# CS-540: Intro to Artificial Intelligence

SECTION 1

LECTURER: ERIN WINTER

Uninformed Search...Continued

# Administrative News

Please welcome your second half TA, **Zhenyu Zhang.** His office hours will be posted on the course Canvas page later today.

Admissible?

Optimal?

These terms are **identical** and describe whether or not an algorithm finds an **optimal** solution.

Complete?

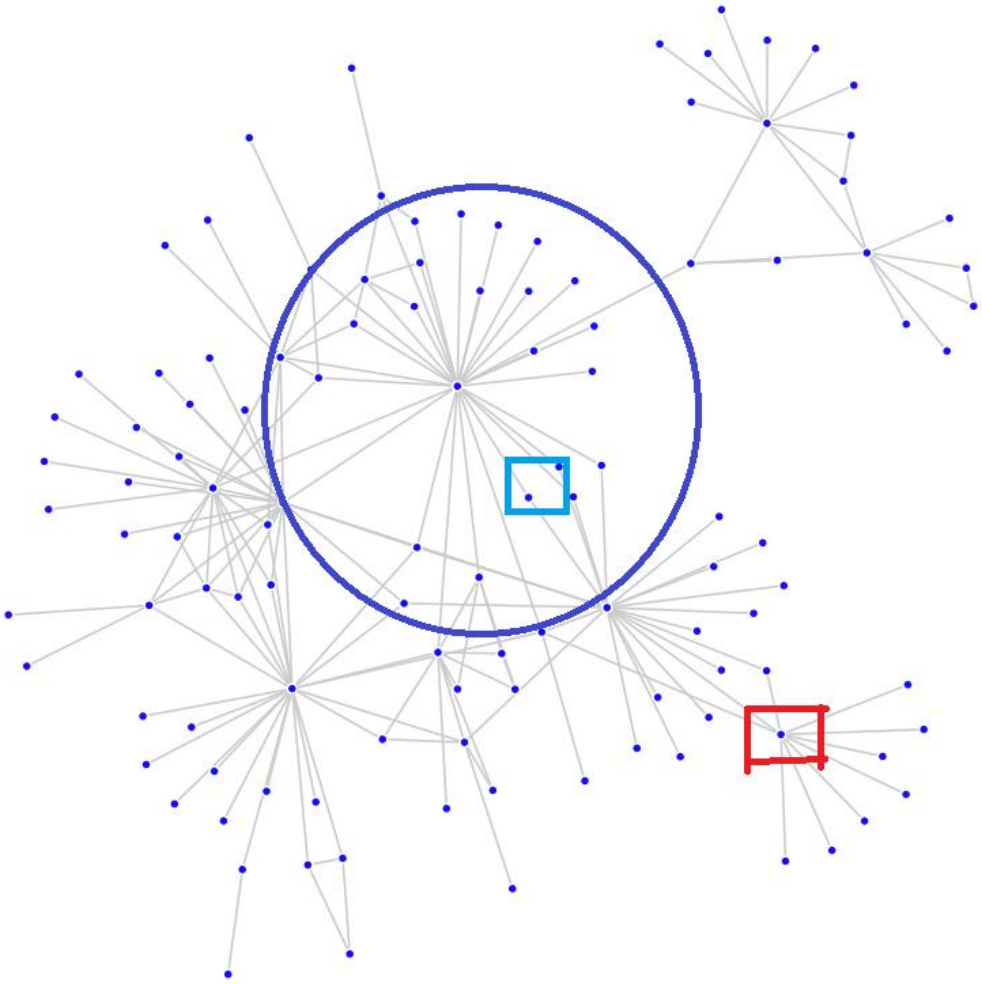
Describes whether or not an algorithm is **guaranteed** to find a solution **if one exists** .

Time Complexity

Space Complexity

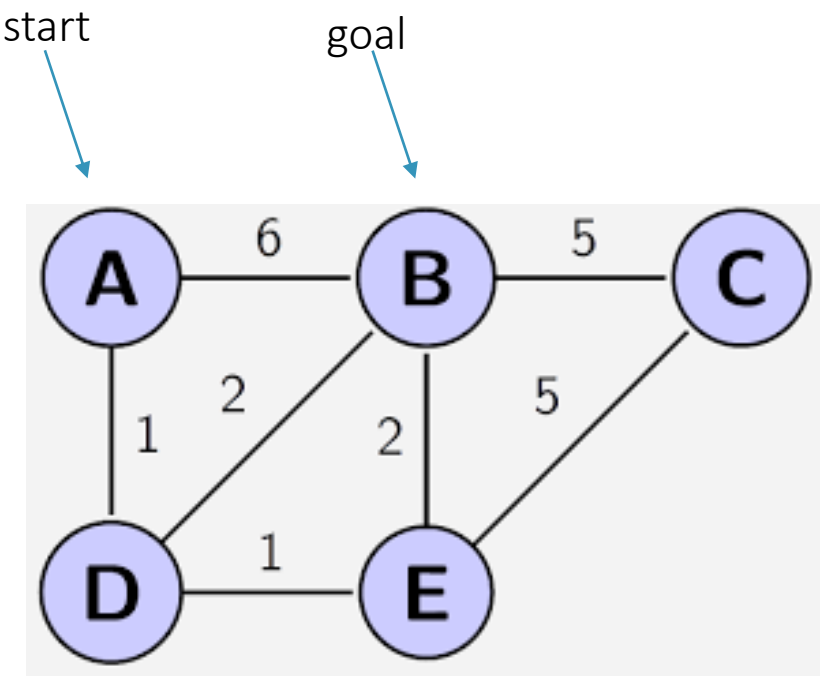
An algorithm can be admissible but not complete

**Ex:** Imagine a naïve implementation of breadth-first search that only searches to depth 3.



# An algorithm can be complete but not admissible

**Ex:** Imagine an algorithm that computes minimum cost using **path length** rather than **path cost** on a graph with positive, non-zero edges.




Pretend B is the Goal State

PATH LENGTH MINIZING ALG RETURNS

A->B = 6

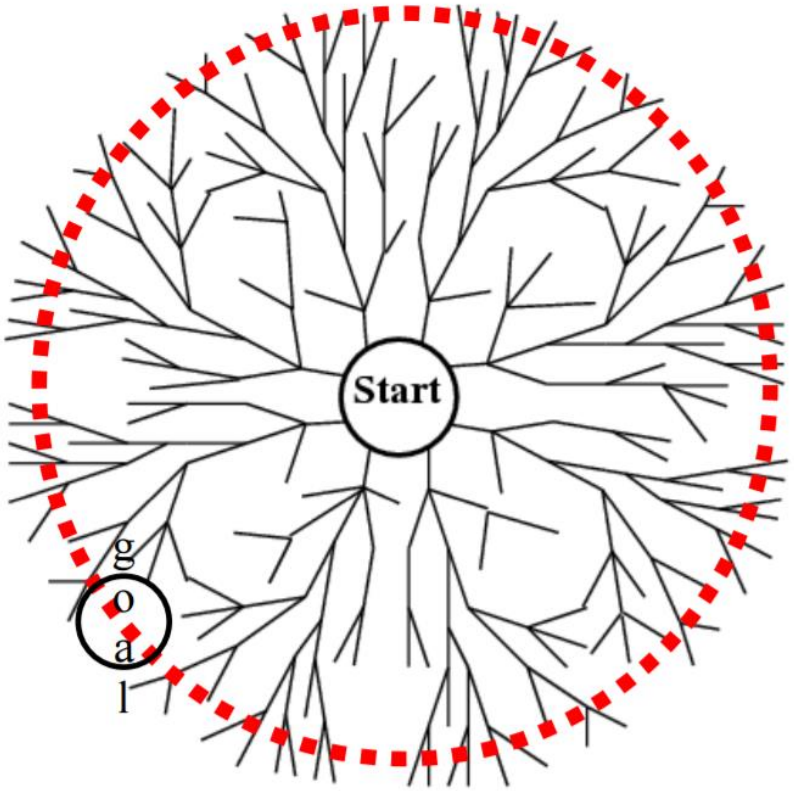
DIJKSTRA'S ALG RETURNS

A->D->B = 3

Hence, not optimal/admissible.

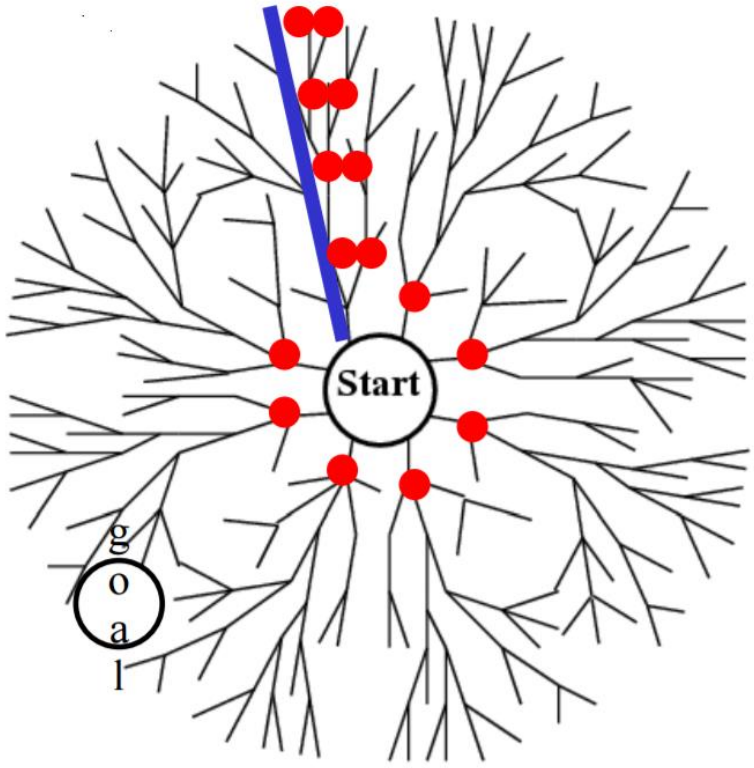
# Key Flaws of Depth-First and Breadth-First Search

- In BFS, the maximum size of the frontier reaches $O(b^d)$ where b is the branching factor and d is the solution depth*
- The explored set also reaches size $b^1 + b^2 \ldots + b^d$ $O(b^d)$
- This gives BFS a space complexity of $O(b^d)$
- Because BFS searches at most $O(b^d)$ nodes before finding a solution, its time complexity is $O(b^d)$




The graph and hence the frontier has b nodes if solution at depth 1, $b^2$ if at depth 2…$b^d$ at depth d.

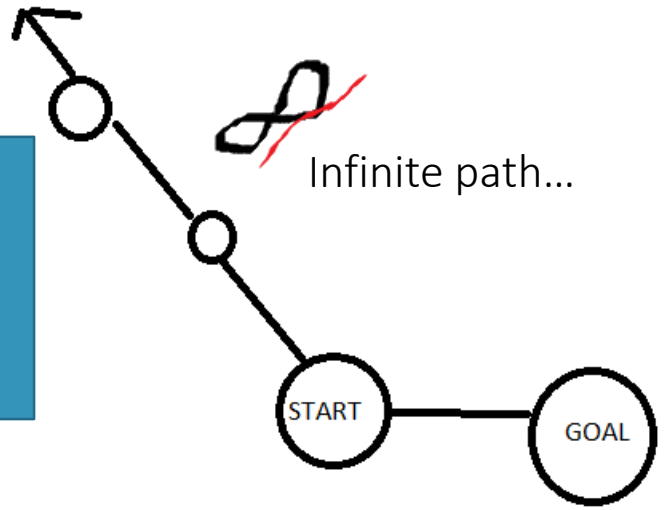
- In DFS, the maximum size of the frontier reaches **O(bm)** where b is the branching factor and m is the **max depth** of the graph*
- This gives DFS a space complexity of **O(bm)**
- Because DFS may search almost the entire graph, before finding a solution, its time complexity is **$O(b^m)$**




Longest path of length m * branching factor of b at start = O(bm) for the frontier.

| Name | Space Complexity | Time Complexity | Admissible | Complete |
|---|---|---|---|---|
| Breadth-First Search | $O(b^d)$ | $O(b^d)$ | True* for positive, equal-edge cost graphs | True |
| Depth-First Search | O(bm) | $O(b^m)$ | False | False |

Why isn't DFS admissible or complete? Here's where DFS goes horribly wrong.

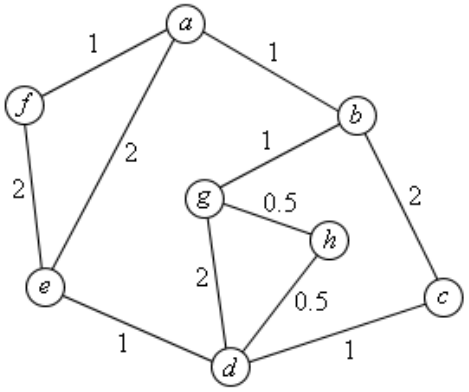


# Evaluating Uniform Cost Search aka Dijkstra's

- Use a "Priority Queue" to order nodes on the *Frontier* list, sorted by path cost
- Let $g(n)$ = cost of path from start node $s$ to current node $n$
- Sort nodes by increasing value of $g$

# Evaluating UCS/Dijkstra's

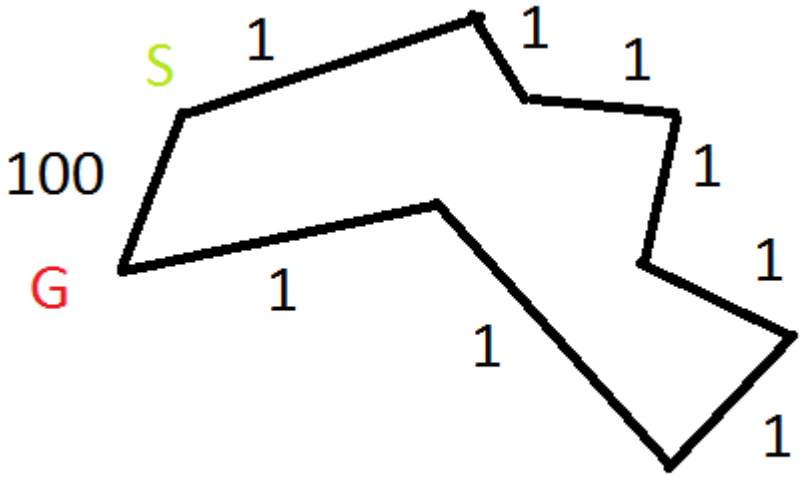
- Dijkstra's picks paths based on path weight not on length, searches increasing cost paths
- When the algorithm terminates **is based on the length of the optimal path from the start state to the goal state**
- How long does it take to search every node until the goal state must be expanded from the optimal path?



# Dijkstra's Runtime

- Let C* be the optimal cost path
- Worst case searches every shorter/equal length path before the optimal one
- Let's also make the path **as long** as possible, assume every edge on optimal path is minimal edge length e
- So, at worst, optimal path is C*/e nodes from start
- $O(b^{C^*/e})$ nodes expanded to find optimal solution, where C* is the cost of the optimal solution and e is the minimum edge cost
- Can exceed $O(b^d)$ in number of nodes searched

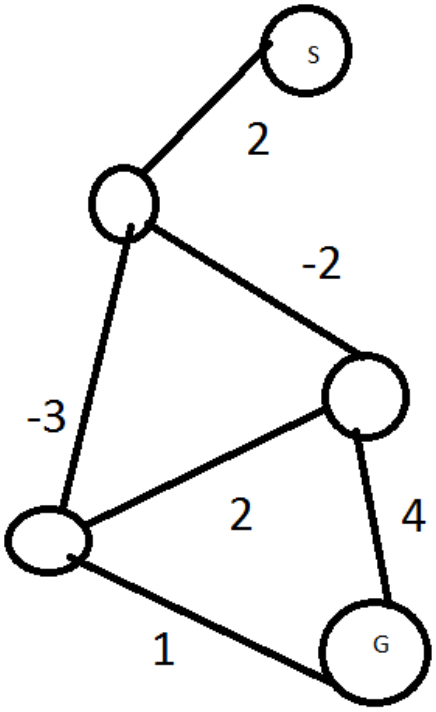
# Dijkstra's Worst Case




Here's a graph where $O(b^{C^*/e})$ and the number of nodes searched on the optimal path greatly exceed $O(b^d)$

| Name | Space Complexity | Time Complexity | Admissible | Complete |
|---|---|---|---|---|
| Breadth-First Search | $O(b^d)$ | $O(b^d)$ | True* for positive, equal-edge cost graphs | True |
| Depth-First Search | $O(bm)$ | $O(b^m)$ | False | False |
| Uniform Cost Search (Dijkstra's) | $O(b^{C^*/e})$ | $O(b^{C^*/e})$ | True | True |

# An Important Footnote to Dijkstra's (and BFS)

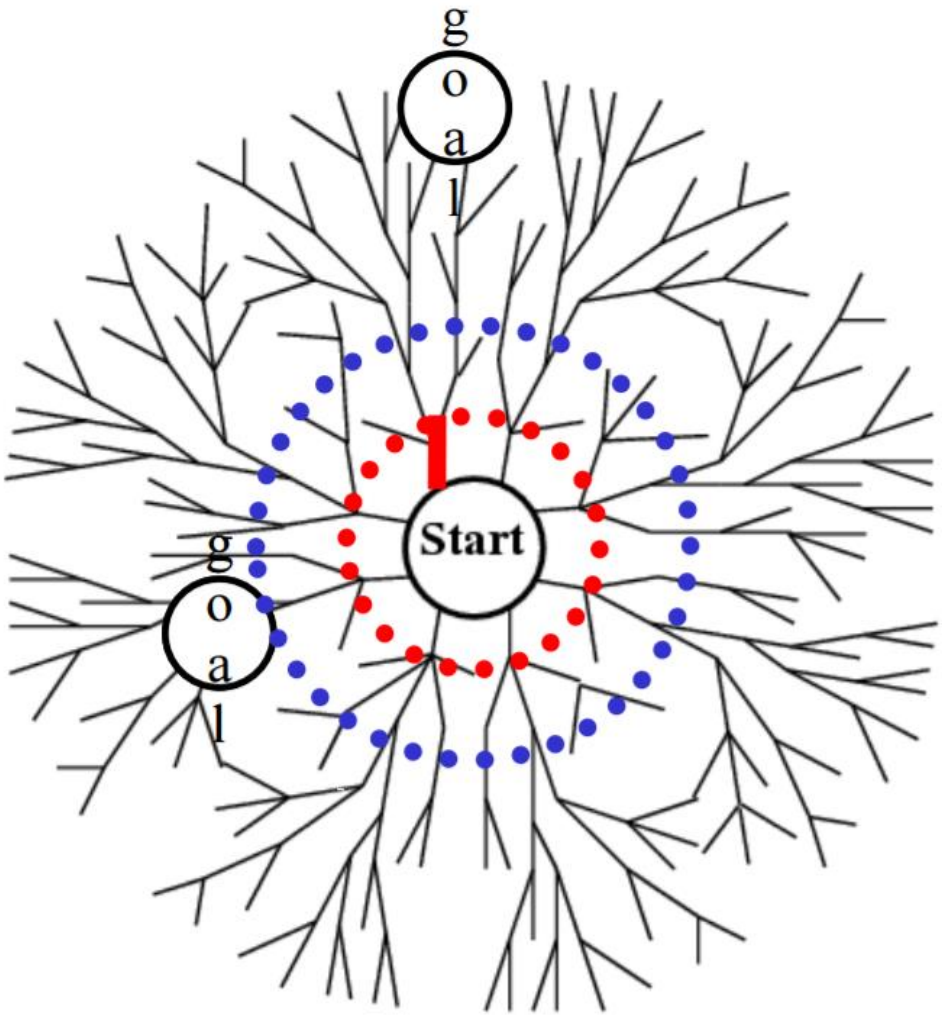



Dijkstra's is not complete or optimal on graphs with zero cost or negative cost edges. Will only work if each additional node increases the path sum by a nonzero amount.

# Iterative Deepening Search

**Strategy: Compromise between DFS and BFS to Reduce Space Consumption**

1. do DFS to depth 1
2. if no solution found, do DFS to depth 2
3. repeat by increasing "depth bound" until a solution found



- If the solution lies at depth d, will IDS find it?

  Yes. So, it's complete under the same conditions BFS is.

- If the solution lies at depth d, will IDS return the optimal (shortest path) to the solution?

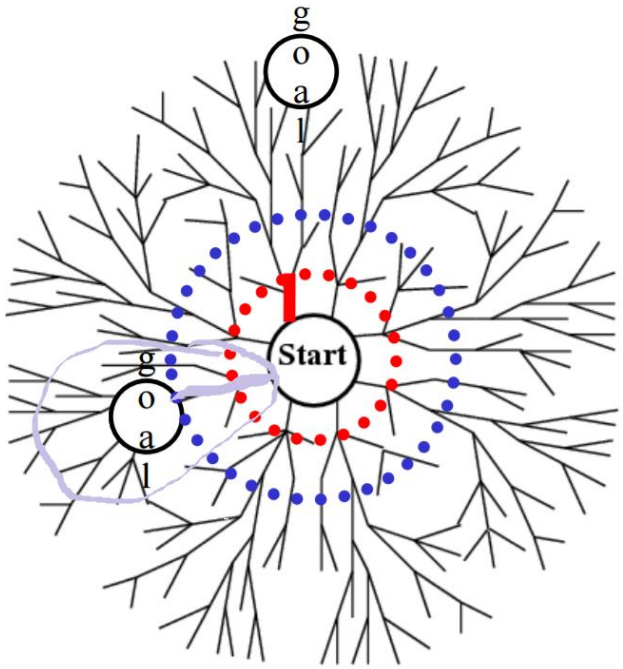
  Yes. It will terminate when it reaches the shallowest – lowest depth – goal state. It's also optimal under the same conditions BFS is.

# IDS Space Complexity

IDS will reach have its largest frontier at its last iteration.

At each iteration, DFS performed to depth d, where d is the depth of the solution.

It's computationally equivalent to the space-complexity of DFS searching the entire graph **O(bm)** of a graph of max depth d. So, the space complexity of IDS is **O(bd).**



# IDS Time Complexity

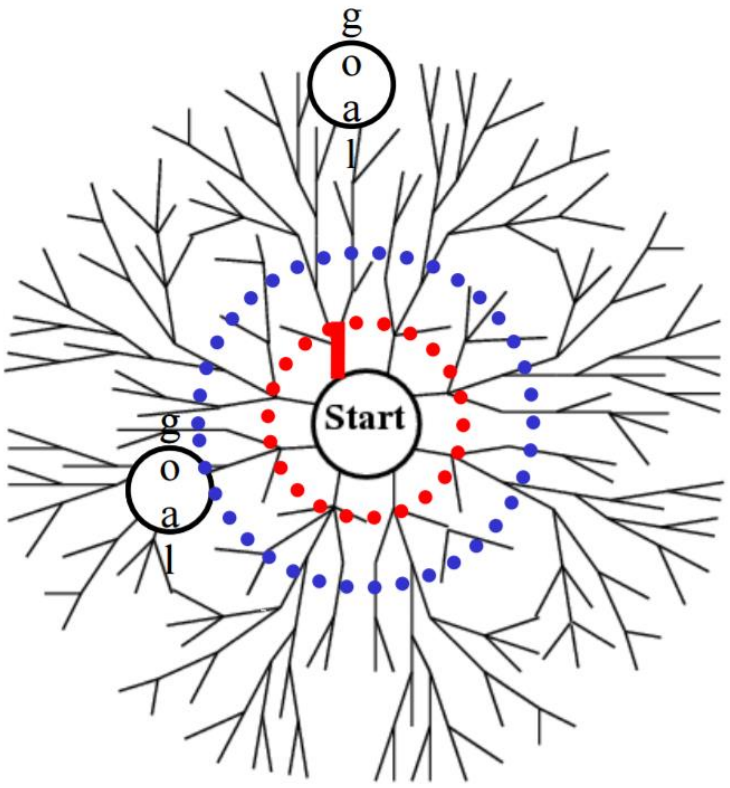
How many nodes will IDS search compared to BFS/DFS?

Solution lies at depth d.

It searches layer 1 d times, layer 2 d-1 times…layer d once.

$$db + (d-1)b^2 + (d-2)b^3 + \ldots b^d$$

**Because $b^d$ is still the largest term, $O(b^d)$, just like BFS.**



# Is the extra effort of IDS worth it?

Has advantages of BFS

- completeness
- optimality as stated for BFS

Has advantages of DFS

- limited space
- in practice, even with redundant effort it still finds longer paths more quickly than BFS

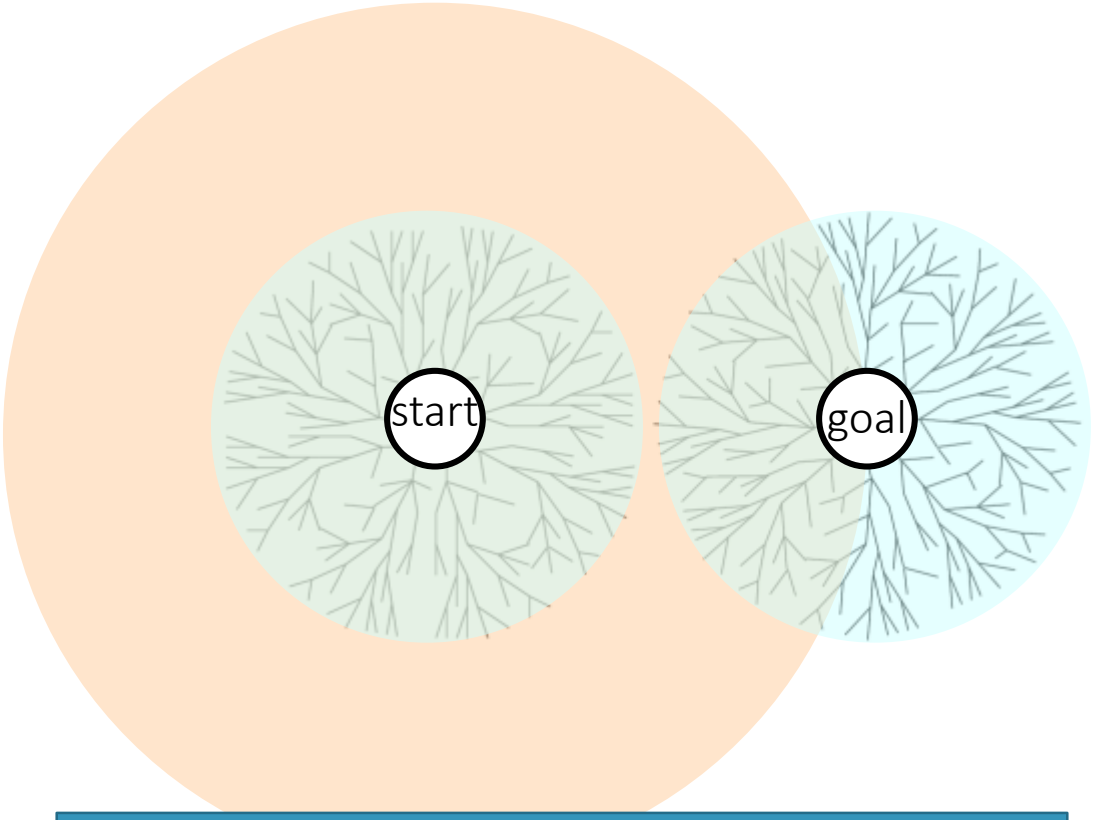
| Name | Space Complexity | Time Complexity | Admissible | Complete |
|---|---|---|---|---|
| Breadth-First Search | $O(b^d)$ | $O(b^d)$ | True* for positive, equal-edge cost graphs | True |
| Depth-First Search | $O(bm)$ | $O(b^m)$ | False | False |
| Uniform Cost Search (Dijkstra's) | $O(b^{C^*/e})$ | $O(b^{C^*/e})$ | True* for positive, non zero edge cost graphs | True |
| Iterative Deepening Search | $O(bd)$ | $O(b^d)$ | True* for positive, equal-edge cost graphs | True |

# Bidirectional Search

- Strategy: given a known start and goal state, start search from both ends in the hopes that the two searches will meet
- If the two searches meet, implies a path from the start to the goal
- Uses **breadth-first search** from each end

# Bidirectional Search

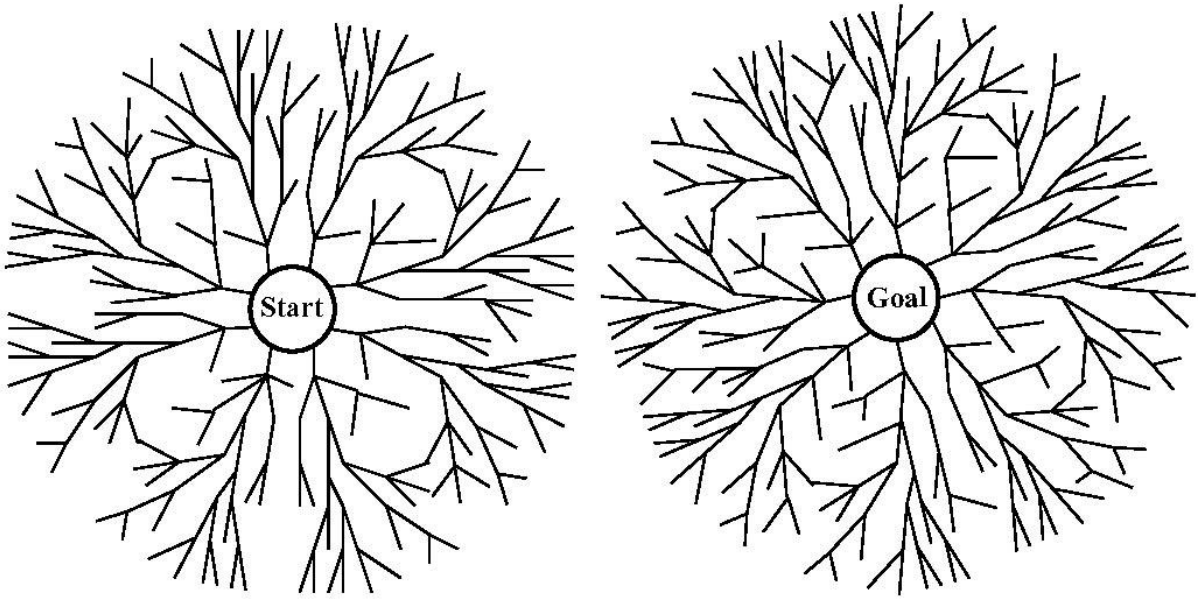
- **BFS** from both start and goal state
- BFS from goal travels reverse arcs in directed graphs
- Stop when Frontiers meet
- Generates $O(b^{d/2})$ instead of $O(b^d)$ nodes




If you can't compute predecessor nodes backwards from the goal, **this algorithm can't be applied.**

| Name | Space | Time | Admissible | Complete |
|---|---|---|---|---|
| Breadth-First Search | $O(b^d)$ | $O(b^d)$ | True* for positive, equal-edge cost graphs | True |
| Depth-First Search | $O(bm)$ | $O(b^m)$ | False | False |
| Uniform Cost Search (Dijkstra's) | $O(b^{C^*/e})$ | $O(b^{C^*/e})$ | True*for positive, non zero edge cost graphs | True |
| Iterative Deepening Search | $O(bd)$ | $O(b^d)$ | True*for positive, equal-edge cost graphs | True |
| Bidirectional Search | $O(b^{d/2})$ | $O(b^{d/2})$ | True on the same graphs as BFS | True |

# Which Direction Should We Search?




- Forward, backwards, or bidirectional

## Criteria:

- How many start and goal states?
- Branching factors in each direction
- How much work is it to compare states?

# Picking a Search Direction

A problem with a single start state but many possible goal states? (Like many board/strategy games)

Forward breadth-first or iterative deepening

A problem with many start states and a single goal state? (Like an 8-puzzle)

Backwards breadth-first or iterative deepening

A problem with a single start state and a single goal state? (Like a maze)

Bidirectional search or forward iterative deepening

# Search Algorithms Steps

1. Initialize frontier

LOOP:

   2. Pull from frontier
   3. Check if goal state
   4. Add children

function *BreadthFirstSearch:*

```
initialize queue frontier
initialize explored set to empty
insert start state into frontier
while frontier is not empty:
  cur = frontier.remove()
  if cur is a goal state:
      compute and return solution
  explored.add(cur)
  generate cur's children as children
  for child in children:
      if child not in explored or frontier:
          frontier.insert(child)
```

function *UniformCostSearch:*

```
initialize priority queue frontier
initialize explored set to empty
insert start state into frontier with 0 priority
while frontier is not empty:
    cur = frontier.remove()
    if cur is a goal state:
        compute and return solution
    explored.add(cur)
    generate cur's children as children
    for child in children:
        if child not in explored:
            frontier.insert(child, path cost)
```

function *DepthFirstSearch:*

```
initialize stack frontier
initialize explored set to empty
push start state onto frontier
while frontier is not empty:
  cur = frontier.pop()
  if cur is a goal state:
      compute and return solution
  explored.add(cur)
  generate cur's children as children
  for child in children:
      if child not in explored or frontier:
          frontier.push(child)
```

# How do we do better than uninformed search?

- Be informed!
- In the real world, we usually know more about the problem than the current state, the start state, and goal state.
- We can **domain knowledge** from problem experts.
- For example, when solving a maze, you usually do it with an approximate knowledge of where the exit is.
- The key: use a heuristic function to evaluate each state.