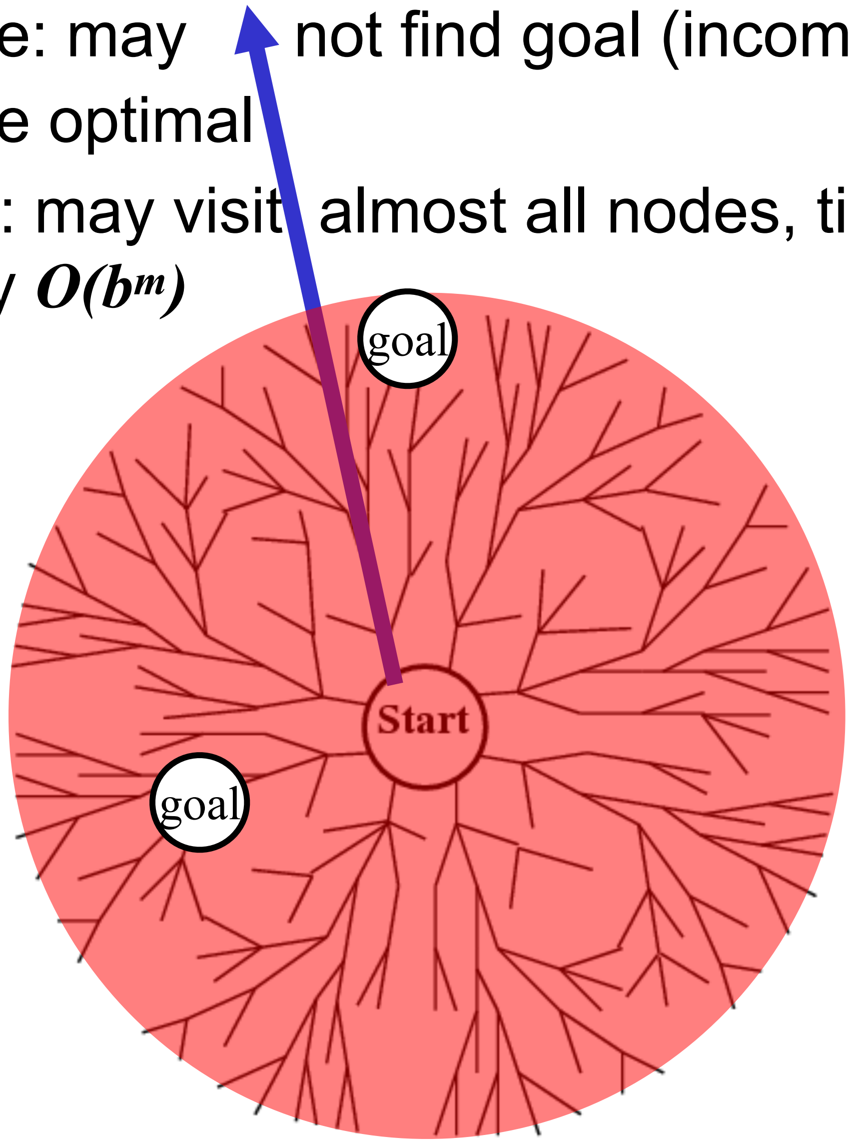
c.f. BFS $\boldsymbol{O(b^d)}$



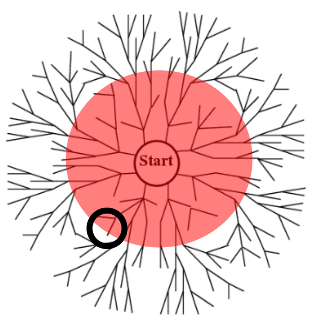
- "backtracking search" even less space
  - generate siblings (if applicable)

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 $O(b^d)$



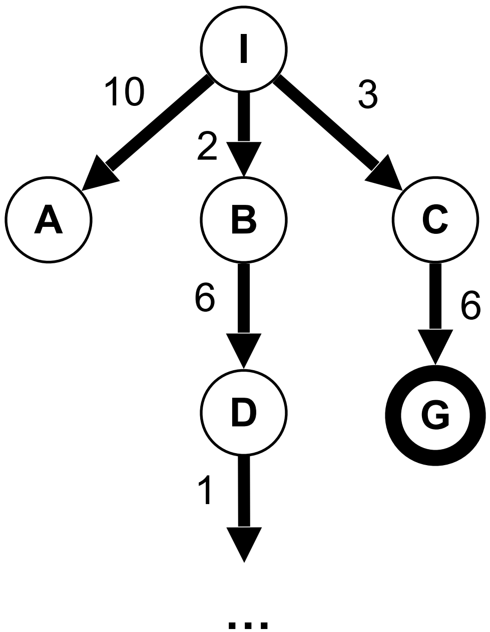
Performance of search algorithms on trees

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

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

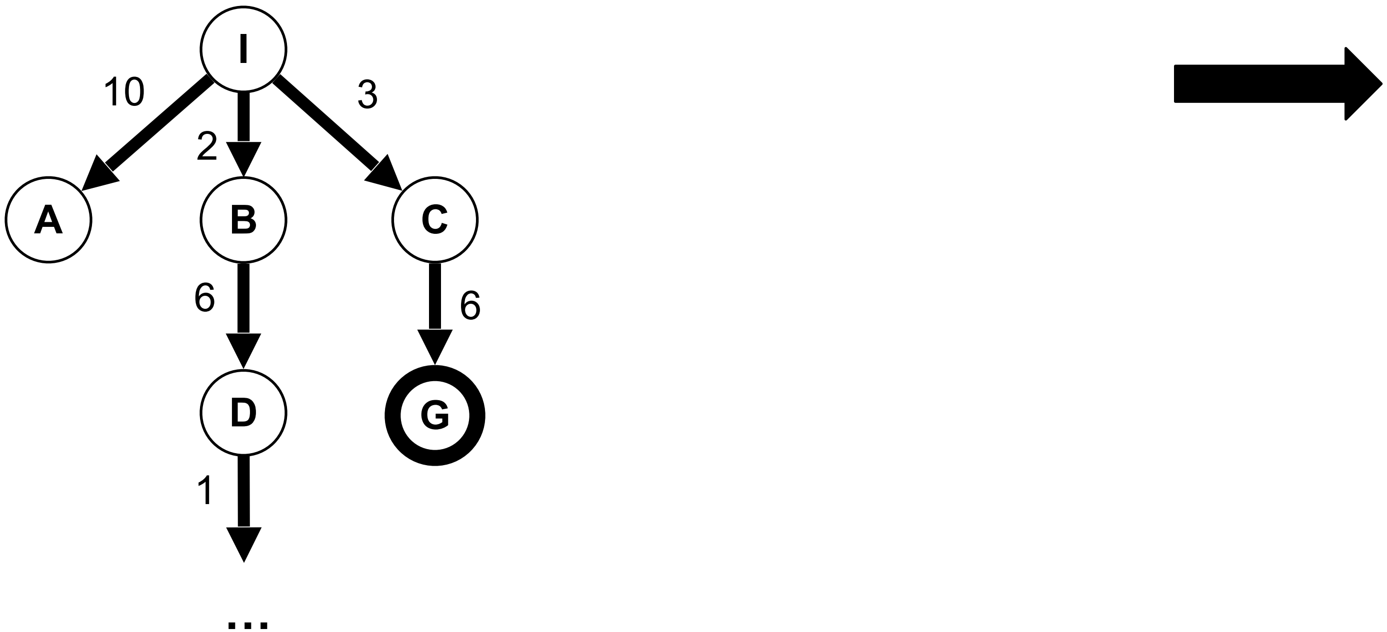
1. edge cost constant, or positive non-decreasing in depth
2. edge costs $\geq \varepsilon > 0$. C^* is the best goal path cost.

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

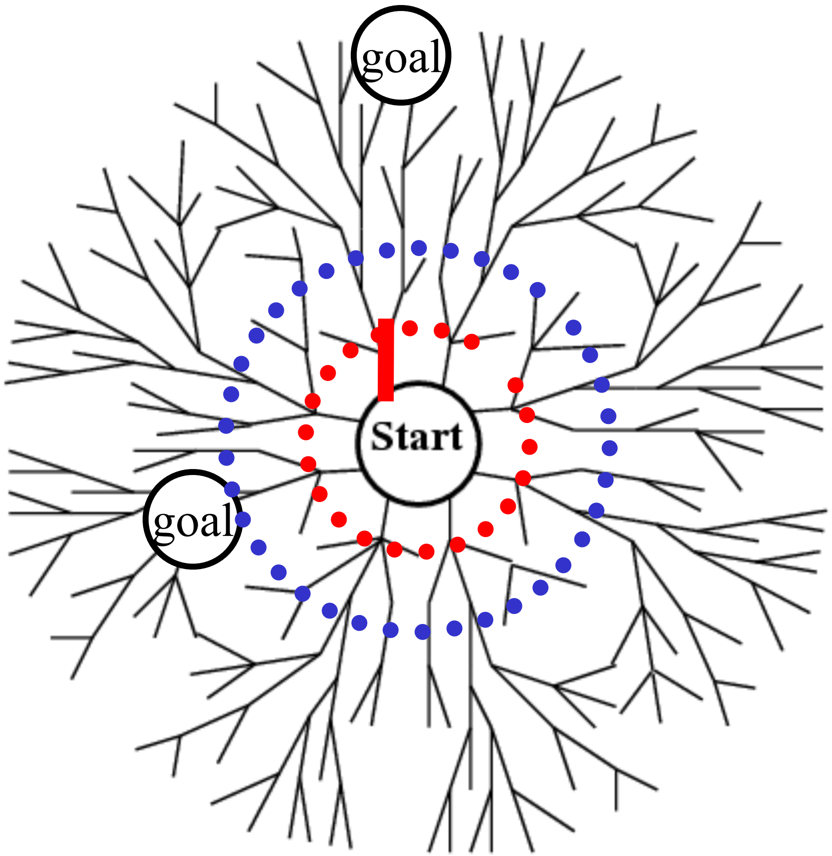
Q2-2: You need to search a randomly generated state space graph with one goal, uniform edges costs, d=2, and m=100. Considering worst case behavior, do you select BFS or DFS for your search?

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

Performance of search algorithms on trees

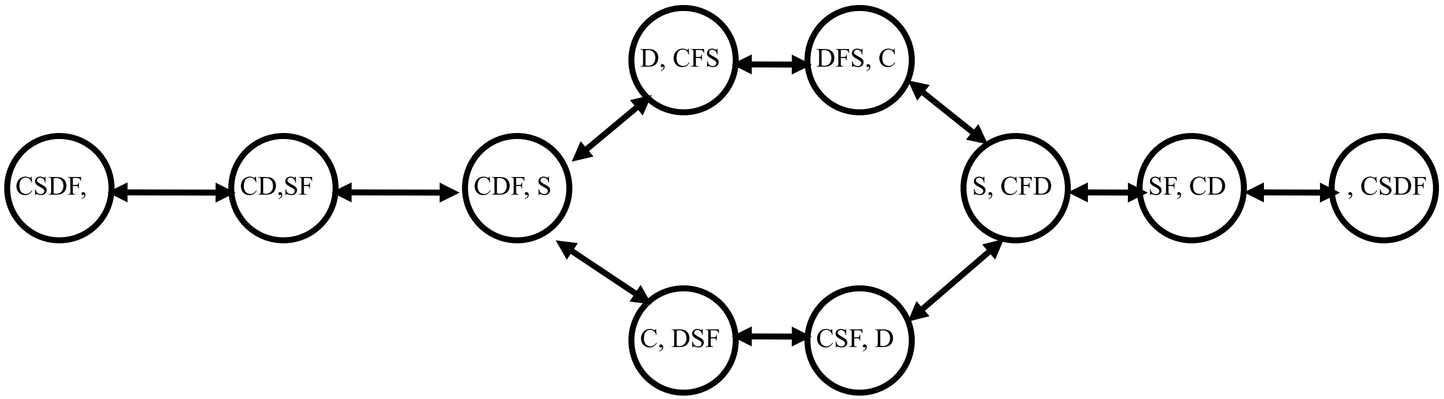
b: branching factor (assume finite)   d: goal depth   m: graph depth

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

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

If state space graph is not a tree

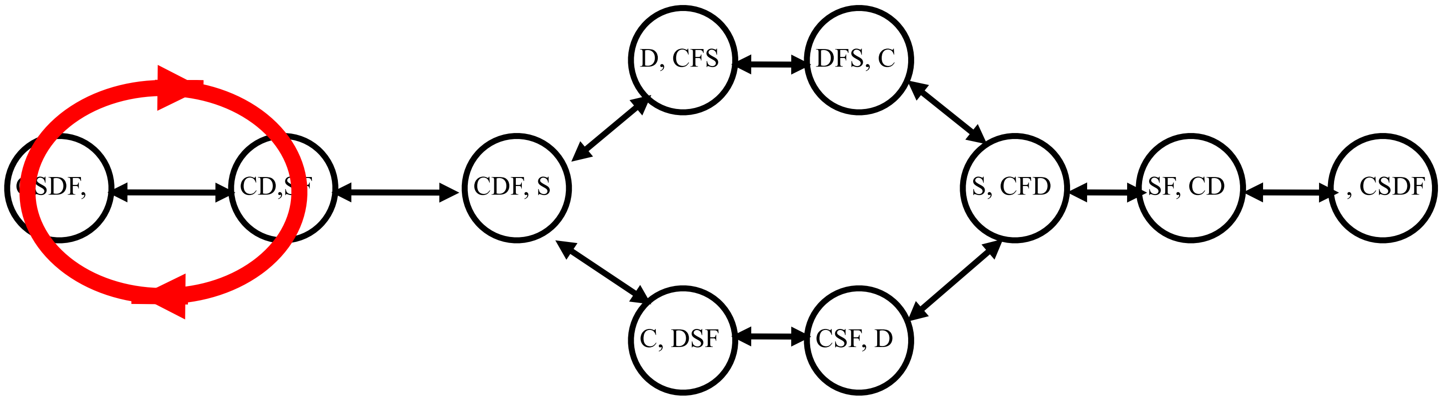
- The problem: repeated states



- 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



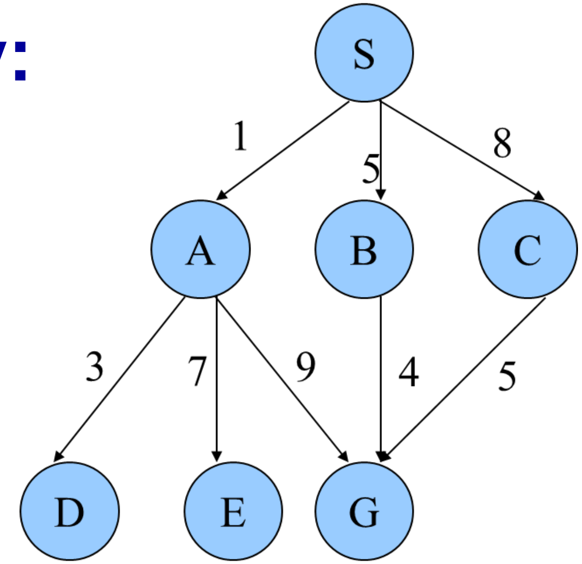
- Ignore the danger of repeated states: wasteful (BFS) or impossible (DFS). Can you see why?
- How to prevent it?

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.

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.)

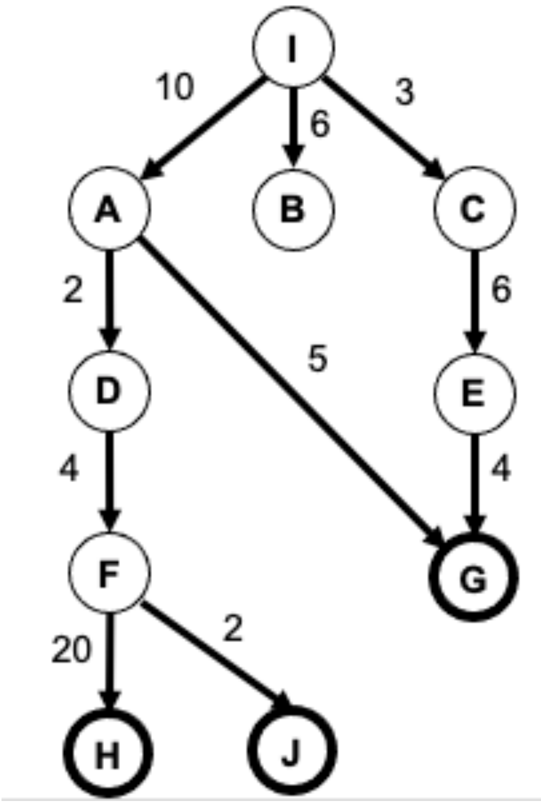
- Depth-First Search: S A D E G

  Solution found: S A G

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

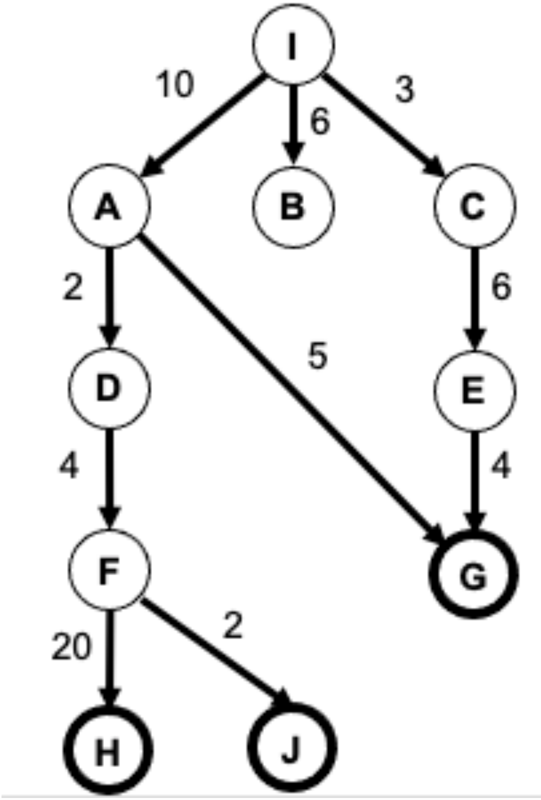
  Solution found: S A G

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?



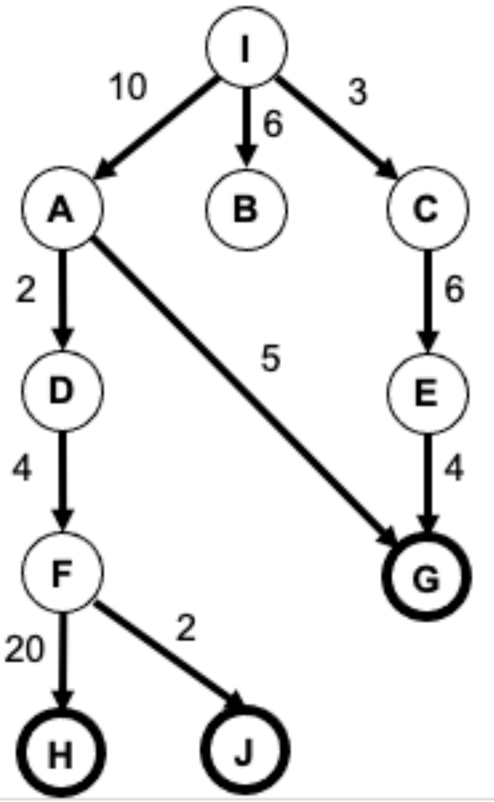
1. IADFH
2. IADFJ
3. IAG
4. ICEG

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?



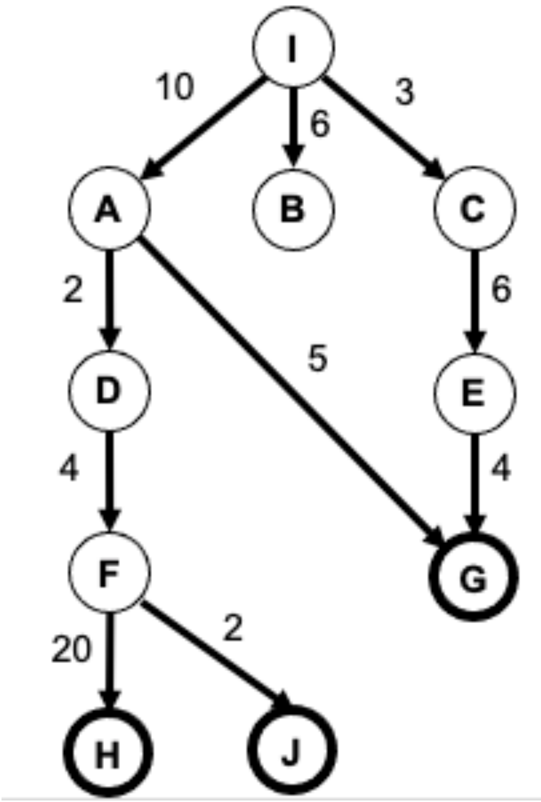
1. IADFH
2. IADFJ
3. IAG
4. ICEG

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?



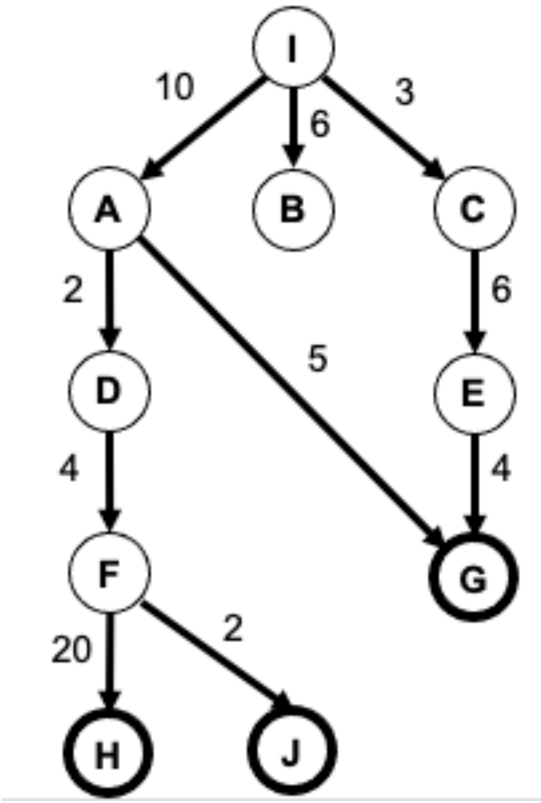
1. IADFH
2. IADFJ
3. IAG
4. ICEG

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?



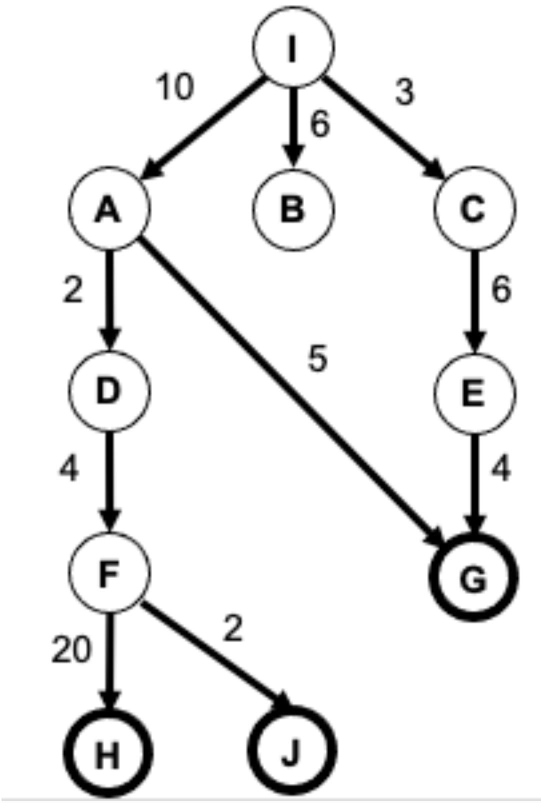
1. IADFH
2. IADFJ
3. IAG
4. ICEG

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?



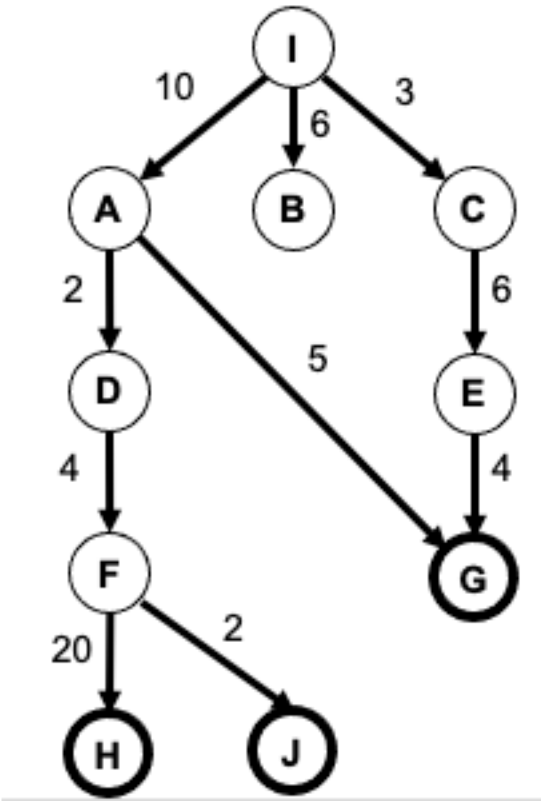
1. IADFH
2. IADFJ
3. IAG
4. ICEG

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?



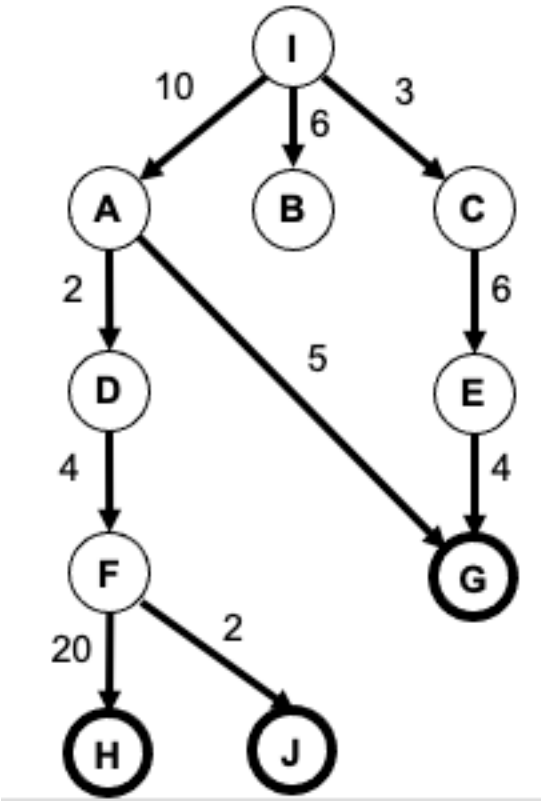
1. IADFH
2. IADFJ
3. IAG
4. ICEG

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