

CS540

Uninformed Search

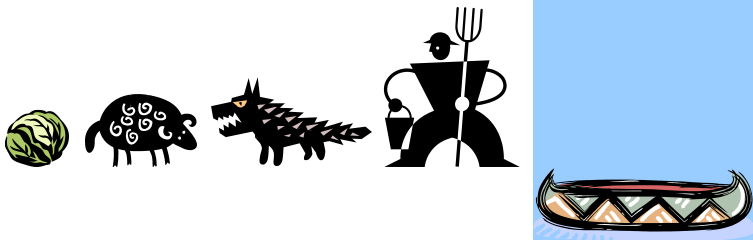
Anthony Gitter

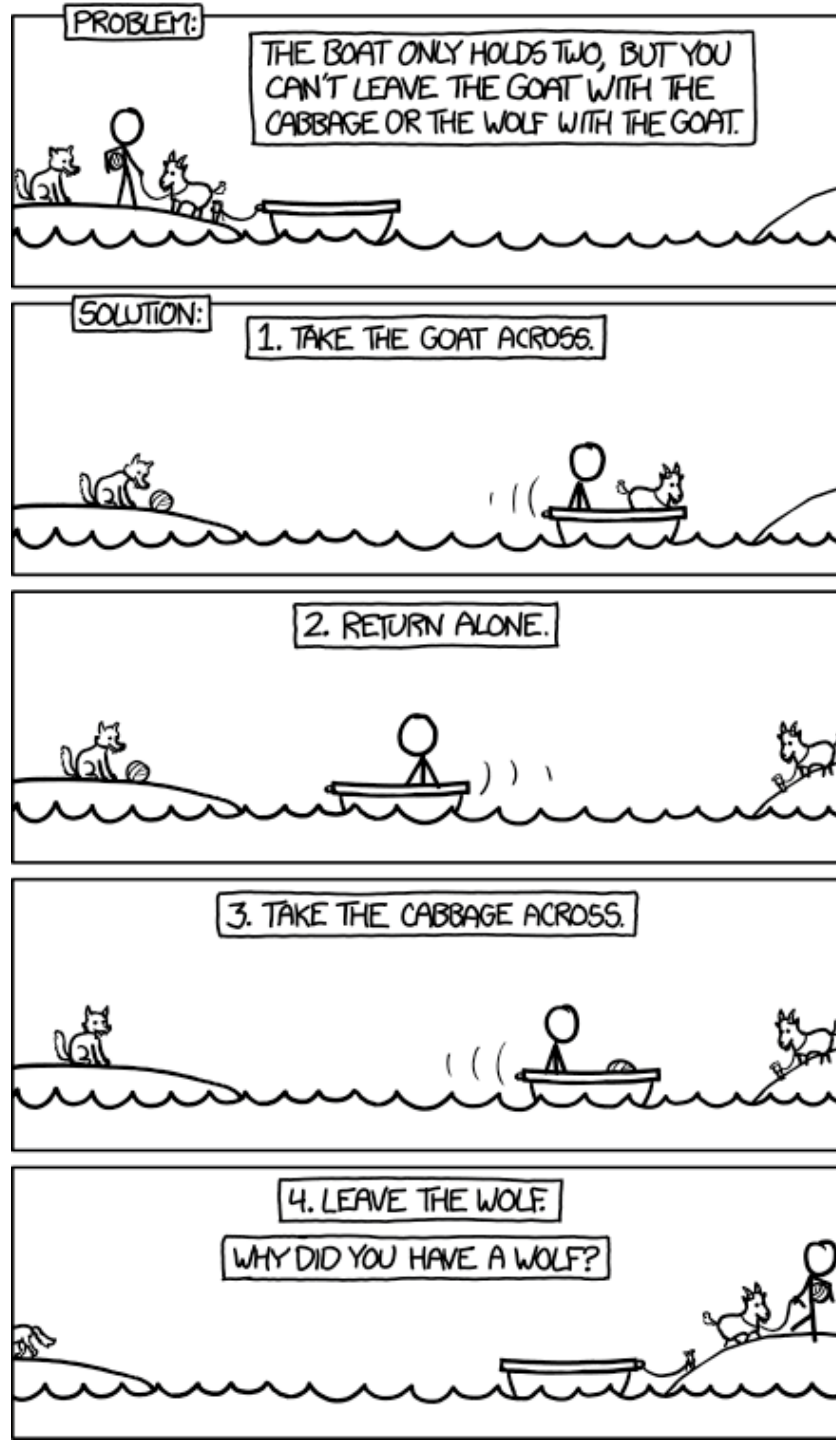
`gitter@biostat.wisc.edu`

University of Wisconsin-Madison

Main messages

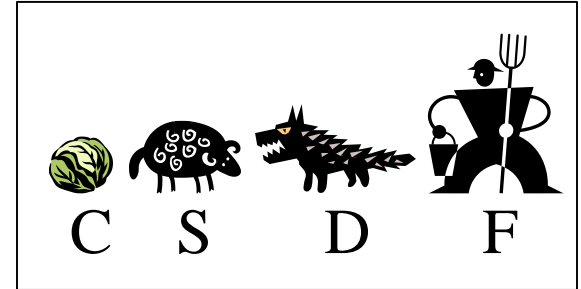
- Many AI problems can be formulated as search.
- Iterative deepening is good when you don't know much.





The search problem

- **State space** S : all valid configurations
- **Initial states (nodes)** $I = \{(CSDF,)\} \subseteq S$
 - Where's the boat?
- **Goal states** $G = \{(\cdot, CSDF)\} \subseteq S$
- **Successor function** $succs(s) \subseteq S$: states reachable in one step (one arc) from state s
 - $succs((CSDF,)) = \{(CD, SF)\}$
 - $succs((CDF, S)) = \{(CD, FS), (D, CFS), (C, DFS)\}$
- **Cost** $(s, s') = 1$ for all arcs. (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

| | | |
|---|---|---|
| 7 | 2 | 4 |
| 5 | | 6 |
| 8 | 3 | 1 |

Start State

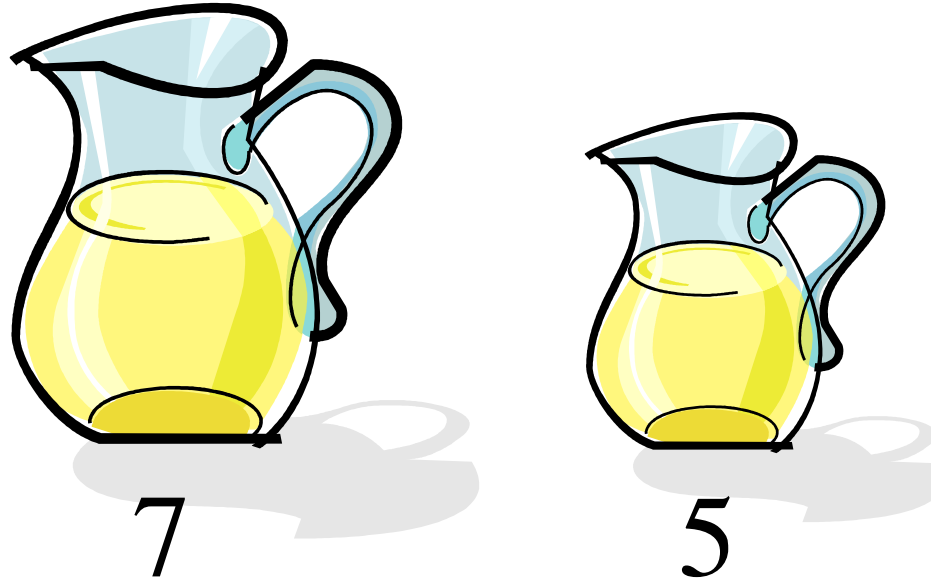
| | | |
|---|---|---|
| | 1 | 2 |
| 3 | 4 | 5 |
| 6 | 7 | 8 |

Goal State

- States = configurations
- Successor function = up to 4 kinds of movement
- Cost = 1 for each move

Search examples

- Water jugs: how to get 1?



- Goal? (How many goal states?)
- Successor function: fill up (from tap or other jug), empty (to ground or other jug)

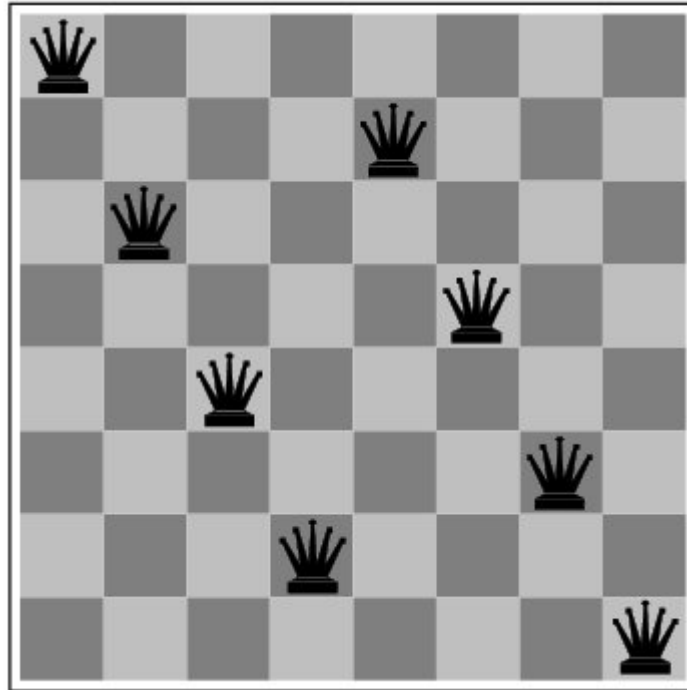
Search examples

- Route finding (State? Successors? Cost weighted)

The screenshot shows the Google Maps interface with a route calculated between two addresses in Madison, WI. The browser title bar indicates the route is from 1210 W Dayton St to State St. The search bar shows the start address as 1210 W Dayton St, Madison, WI 53706 and the end address as State St, Madison, WI 53703. The map displays a blue route starting at the start address, heading east on W Dayton St, then turning left at N Frances St, right at W Gilman St, right at N Henry St, and finally right at W Gorham St. The right sidebar provides details: Start address: 1210 W Dayton St, Madison, WI 53706; End address: State St, Madison, WI 53703; Distance: 1.2 mi (about 2 mins). It also lists 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. A disclaimer states: "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 is attributed to ©2005 NAVTEQ™, Tele Atlas.

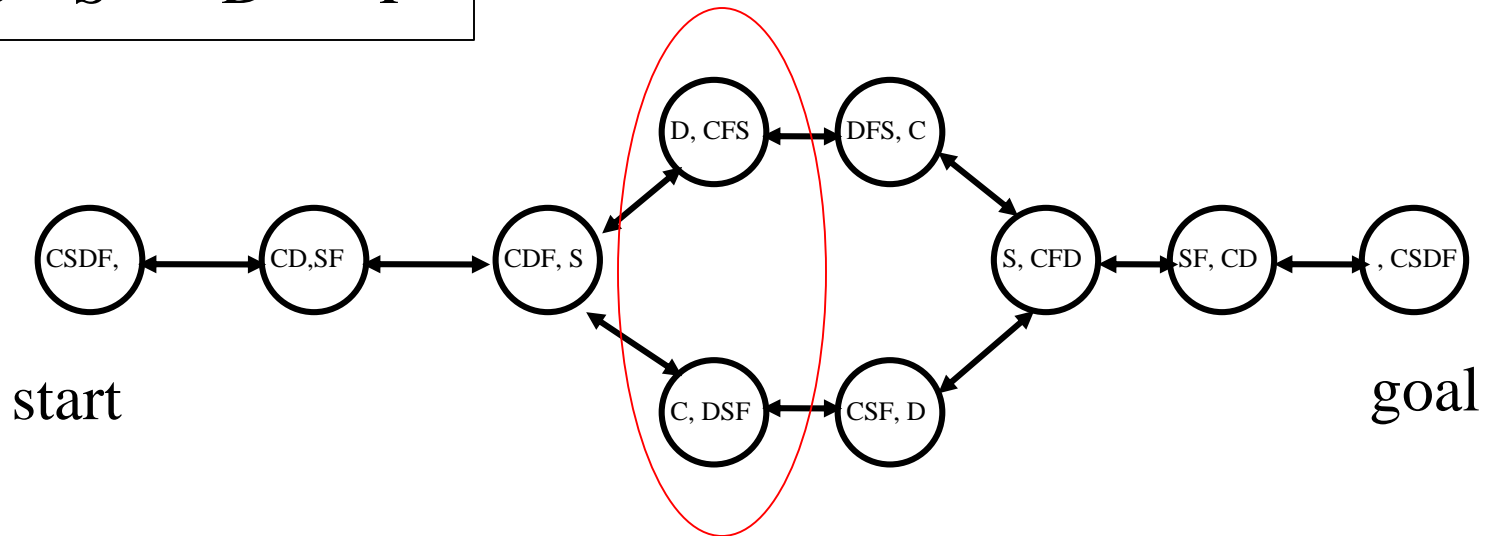
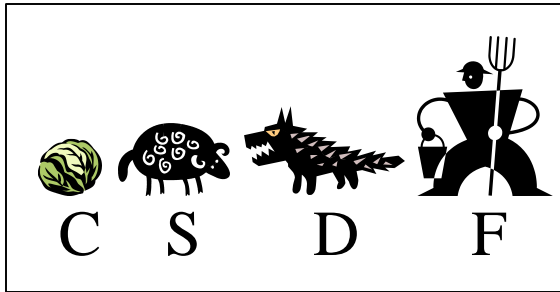
8-queens

- State: complete configuration vs. column-by-column



- Column-by-column produces **tree** instead of graph

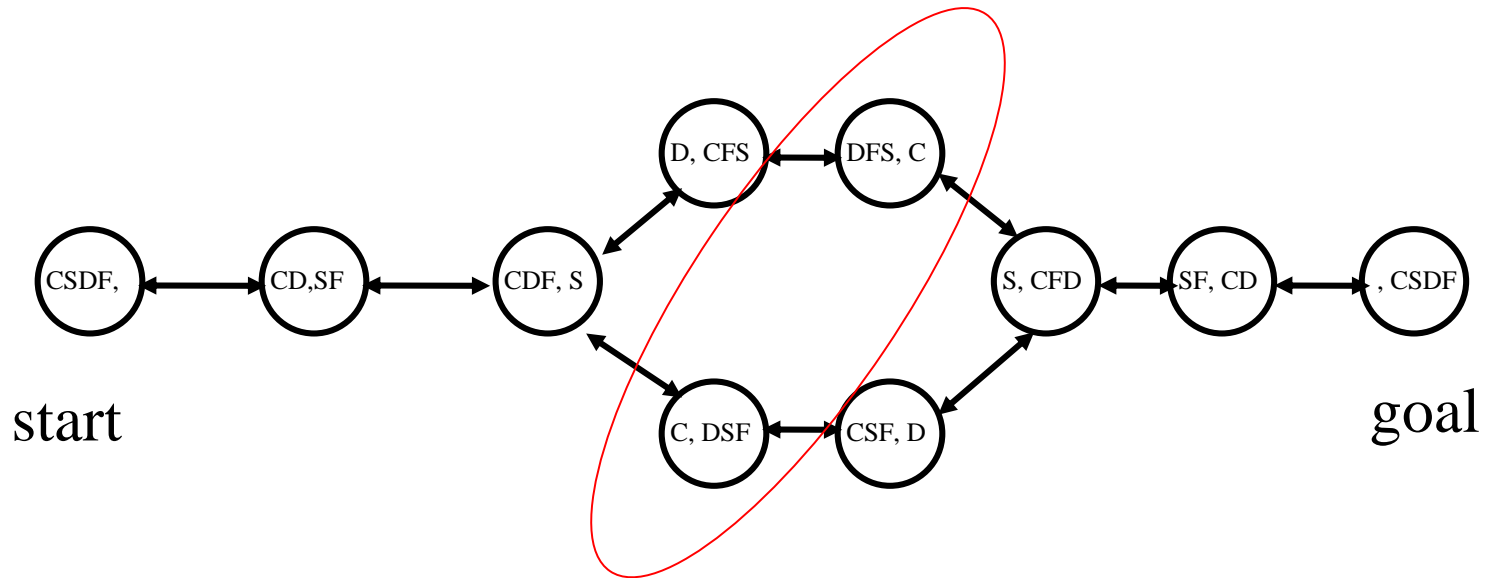
A directed graph in state space



- 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

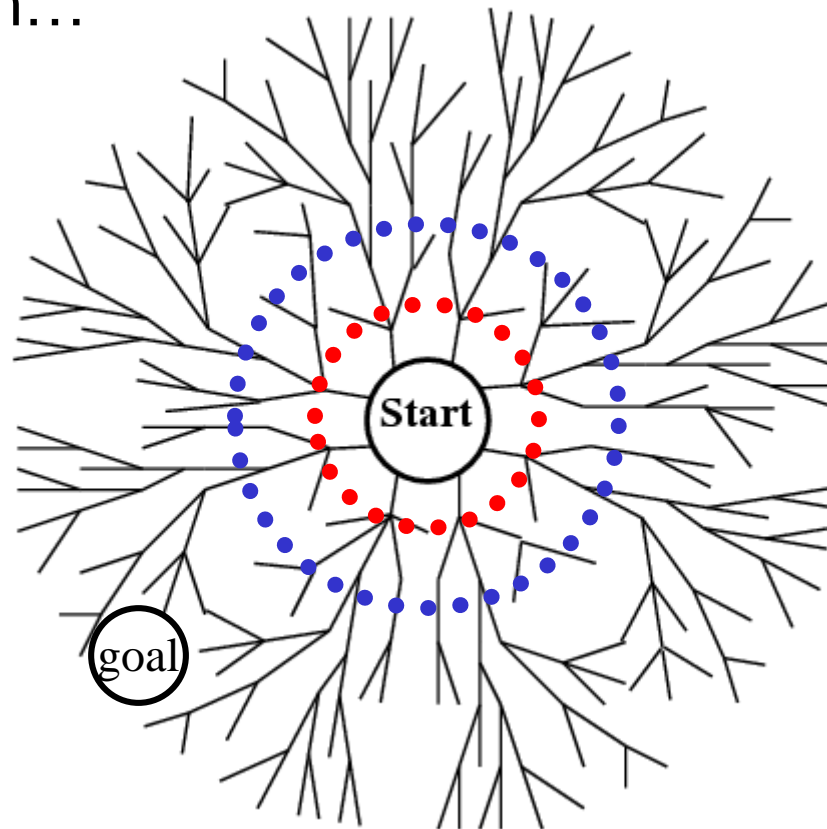
- **Uninformed** means we only know:
 - The goal test
 - The *succs()* function
- But **not** which non-goal states are better: that would be informed search (next topic).
- For now, we also assume *succs()* graph is **a tree**.
 - Won't encounter repeated states.
 - We will relax it later.
- Search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

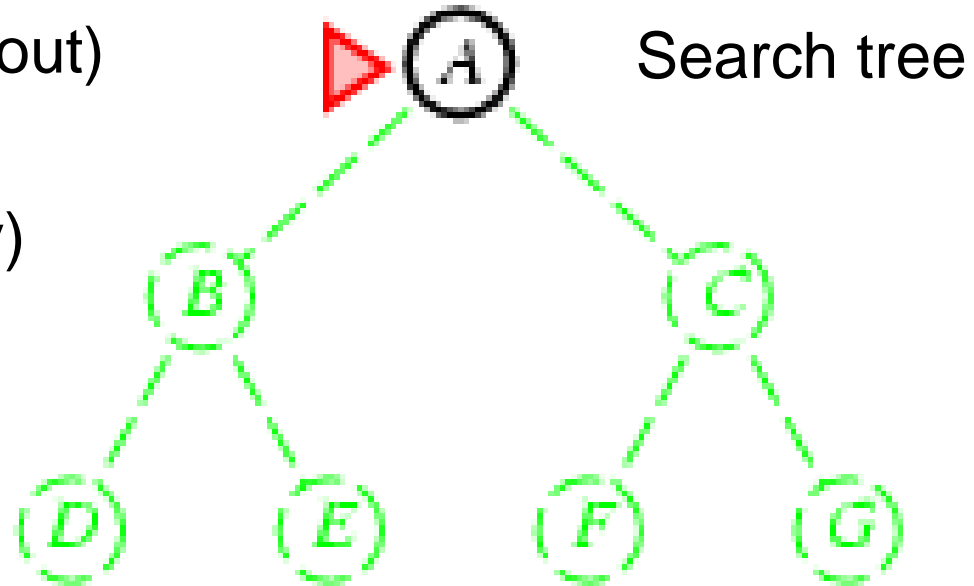
ripple



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



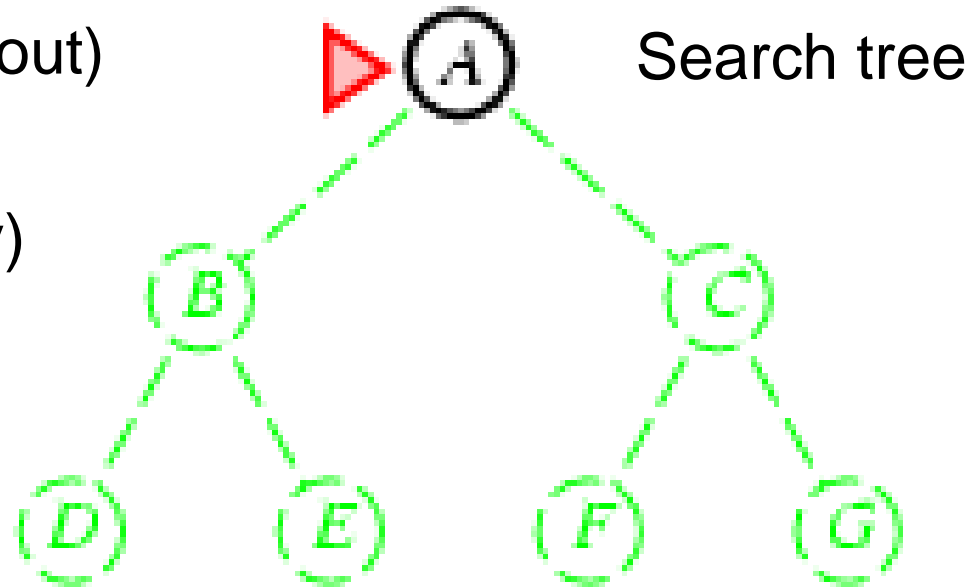
Initial state: **A**

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] →

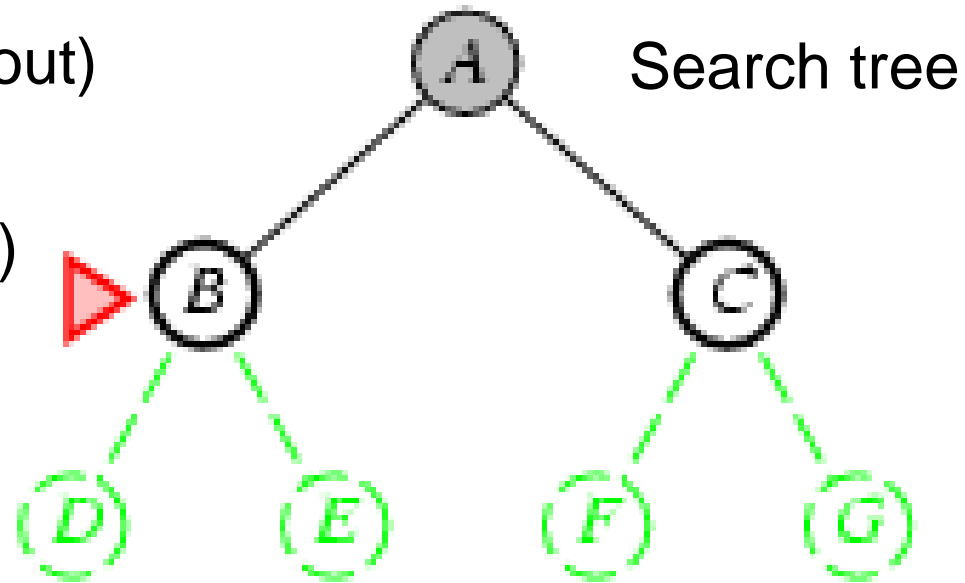
Initial state: **A**

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

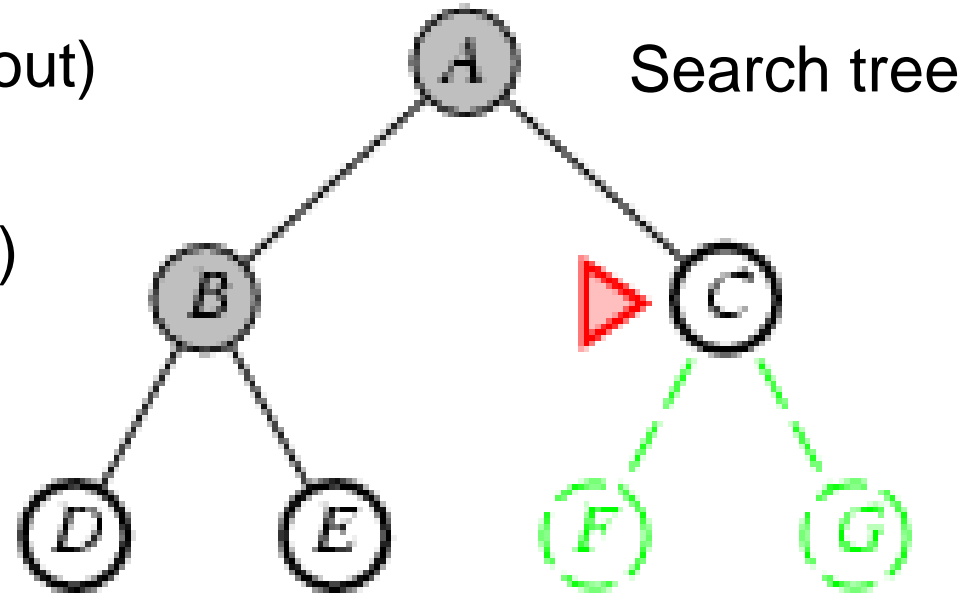
Initial state: **A**

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**

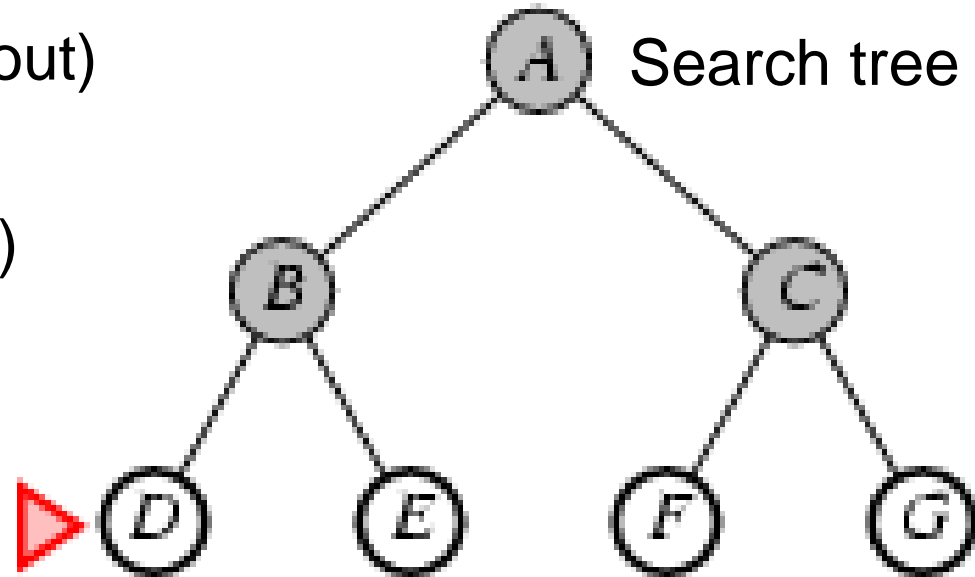
Goal state: **G**

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

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



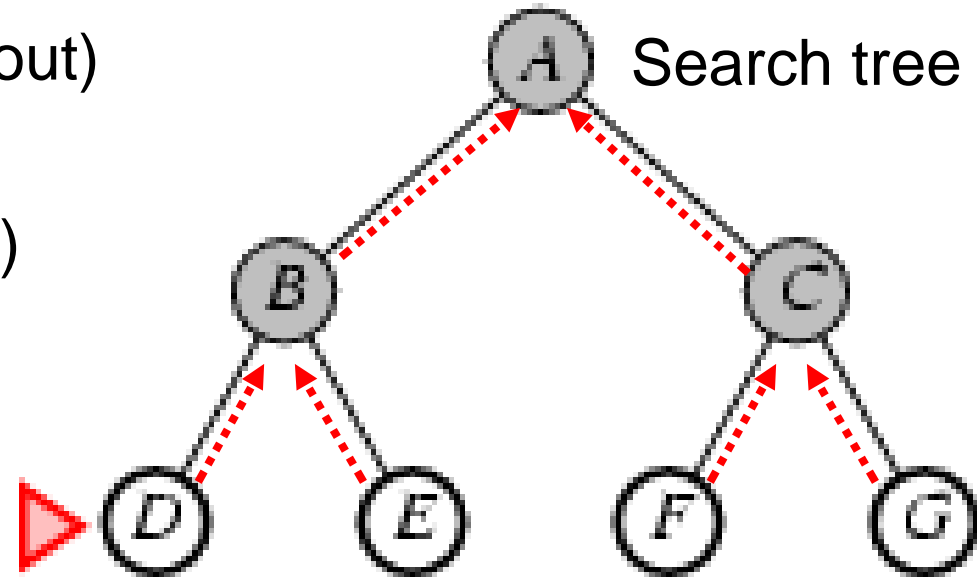
queue (**fringe, OPEN**)
→[GFED] → C

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

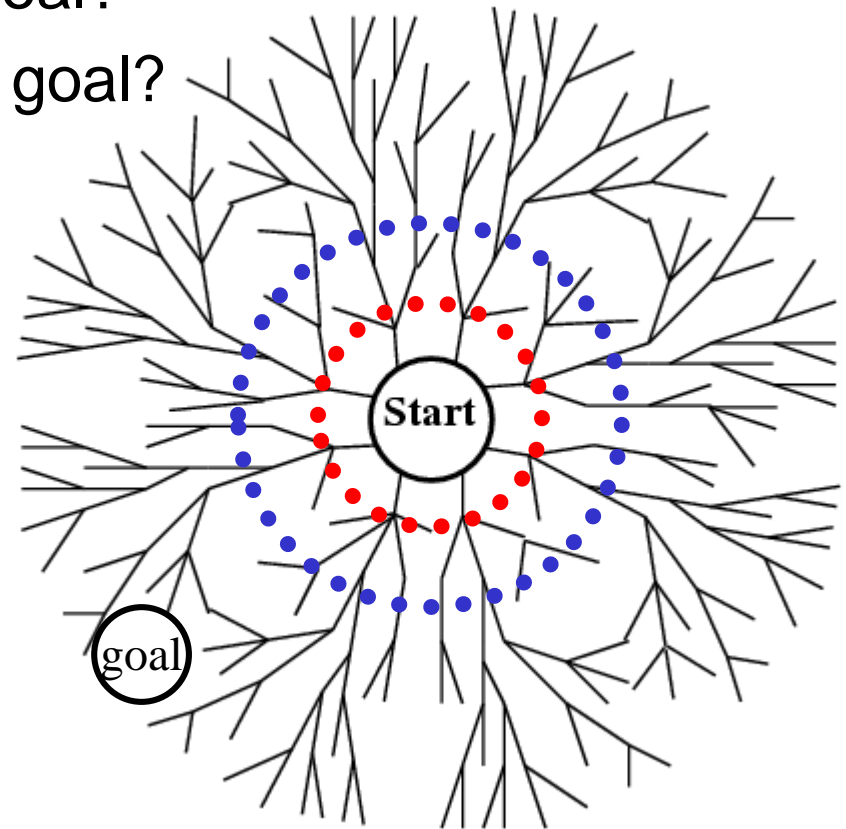
... until much later we pop G.

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



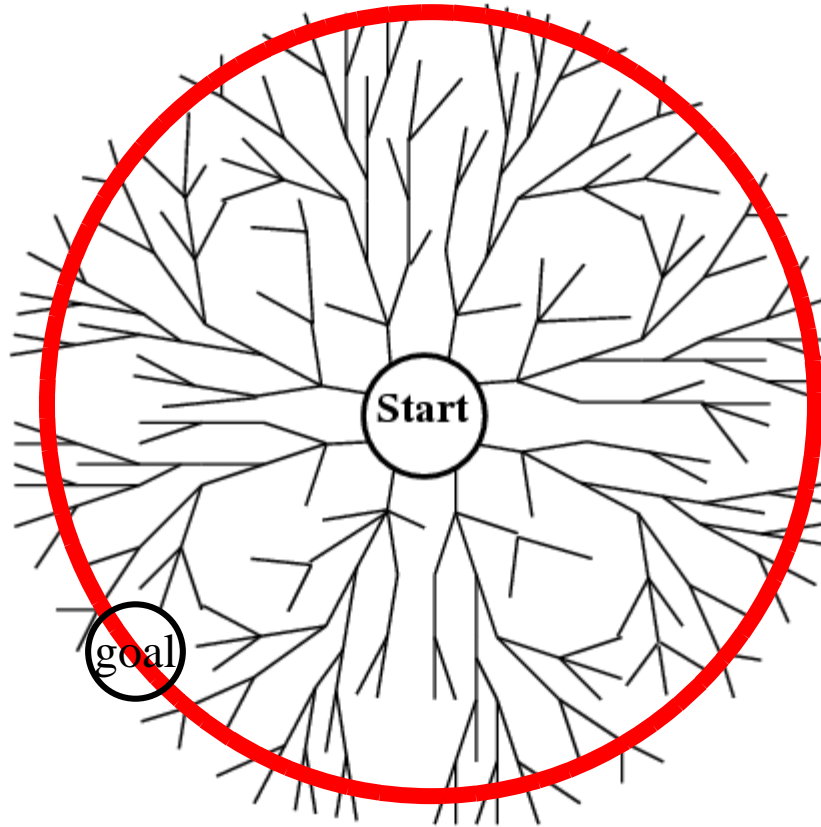
Performance of BFS

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 .
 - $b + b^2 + \dots + b^d \sim O(b^d)$
- **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

Performance of BFS

Four measures of search algorithms:

- **Completeness** (not finding all goals): 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 .
 - $b + b^2 + \dots + b^d \sim O(b^d)$
- **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**

Uniform-cost search

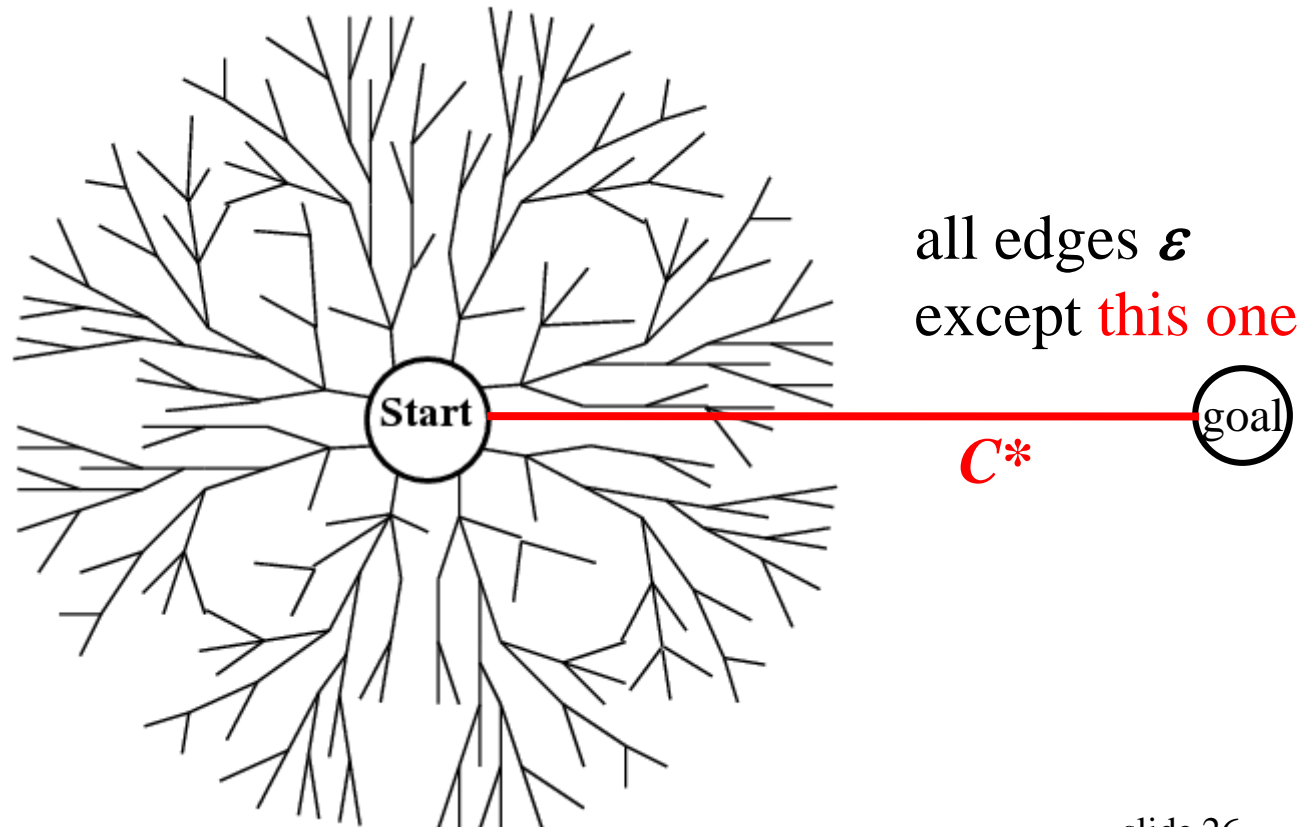
- Find the least-cost goal
- Each node has a path cost from start (= sum of edge costs along the path).
- Expand the least cost node first.
- Use a **priority queue** instead of a normal queue
 - Always take out the least cost item
 - Remember *heap*? time $O(\log(\#items\ in\ heap))$

That's it*

* Complications on graphs (instead of trees). Later.

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
 - $O(b^{C^*/\epsilon})$, possibly $C^*/\epsilon \gg d$



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 | $O(b^d)$ | $O(b^d)$ |
| Uniform-cost search ² | 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.

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
```

Recall the bad space complexity of BFS

Four measures of search algorithms:




- **Completeness** (not finding all goals): find a goal.
- **Optimality**: yes if edges cost 1 (more generally positive non-decreasing with 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, Figure 3.11)
 - Back points for all generated nodes $O(b^d)$
 - The queue (smaller, but still $O(b^d)$)

**Solution:
Uniform-cost
search**

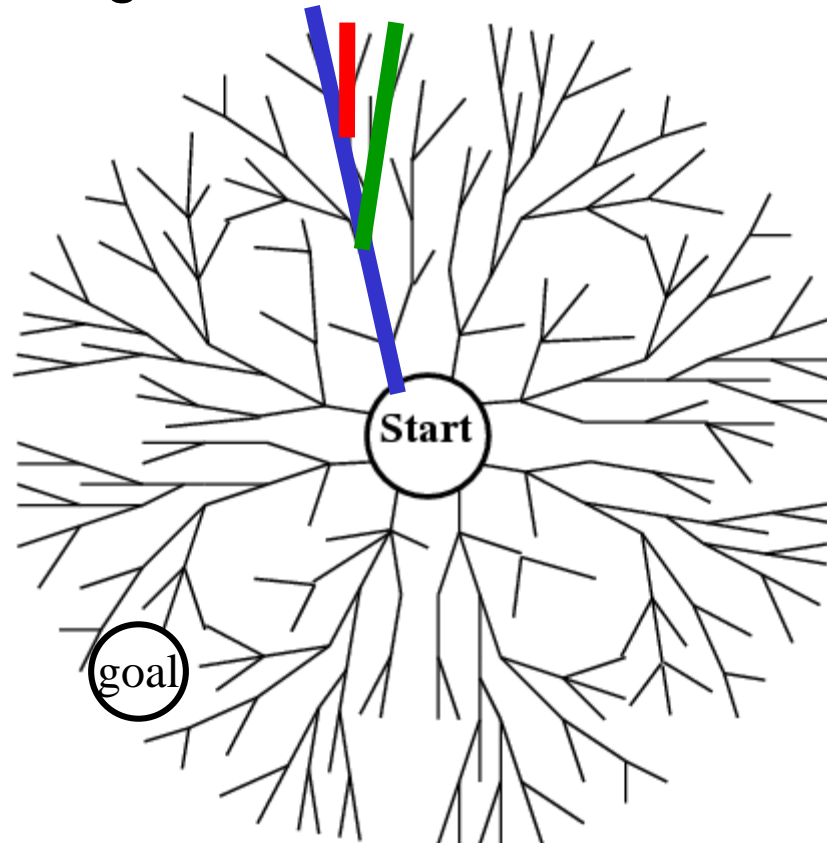
**Solution:
Depth-first
search**

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... 

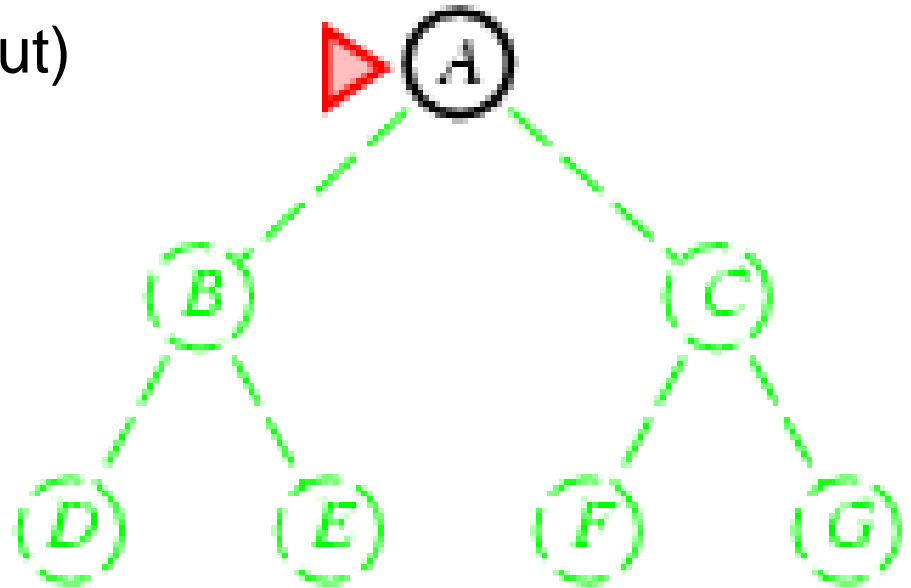
fan



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

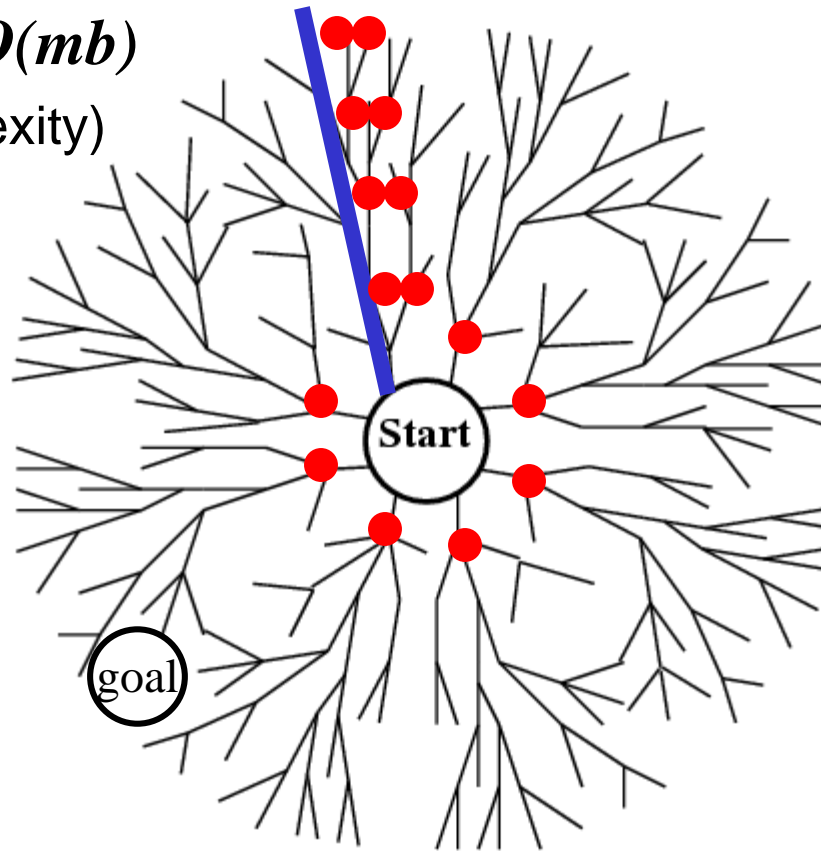


stack (**fringe**)

[] ⇔

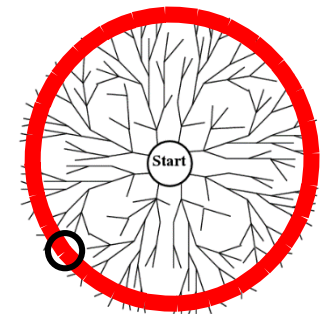
What's in the fringe for DFS?

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



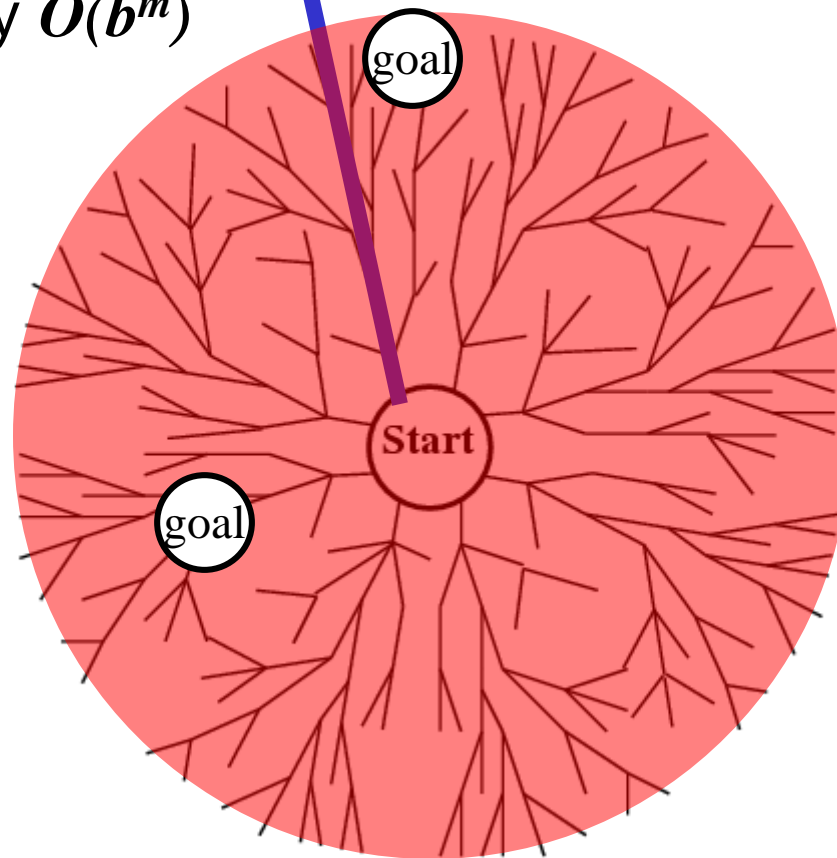
- “backtracking search” even less space
 - generate siblings (if applicable)

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

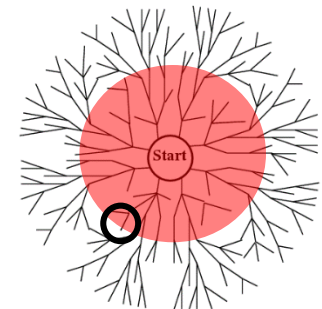


What's wrong with DFS?

- 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

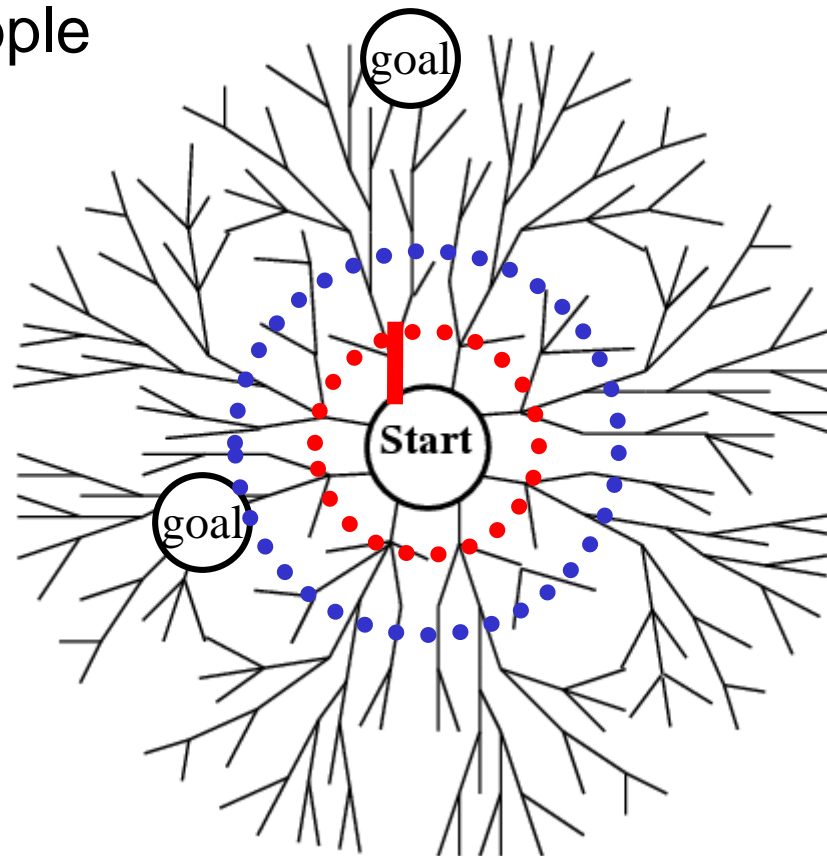
| | Complete | optimal | time | space |
|----------------------------------|----------|--------------------|-----------------------|-----------------------|
| Breadth-first search | Y | Y, if ¹ | $O(b^d)$ | $O(b^d)$ |
| Uniform-cost search ² | Y | Y | $O(b^{C^*/\epsilon})$ | $O(b^{C^*/\epsilon})$ |
|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

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
- A huge waste?
 - Each deepening repeats DFS from the beginning
 - No! $db + (d-1)b^2 + (d-2)b^3 + \dots + b^d \sim O(b^d)$
 - 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 | $O(b^{C^*/\epsilon})$ | $O(b^{C^*/\epsilon})$ |
| 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 $\geq \epsilon > 0$. C^* is the best goal path cost.

Performance of search algorithms on trees

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

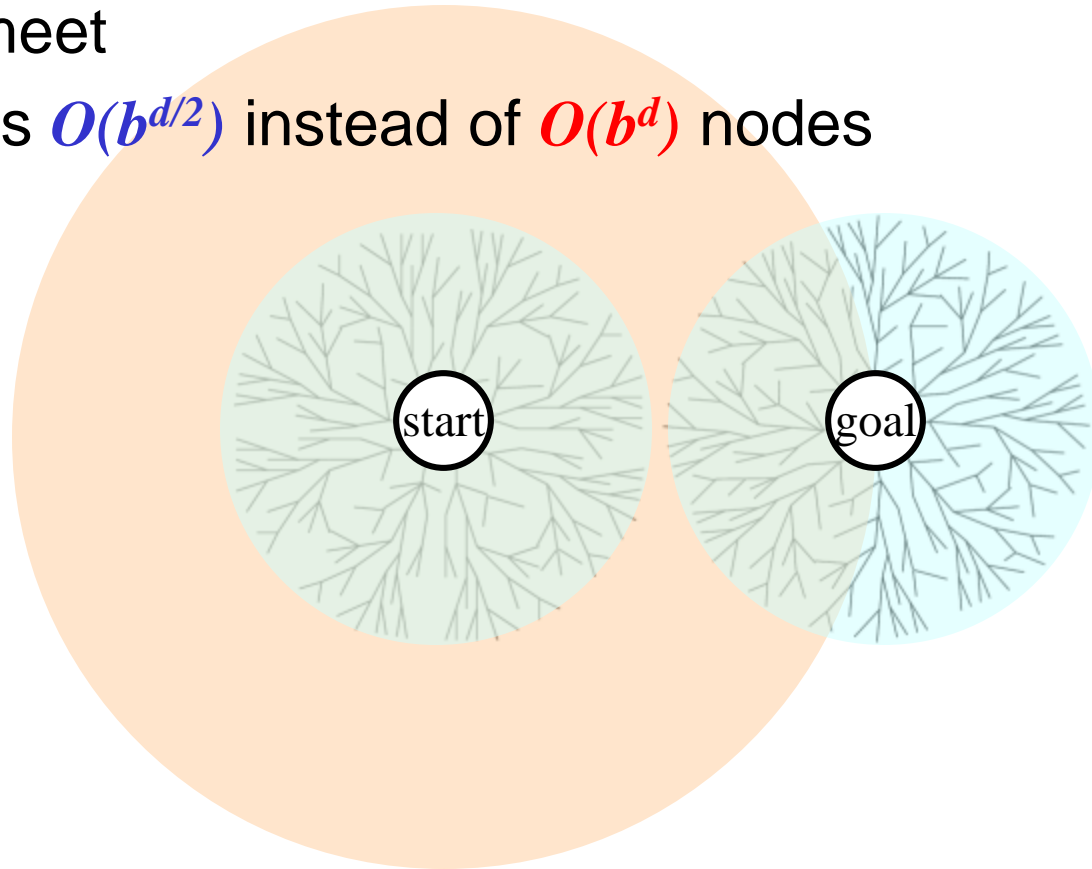
| | Complete | optimal | time | space |
|---------------------|----------|---------|-----------------------|-----------------------|
| Breadth-first | | | $O(b^d)$ | $O(b^d)$ |
| | | | $O(b^{C^*/\epsilon})$ | $O(b^{C^*/\epsilon})$ |
| | | | $O(b^m)$ | $O(bm)$ |
| Iterative deepening | | | $O(b^d)$ | $O(bd)$ |
| | | | | |

How to reduce the number of states we have to generate?

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

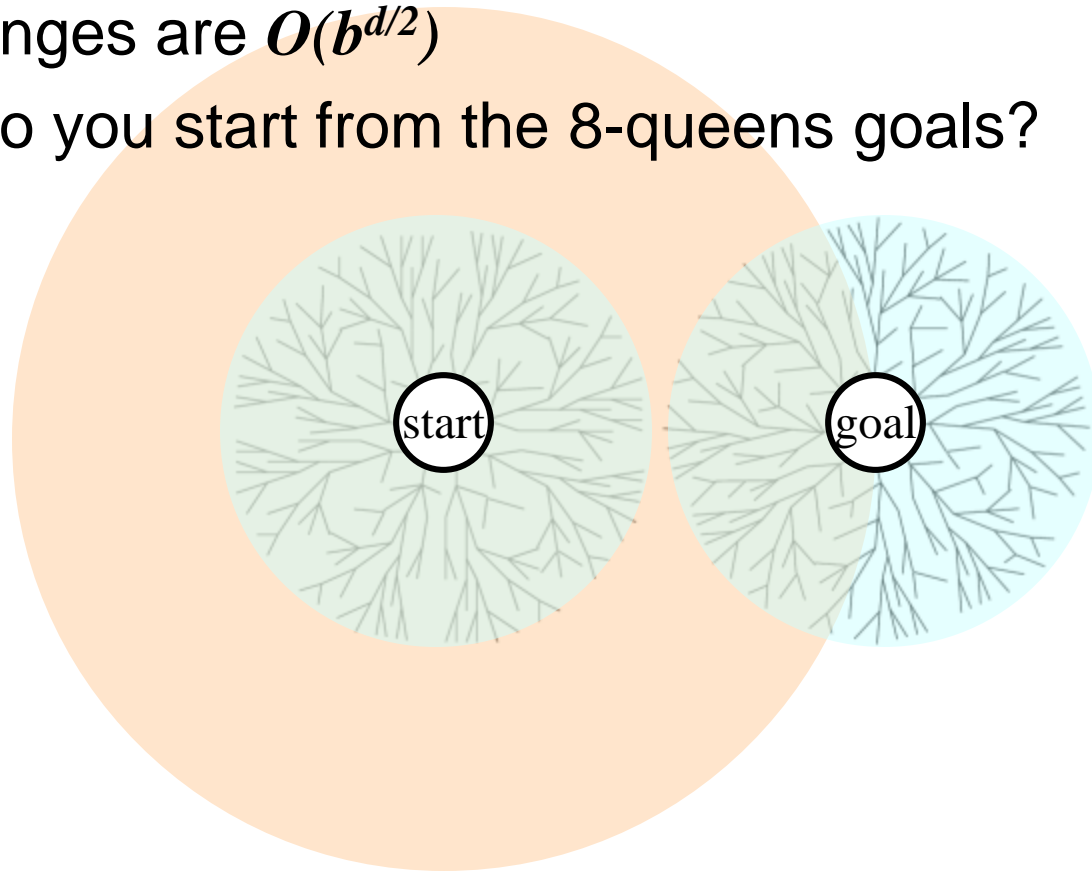
Bidirectional search

- Breadth-first search from both start and goal
- Fringes meet
- Generates $O(b^{d/2})$ instead of $O(b^d)$ nodes



Bidirectional search

- But
 - The fringes are $O(b^{d/2})$
 - How do you start from the 8-queens goals?



Performance of search algorithms on trees

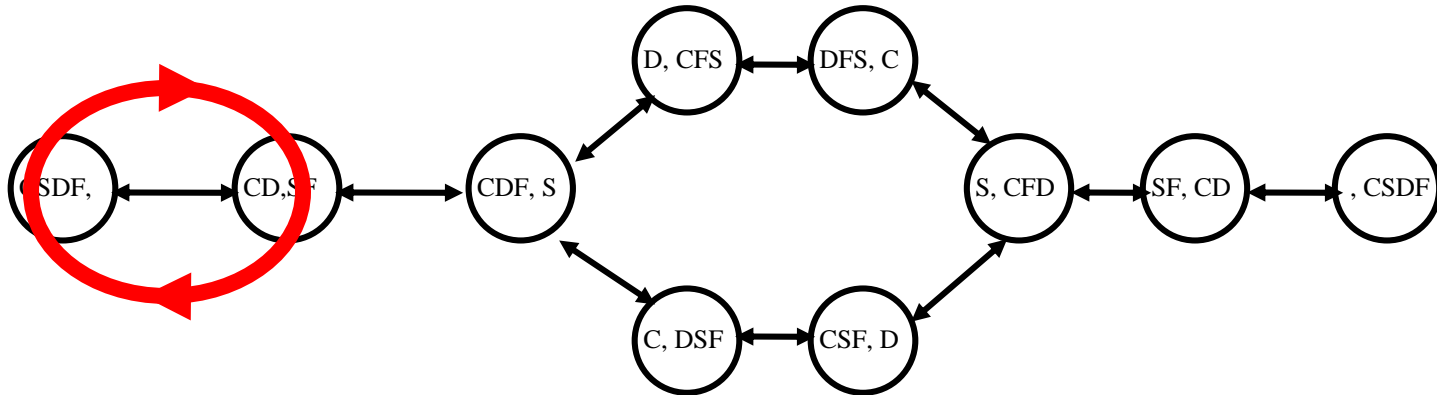
b: branching factor (assume finite) d: goal depth m: graph depth

| | Complete | optimal | time | space |
|-----------------------------------|----------|--------------------|-----------------------|-----------------------|
| Breadth-first search | Y | Y, if ¹ | $O(b^d)$ | $O(b^d)$ |
| Uniform-cost search ² | Y | Y | $O(b^{C^*/\epsilon})$ | $O(b^{C^*/\epsilon})$ |
| Depth-first search | N | N | $O(b^m)$ | $O(bm)$ |
| Iterative deepening | Y | Y, if ¹ | $O(b^d)$ | $O(bd)$ |
| Bidirectional search ³ | Y | Y, if ¹ | $O(b^{d/2})$ | $O(b^{d/2})$ |

1. edge cost constant, or positive non-decreasing in depth
2. edge costs $\geq \epsilon > 0$. C^* is the best goal path cost.
3. both directions BFS; not always feasible.

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.

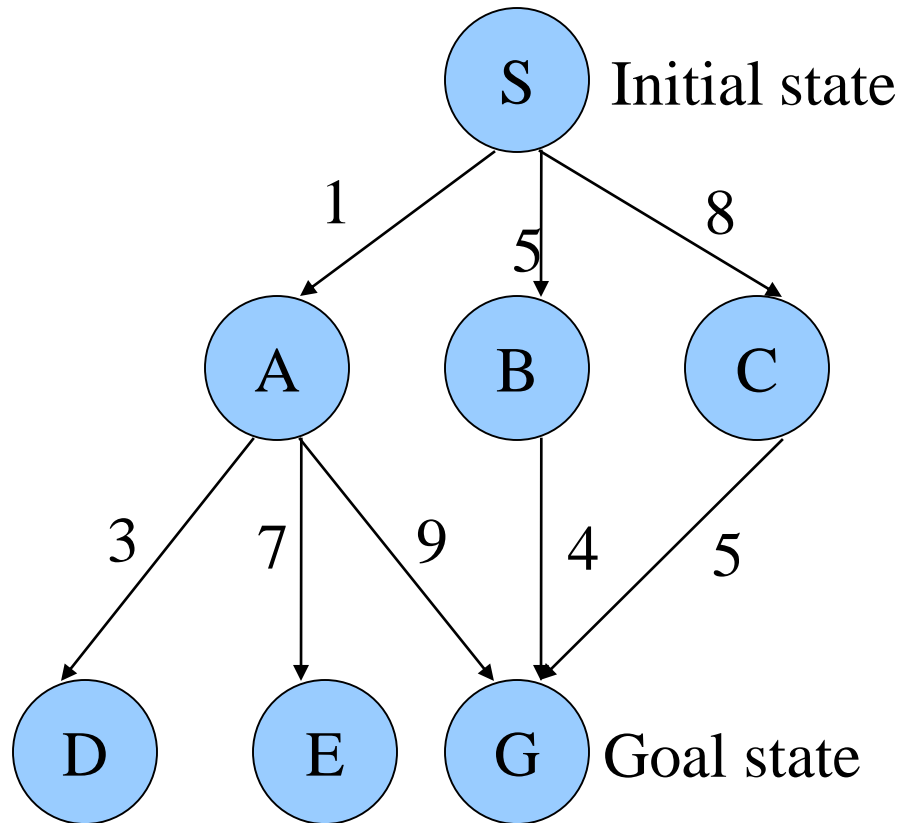
If state space graph is not a tree

- BFS:
 - Still $O(b^d)$ space complexity, not worse
- DFS:
 - Known as Memorizing DFS (**MEMDFS**)
 - Space and time now $O(\min(N, b^m))$ – much worse!
 - N: number of states in problem
 - m: length of longest cycle-free path from start to anywhere
 - Alternative: Path Check DFS (**PCDFS**): remember only expanded states on current path (from start to the current node)
 - Space $O(m)$
 - Time $O(b^m)$

Path Checking DFS

1. Maintain a “prefix” path from root to current node, initially empty.
2. Pop a state **s**. If **s** in prefix, skip to next pop
3. Goal-checking **s**.
4. **s** comes with a backpointer to its parent **p**. The prefix should contain **p** somewhere as in **initial, ..., p, ...**
5. Remove everything after **p** and put **s** there, so prefix is now **initial, ..., p, s**.
6. When you generate a successor **t** of **s**, check if **t** is in **prefix or stack**. If no, push **t** to the stack; if yes, do not push it.

Example



(All edges are directed, pointing downwards)

Nodes expanded by:

- Depth-First Search: S A D E G

Solution found: S A G

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

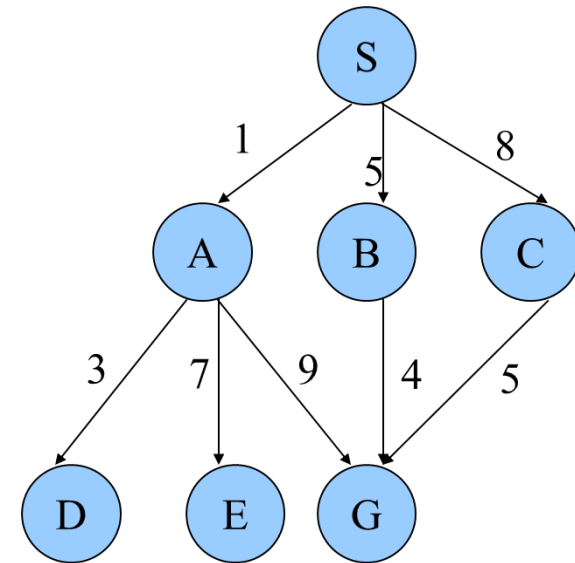
Solution found: S A G

- 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.)

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

Solution found: S A G



Depth-First Search

| expanded node | nodes list |
|------------------|---------------|
| ----- | ----- |
| | { S } |
| S | { A B C } |
| A | { D E G B C } |
| D | { E G B C } |
| E | { G B C } |
| G | { B C } |

Solution path found is S A G <-- this G has cost 10
Number of nodes expanded (including goal node) = 5

Breadth-First Search

| expanded node | nodes list |
|------------------|------------------|
| ----- | ----- |
| | { S } |
| S | { A B C } |
| A | { B C D E G } |
| B | { C D E G G' } |
| C | { D E G G' G'' } |
| D | { E G G' G'' } |
| E | { G G' G'' } |
| G | { G' G'' } |

Solution path found is S A G <-- this G also has cost 10
Number of nodes expanded (including goal node) = 7

Uniform-Cost Search

| expanded node | nodes list |
|------------------|---|
| ----- | ----- |
| | { S } |
| S | { A(1) B(5) C(8) } |
| A | { D(4) B(5) C(8) E(8) G(10) } (note, we don't return G) |
| D | { B(5) C(8) E(8) G(10) } |
| B | { C(8) E(8) G(9) G(10) } |
| C | { E(8) G(9) G(10) G(13) } |
| E | { G(9) G(10) G(13) } |
| G | { } |

Solution path found is S B G <-- this G has cost 9, not 10
Number of nodes expanded (including goal node) = 7

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** ★
 - Bidirectional search
- Can you unify them (except bidirectional) using the same algorithm, with different priority functions?
- Performance measures
 - Completeness, optimality, time complexity, space complexity

