CS540 Intro to AI
Uninformed Search II

University of Wisconsin–Madison

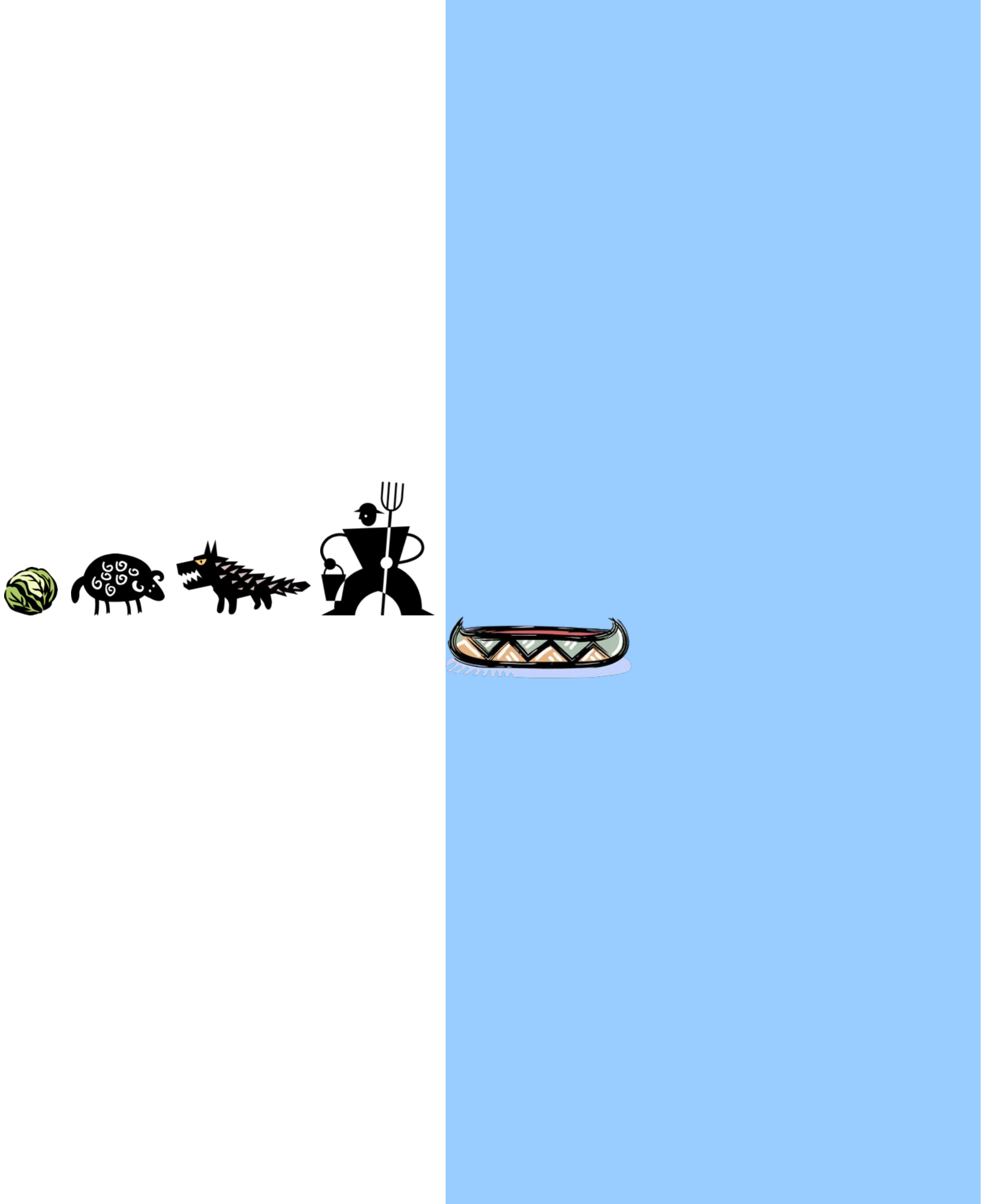
Fall 2025, Section 3
November 5, 2025

Many AI problems can be formulated as search.

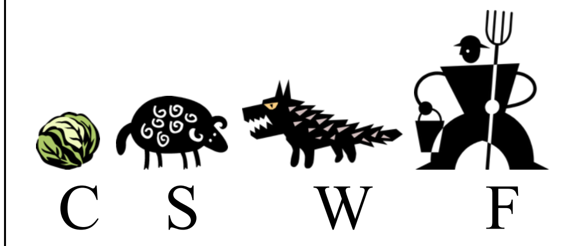
How to make a sequence of decisions to reach a desired goal.

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

“How the world changes in response to agent actions”

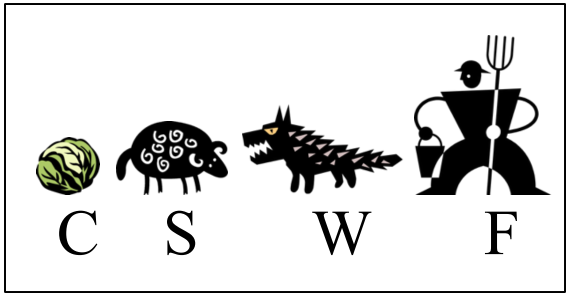


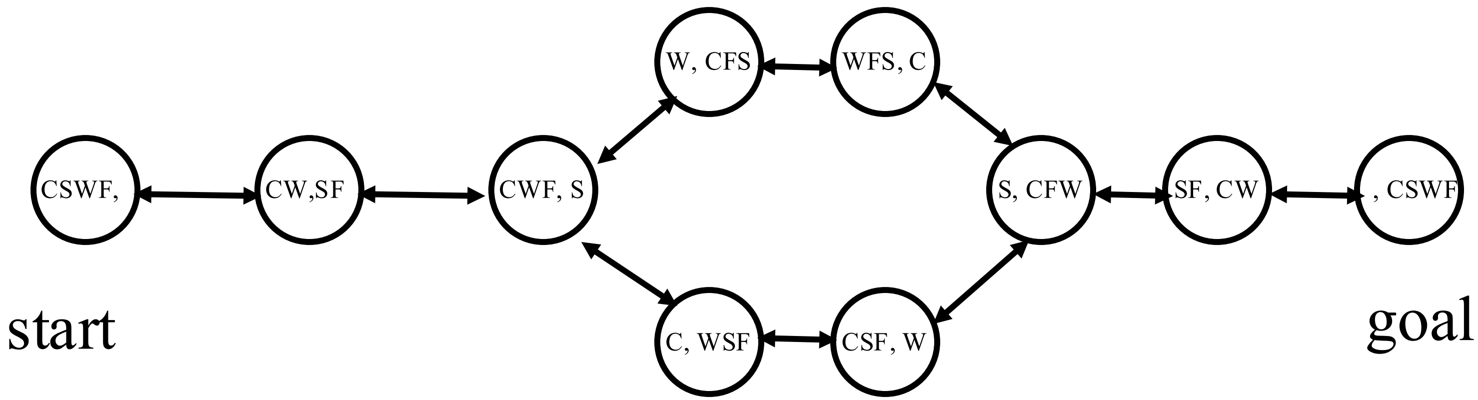
The search problem



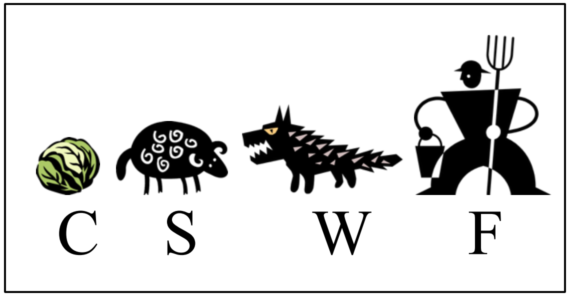
- State space ***S*** : all valid configurations
- Initial state ***I*** = {(CSWF,)} $\subseteq$ ***S***
- Goal state ***G*** = {(,CSWF)} $\subseteq$ ***S***
- Successor function ***succs***(s) $\subseteq$ ***S*** : states reachable in one step from state s
  - ***succs***((CSWF,)) = {(CW, SF)}
  - ***succs***((CWF,S)) = {(CW,FS), (W,CFS), (C, WFS)}
- 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.

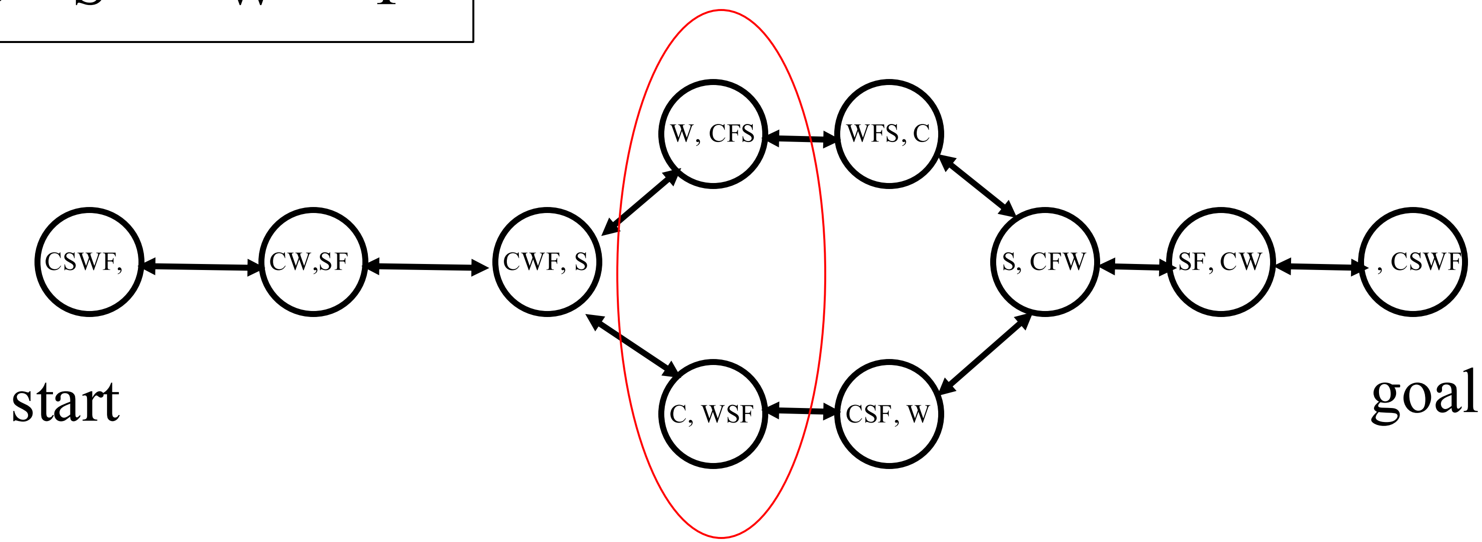
A directed graph in state space





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

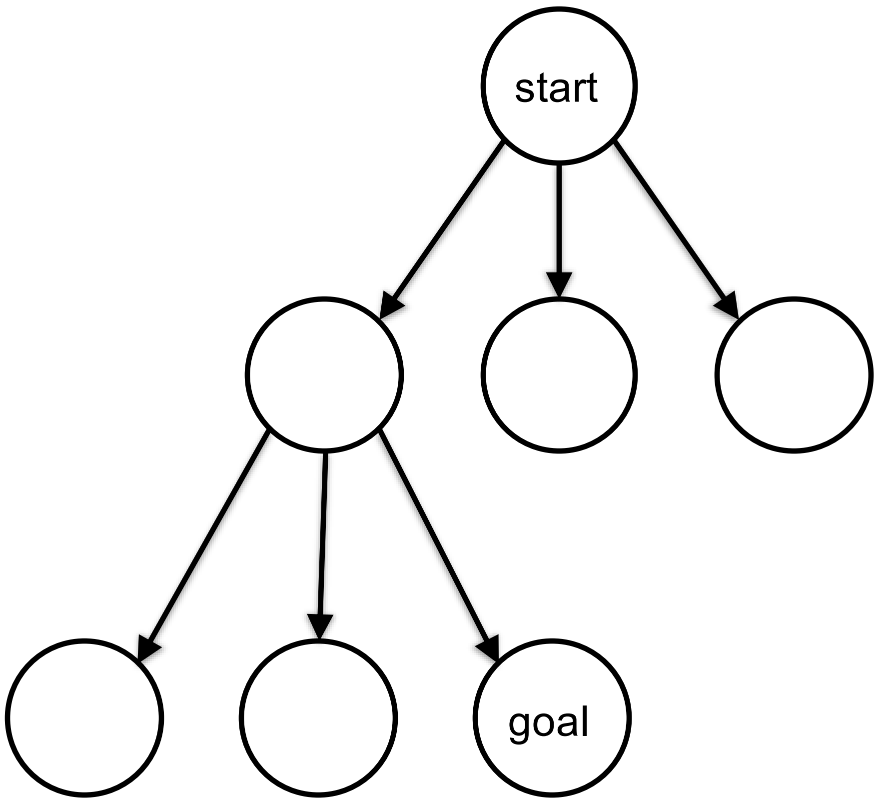
Uninformed search on trees

- 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

Uninformed search on trees

- Assume graph of state space is a tree
  - Only one path from start to each node
  - Could be infinite

- Assume every state has exactly b successor states



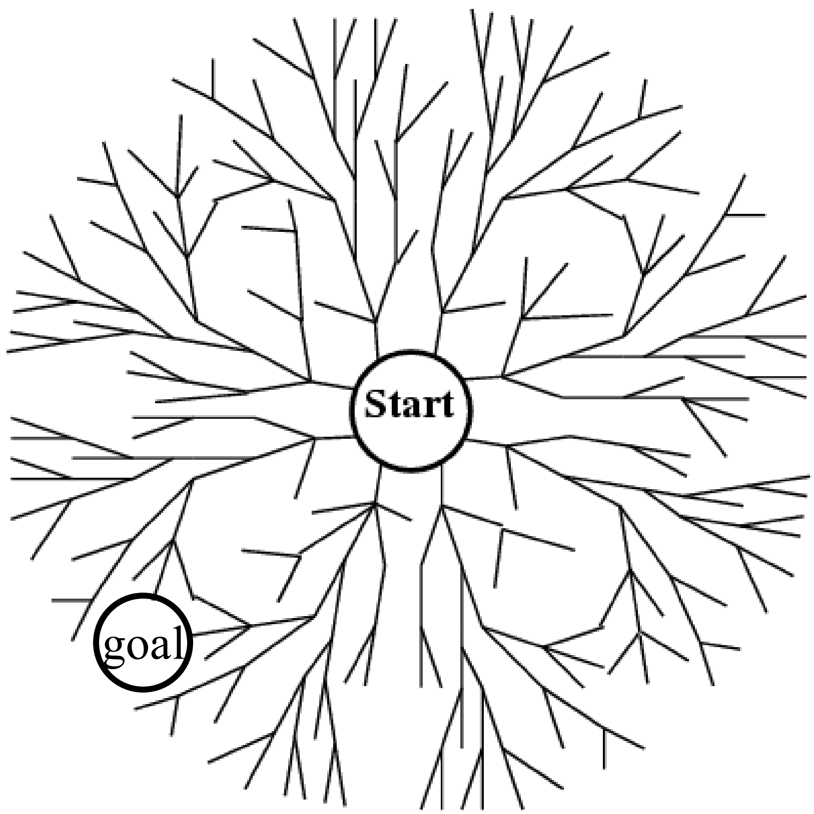
Goal depth d

Max depth m

Uninformed search on trees

- Assume graph of state space is a tree
  - Only one path from start to each node
  - Could be infinite
- Assume every state has exactly b successor states

Alternative Picture:

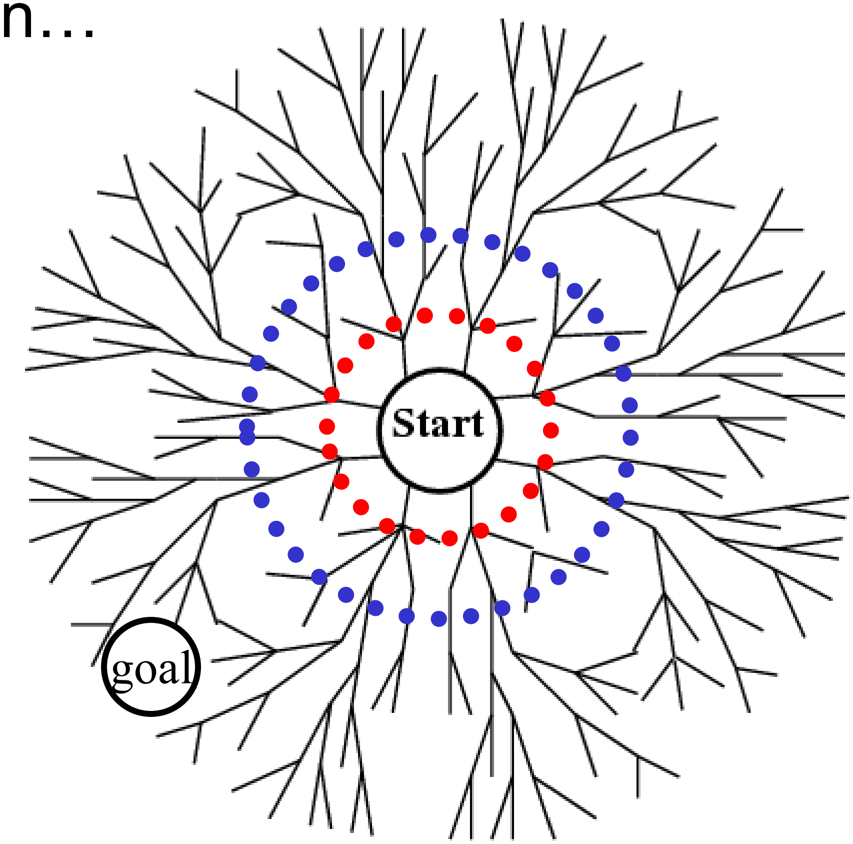


Breadth-first search (BFS)

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…

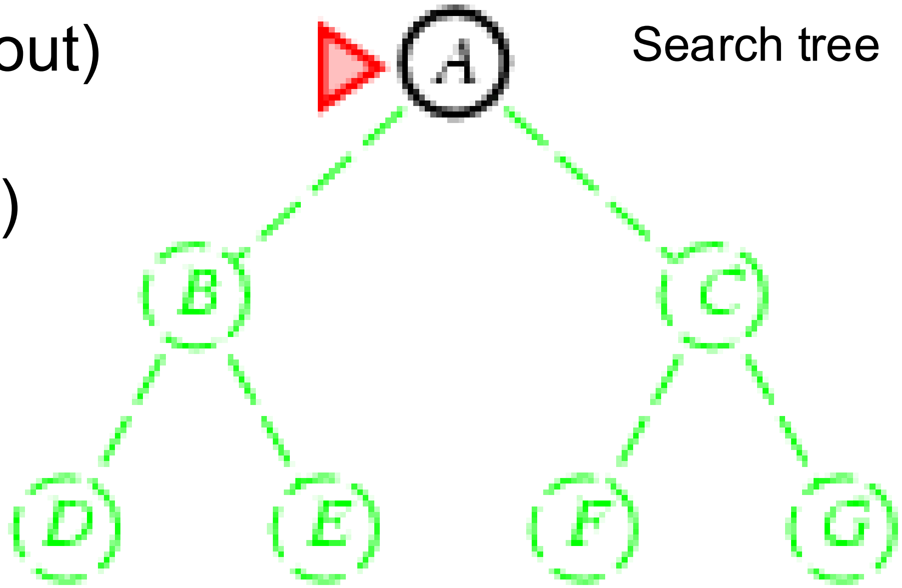
ripple



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



Search tree

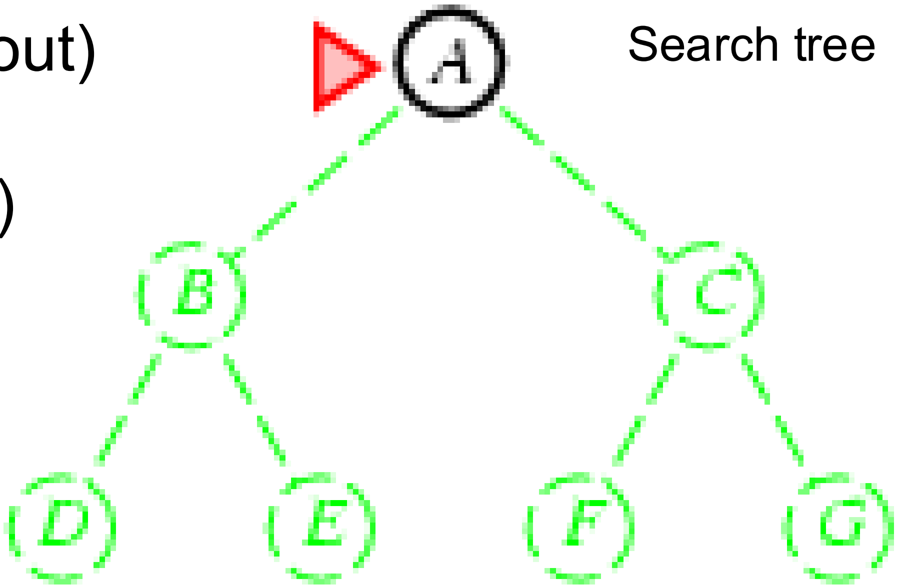
Initial state: **A**
Goal state: **G**

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] →

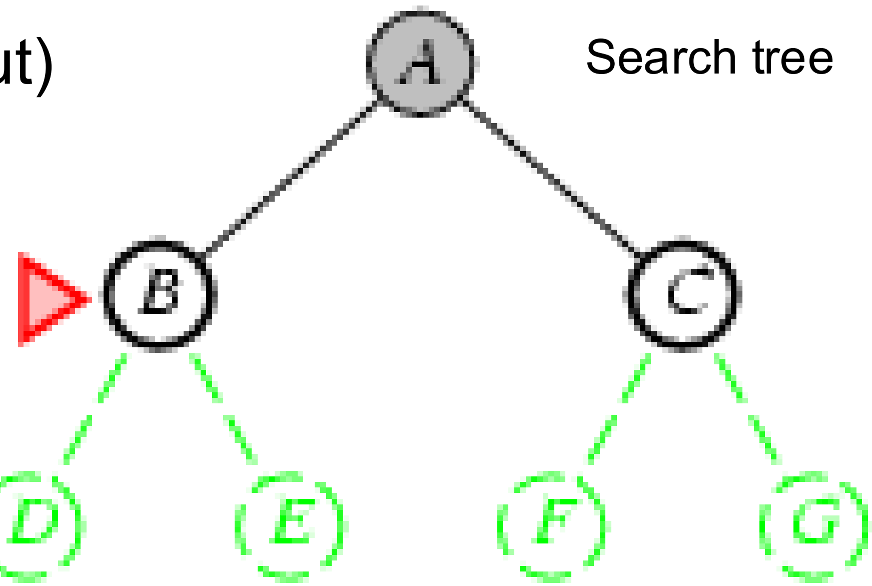
Initial state: **A**

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

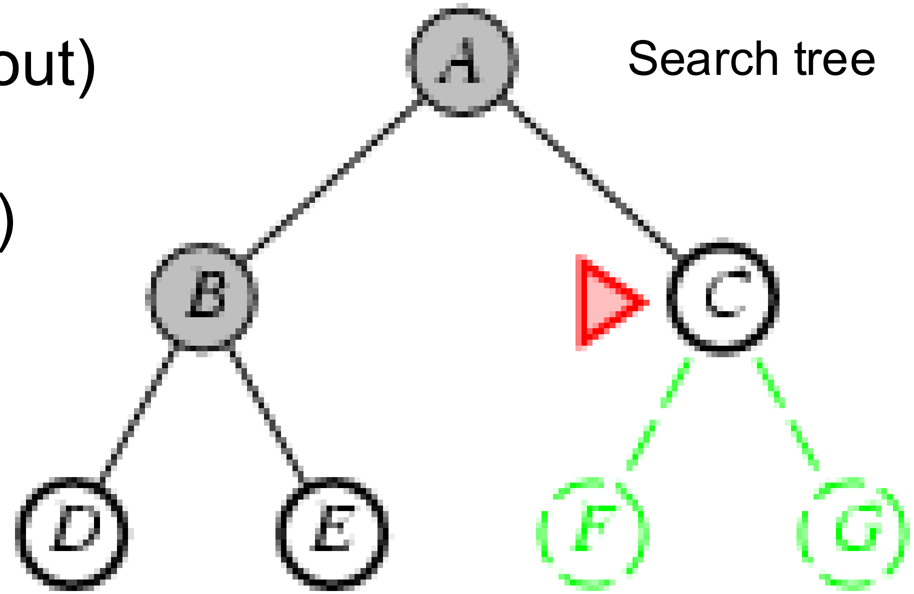
queue (fringe, OPEN)
→ [CB] → A

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



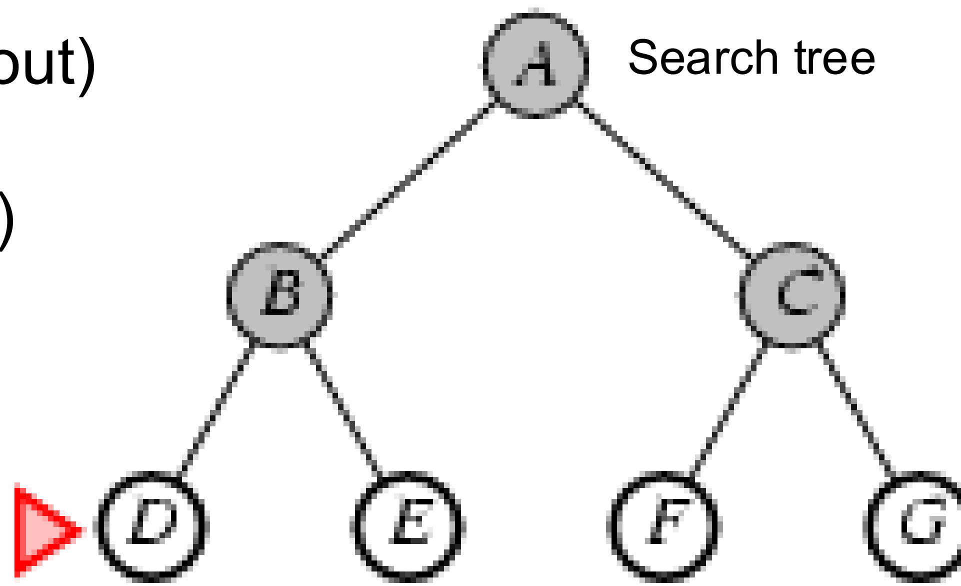
queue (fringe, OPEN)
→ [EDC] → B

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

Breadth-first search (BFS)

Use a queue (First-in First-out)

```
en_queue(Initial states)
While (queue not empty)
    s = de_queue()
    if (s==goal) success!
    T = succs(s)
    en_queue(T)
endWhile
```



queue (fringe, OPEN)
□[GFED] → C

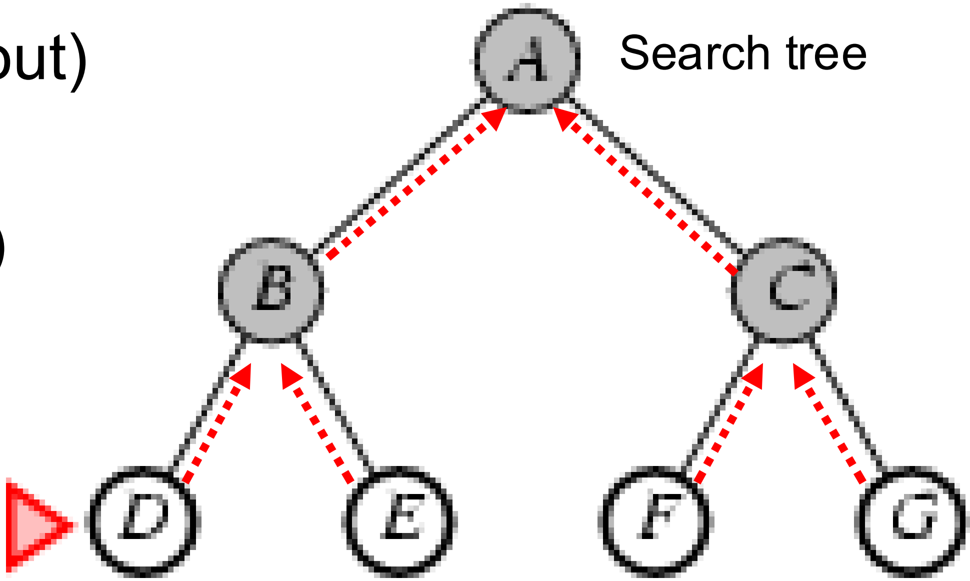
Initial state: **A**
Goal state: **G**

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

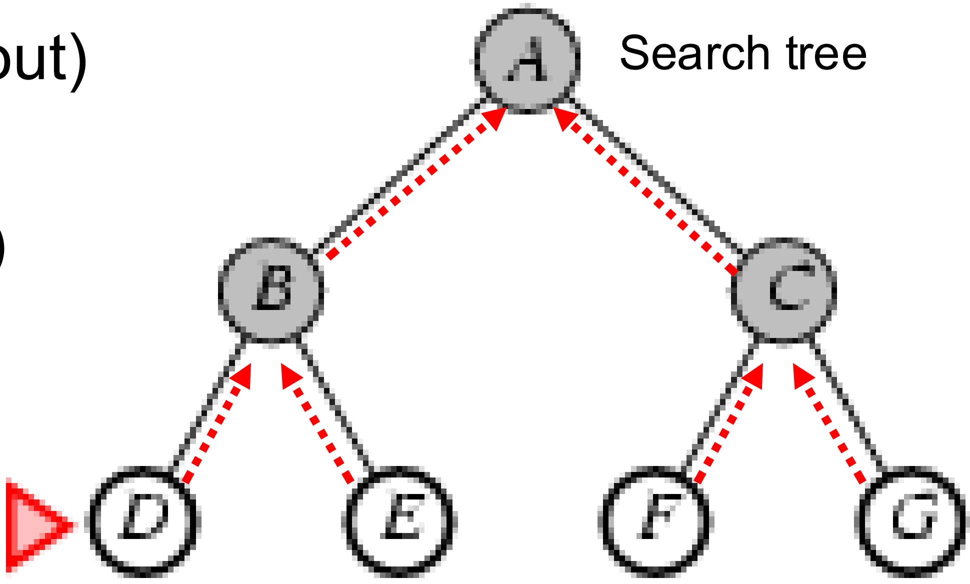
... until much later we pop G.

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

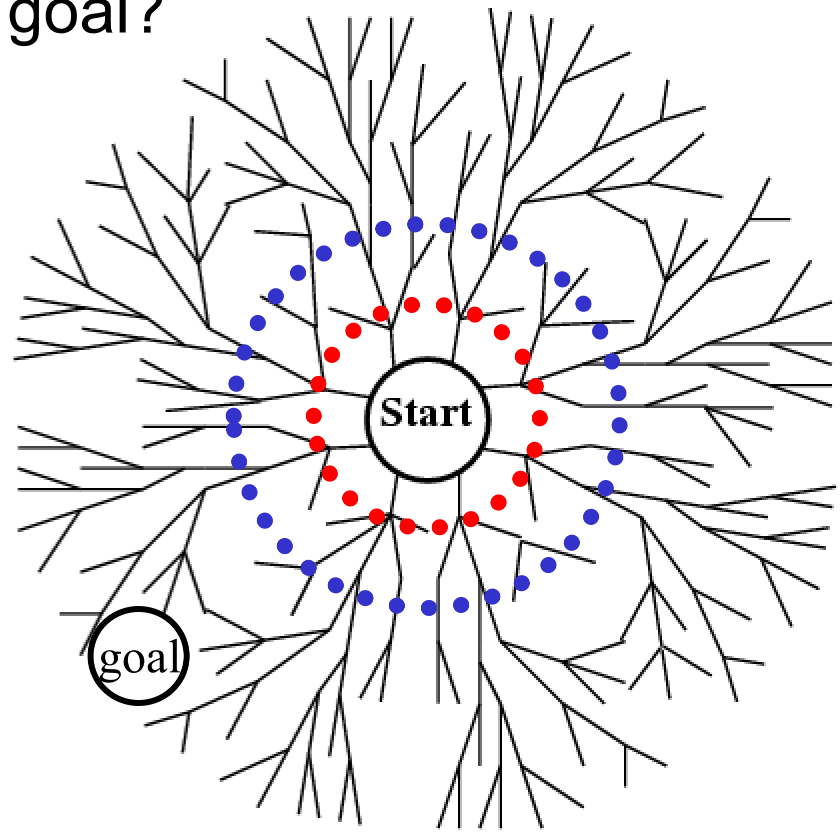
... until much later we pop G.

We need back pointers to recover the solution path.

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

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



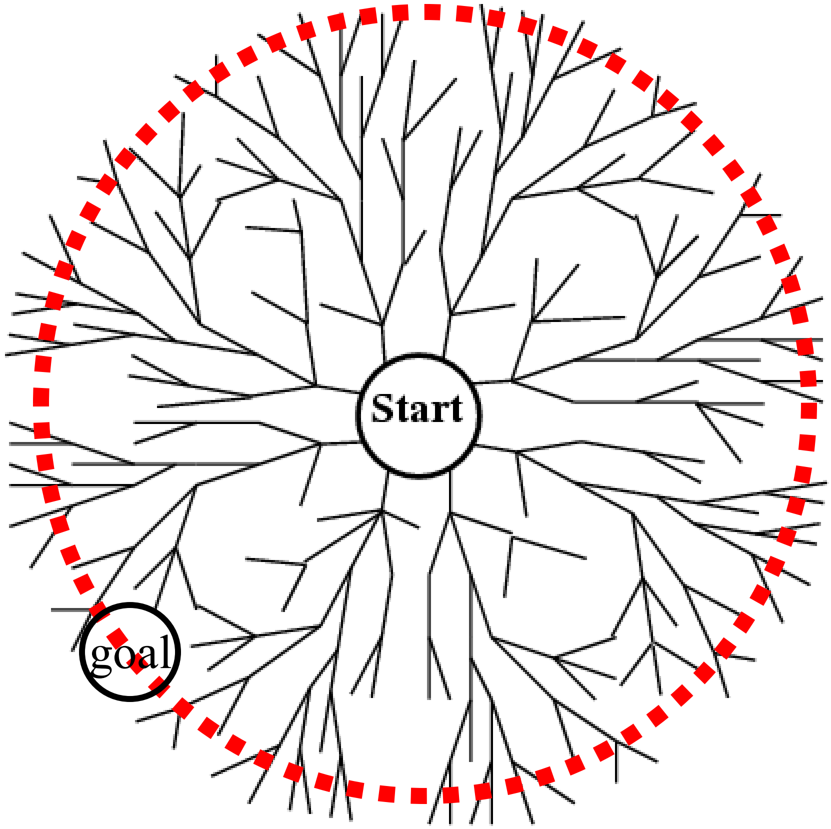
Performance of BFS

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

Performance of BFS

Four measures of search algorith

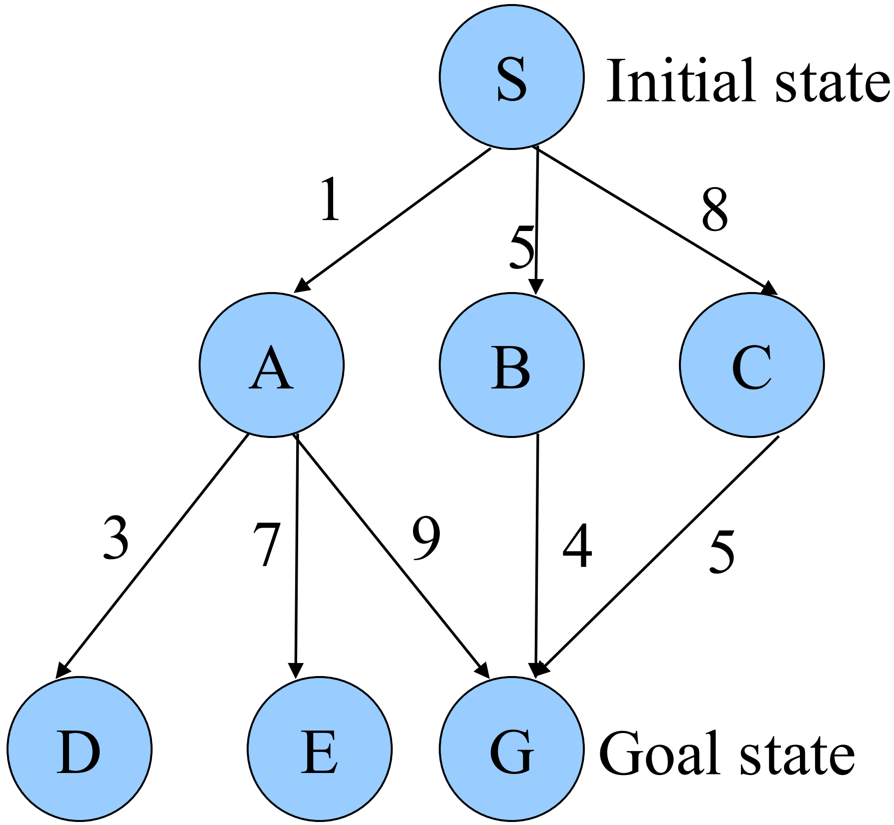
Solution: Uniform-cost search

- Completeness (not finding all g S will find a goal.
- Optimality: yes if edges cost 1 (mo 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 $O(b^d)$)

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

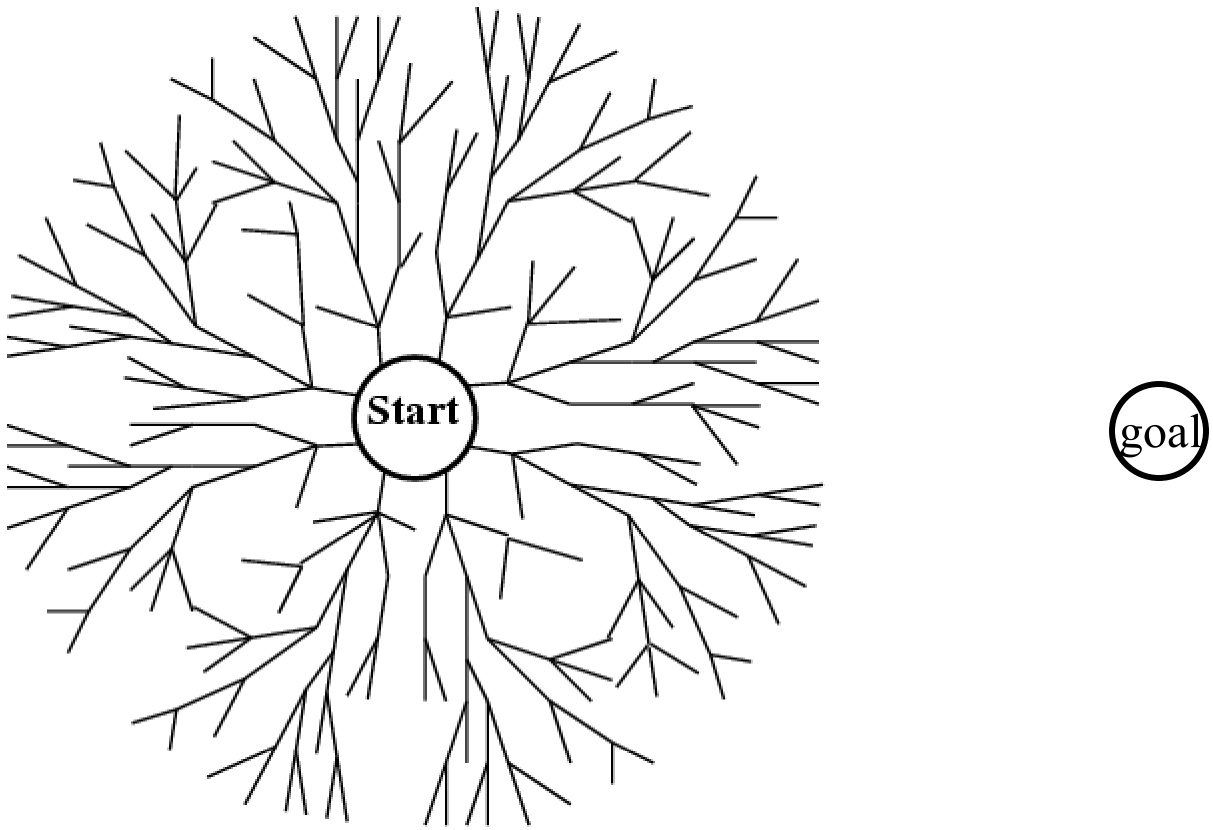


1: (S,0), [(A,1), (B,5), (C,8)]
2: (A,1), [(B,5), (C,8), (D,4), (E,8), (G,10)]
3: (D,4), [(B,5), (C,8), (E,8), (G,10)]
4: (B,5), [(C,8), (E,8), (G,9)]
5: (C,8), [(E,8), (G,9)]
6: (E,8), [(G,9)]
7: (G,9), []: Success!

(All edges are directed, pointing downwards)

Uniform-cost search (UCS)

- Complete and optimal (if edge costs $\geq \varepsilon > 0$)
- 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.

Performance of BFS

Four measures of search algorith

- Completeness (not finding all g S will find a goal.
- Optimality: yes if edges cost 1 (mo generally positive non-decreasing in depth), no otherwise.
- Time complexit goal is the last node at radius ***d***.
  - Have to generate all es at radius ***d***.
  - ***b + b² + … + bᵈ ~ O(bᵈ)***
- Space complexity (bad)
  - Back pointers for all generated nodes ***O(bᵈ)***
  - The queue / fringe (smaller, but still ***O(bᵈ)***)

Solution: Uniform-cost search

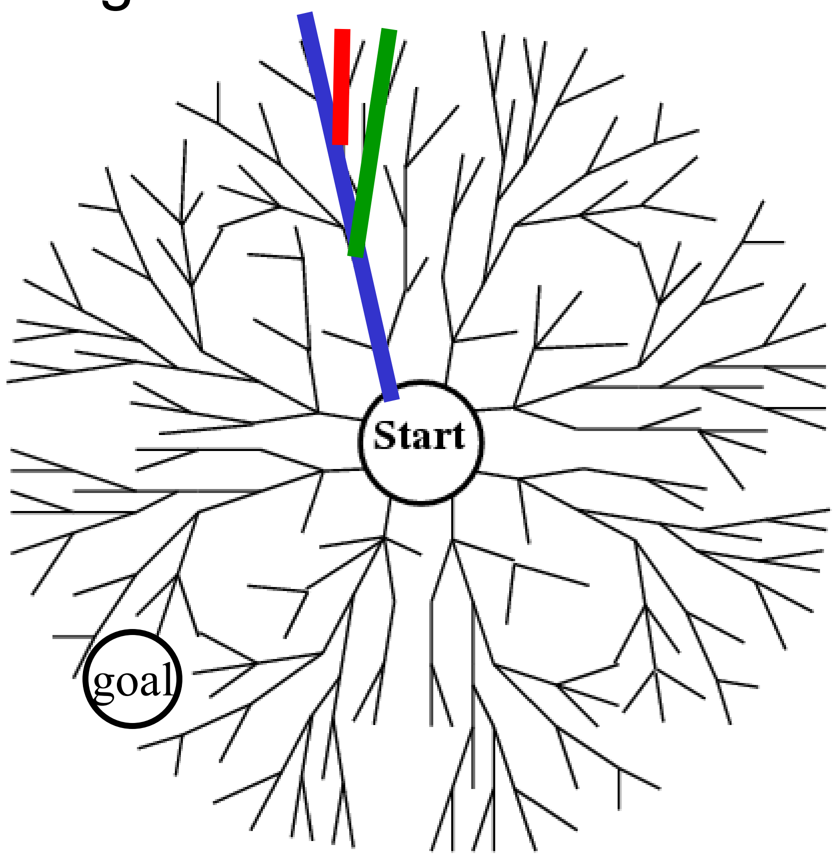
Solution: Depth-first search

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…

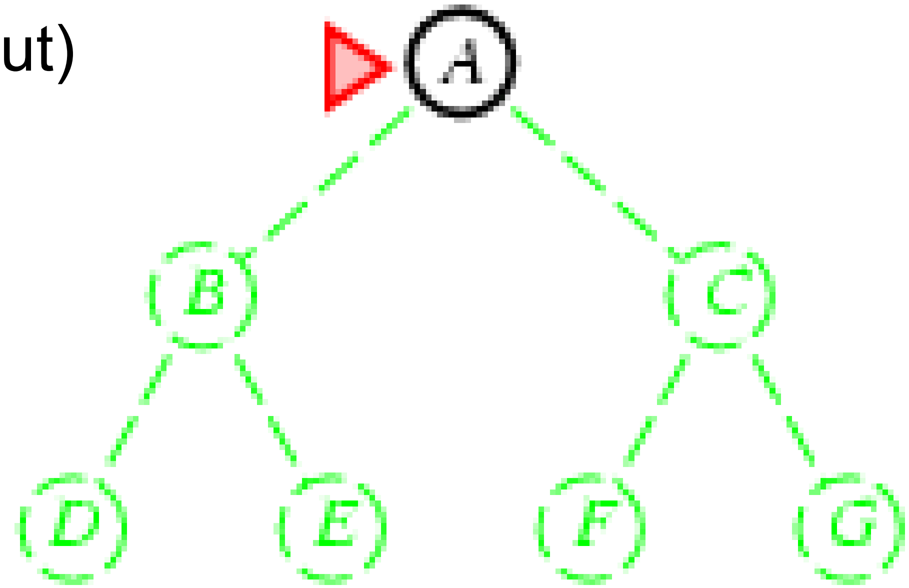
fan



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



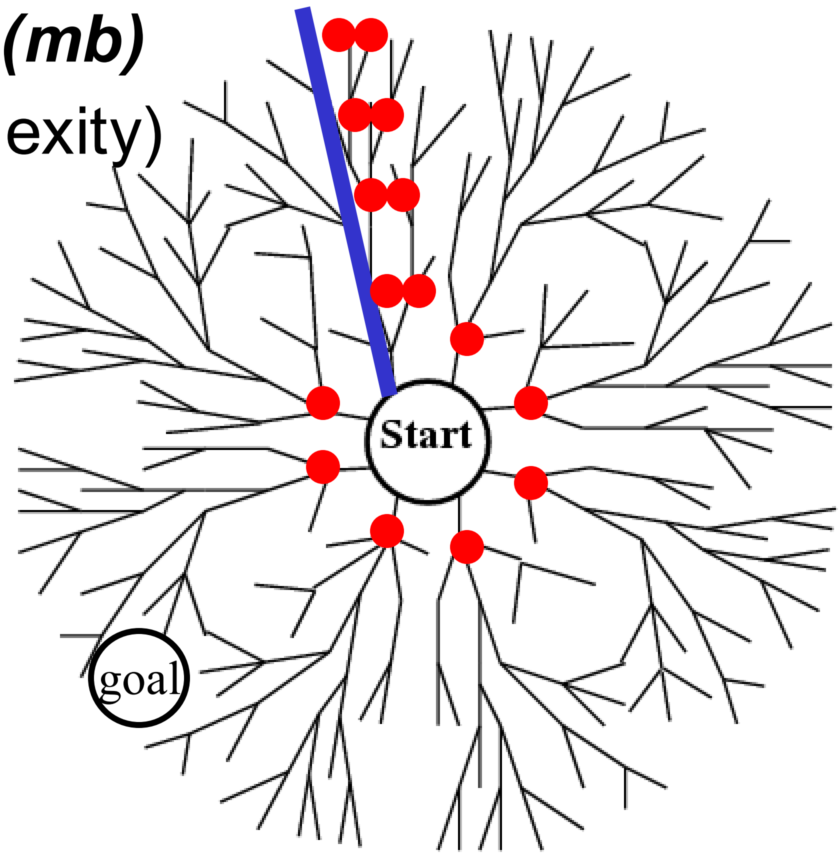
stack (fringe)

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

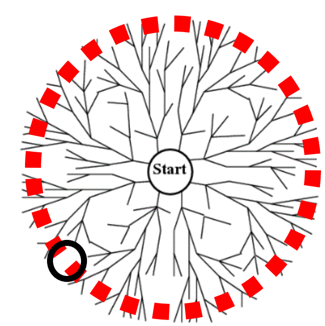
What's in the fringe for DFS?

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

(Space complexity)



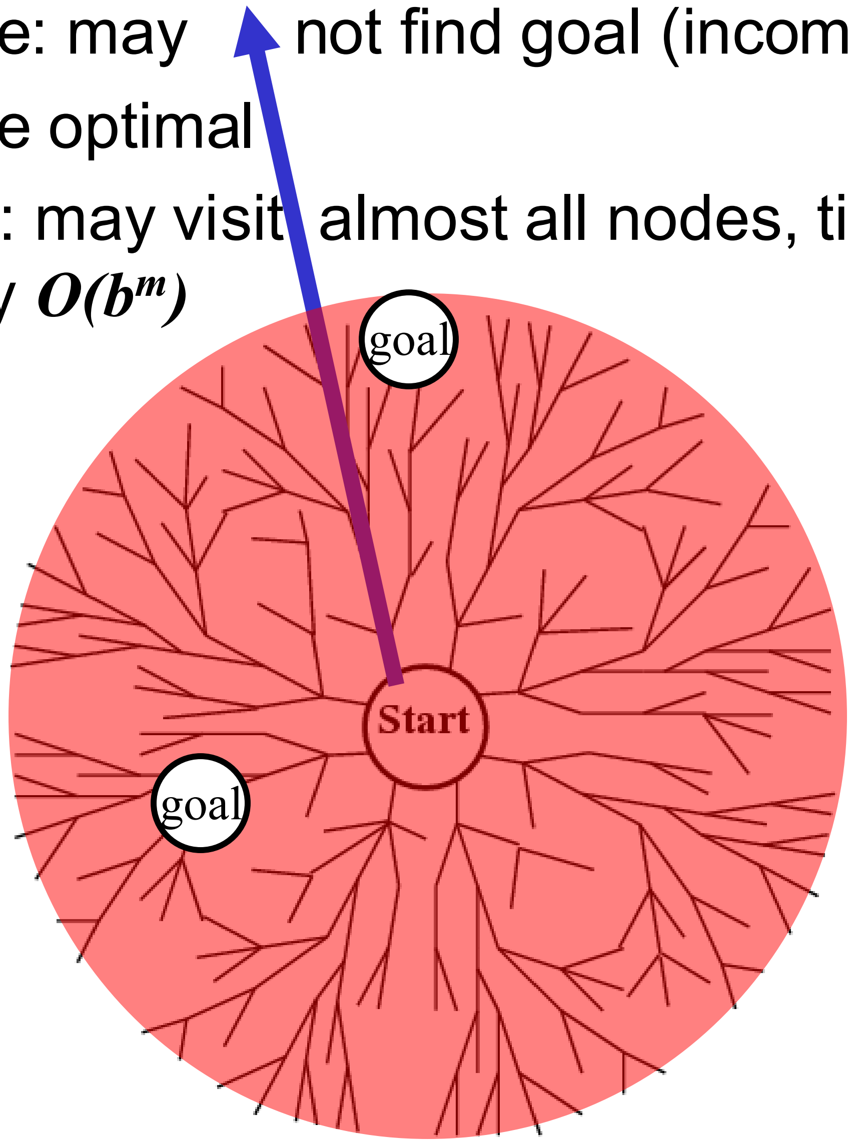
c.f. BFS $O(b^d)$



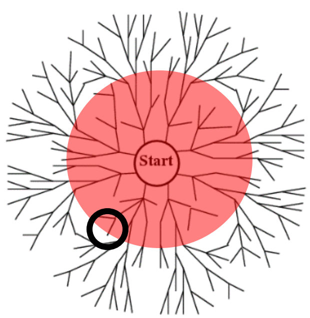
- "backtracking search" even less space
  - generate siblings (if applicable)

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

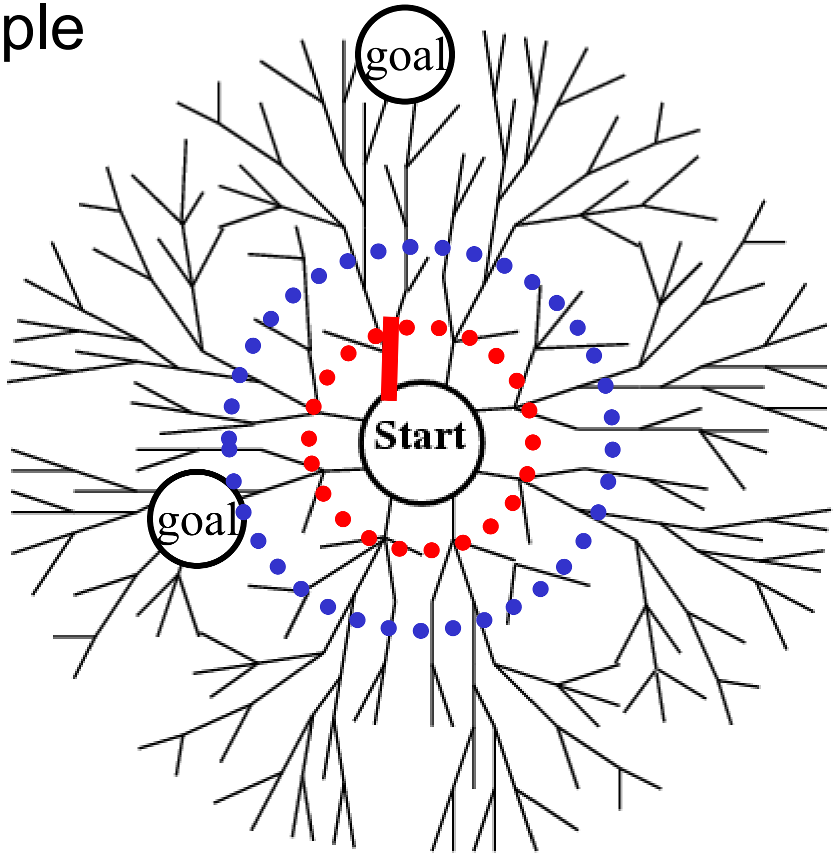
	Complete	optimal	time	space
Breadth-first search	Y	Y, if [1]	$O(b^d)$	$O(b^d)$
Uniform-cost search[2]	Y	Y	$O(b^{C^*/\varepsilon})$	$O(b^{C^*/\varepsilon})$
Depth-first search	N	N	$O(b^m)$	$O(bm)$

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

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…

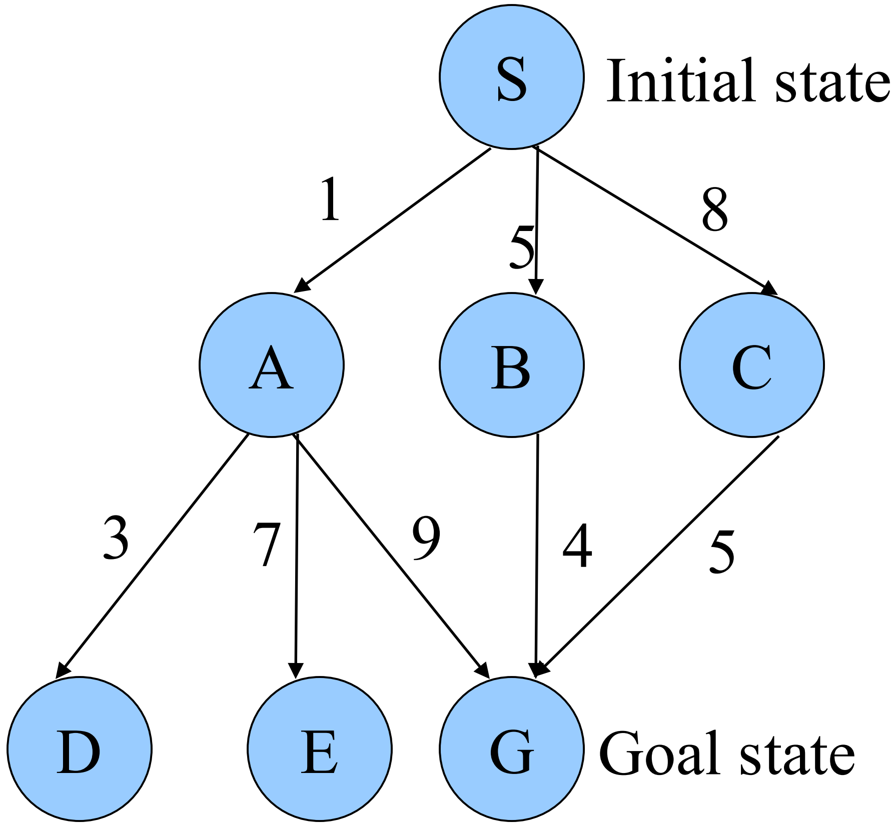
fan within ripple



Iterative deepening

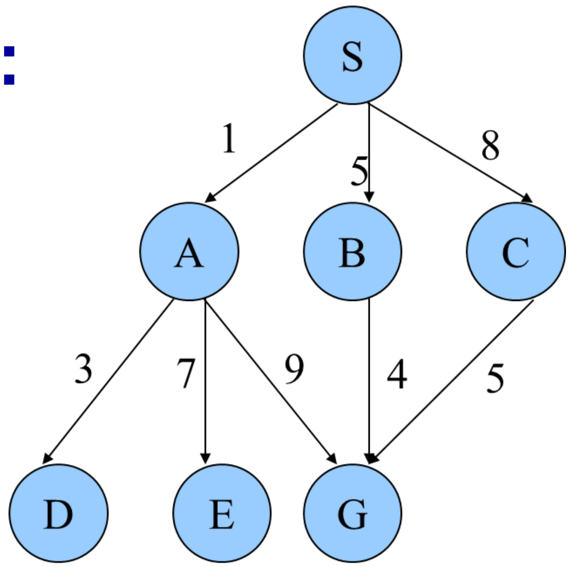
- Search proceeds like BFS, but fringe is like DFS
  - Complete, optimal like BFS
  - Small space complexity like DFS
  - Time complexity like BFS
- Preferred uninformed search method

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

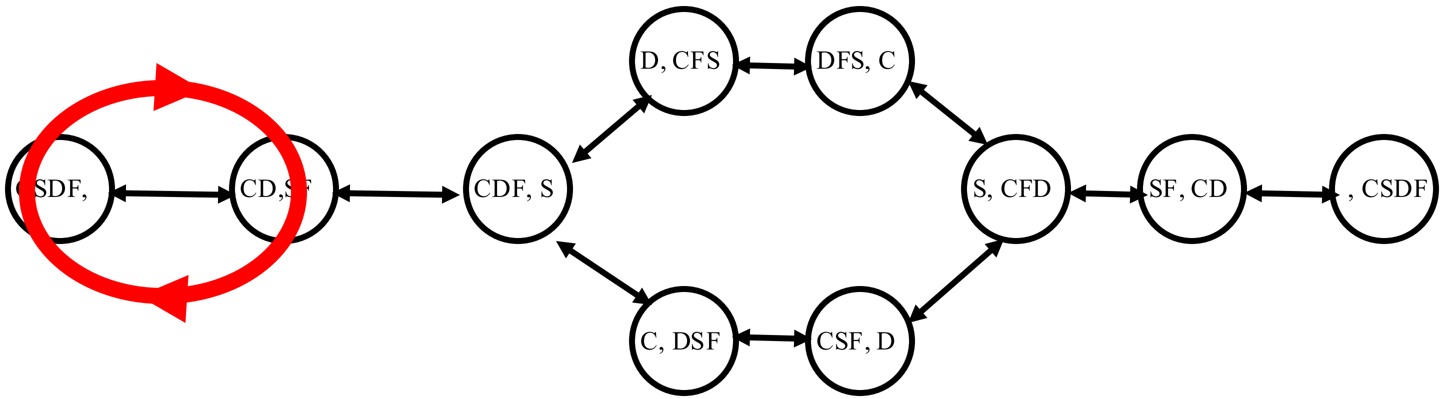
b: branching factor (assume finite) d: goal depth m: graph depth

	Complete	optimal	time	space
Breadth-first search	Y	Y, if [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

- 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.

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