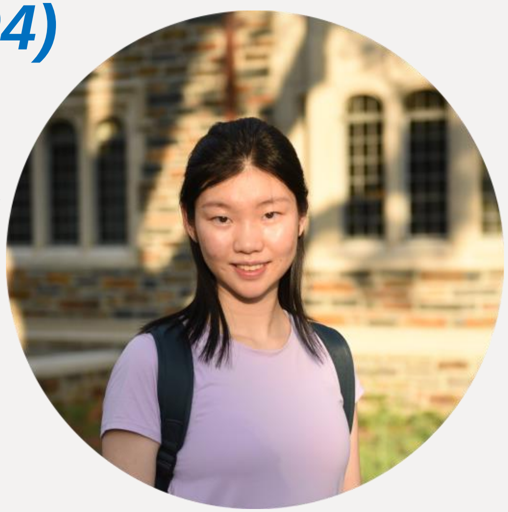
Class starts after this song

Kings of Convenience – Homesick (2004)

requested by Carrie Hang (TA-of-CM7)

Amateur dancer. Jellyfish lover. Night owl.



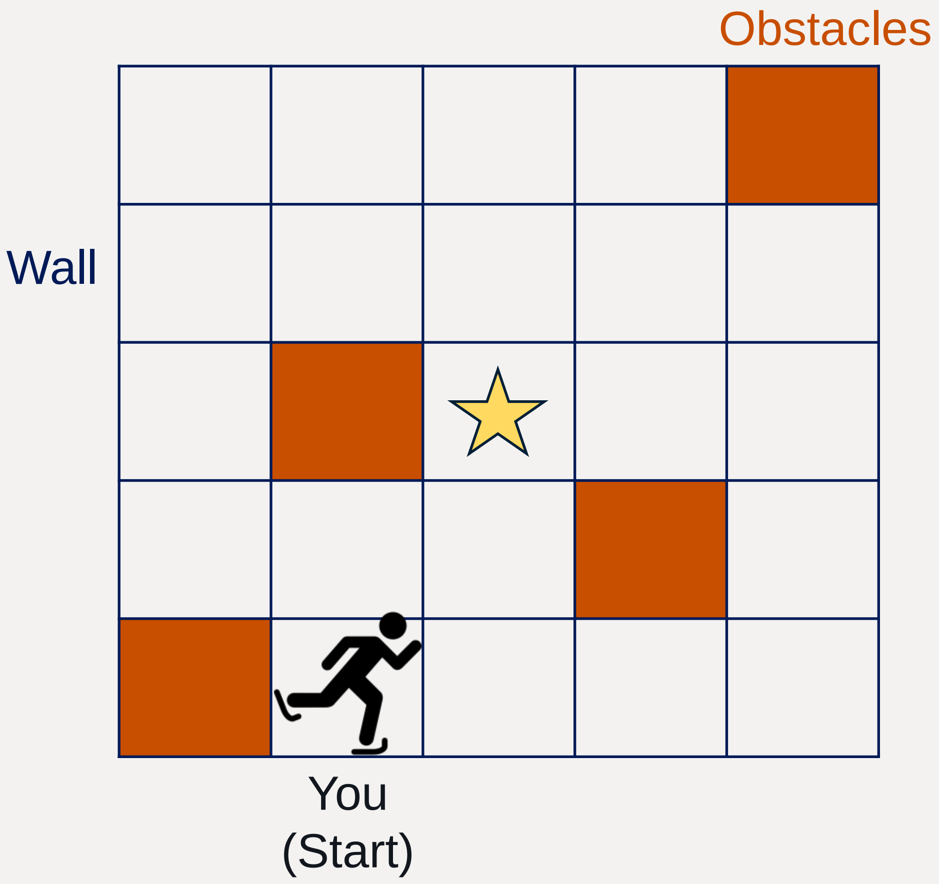
Duke

CS230 Spring 2024
EM A/D: Graph Applications

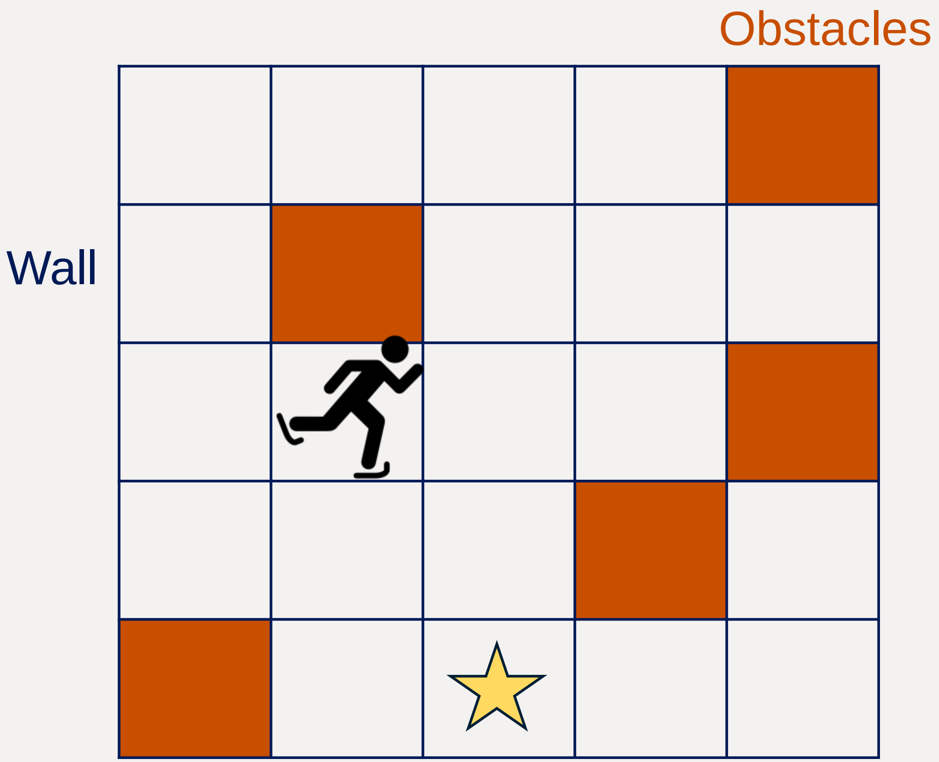
Duke

Ice Skating

- You're a lousy ice skater
  - Can only move NSEW
  - Cannot turn while in motion
  - Cannot decelerate while moving
  - Only stops when hitting something
  - Once stopped, can turn

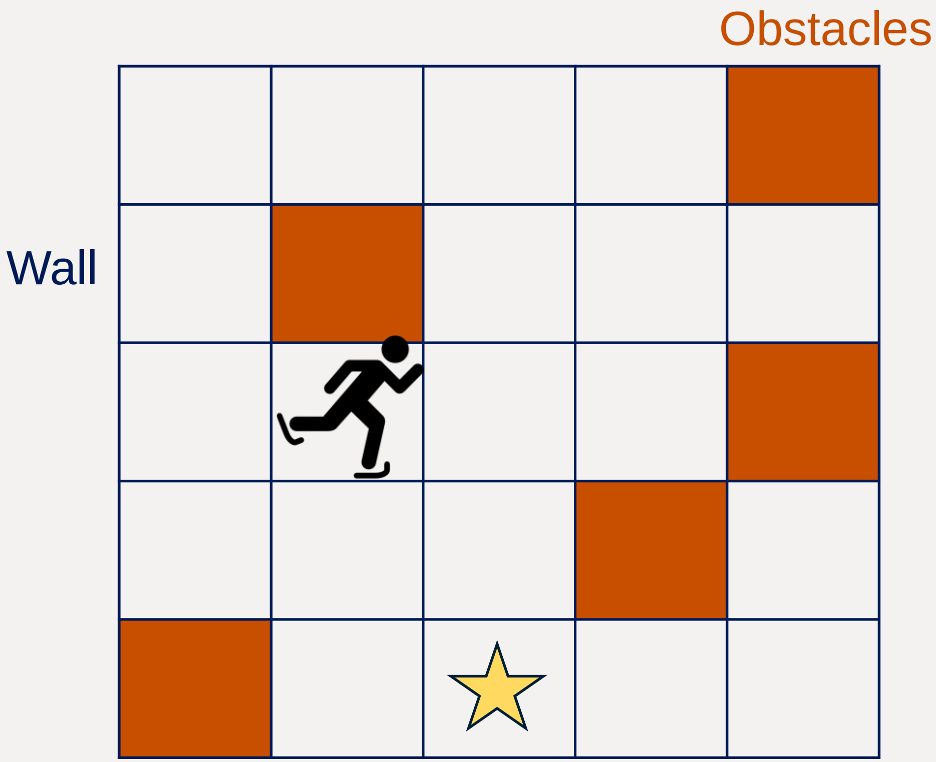


Ice Skating



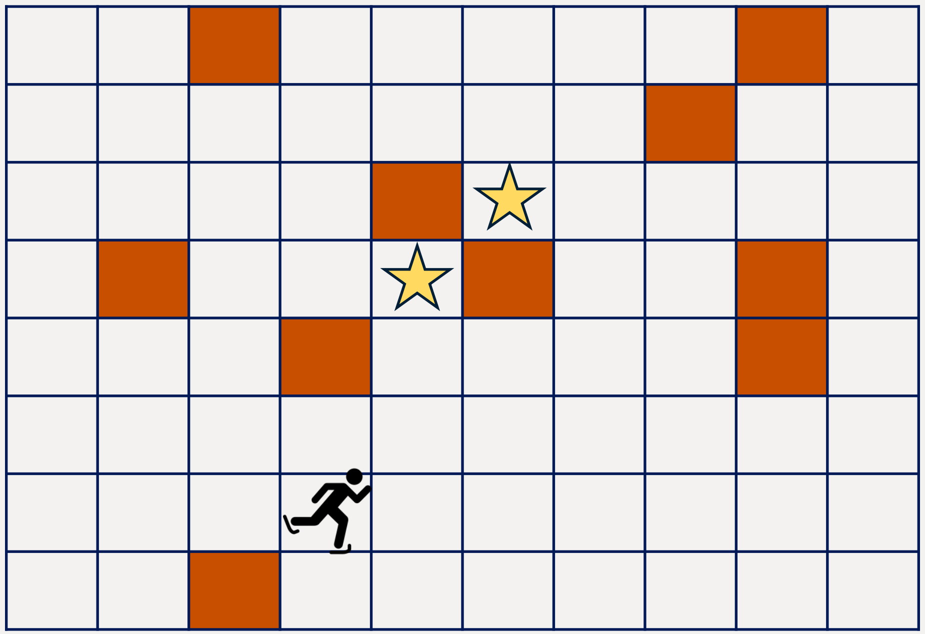
Duke

Ice Skating



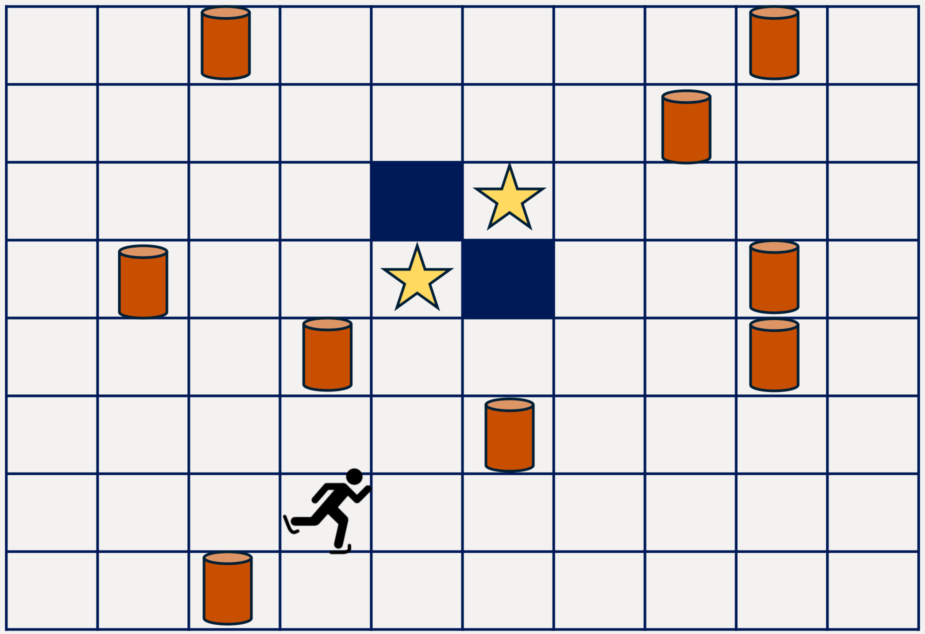
Ice Skating

- Which star is reachable, and how?



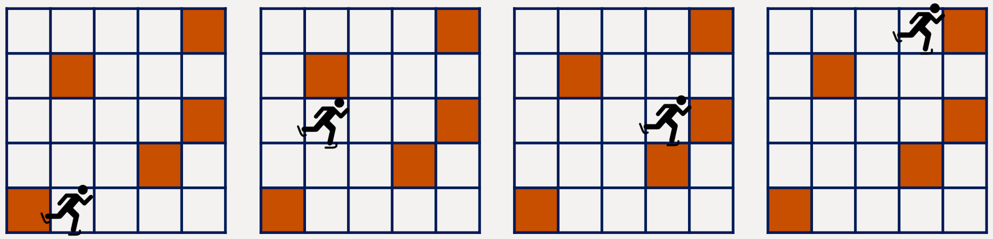
Ice Skating

- What if some obstacles can move?
  - If you hit some movable obstacles, you “push them forward” until you both hit a wall

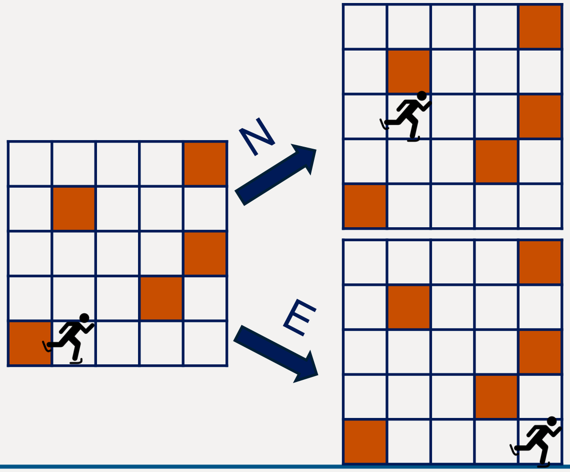


Search Problem and State Space Graphs

- An abstraction of the "world":
  - A state space



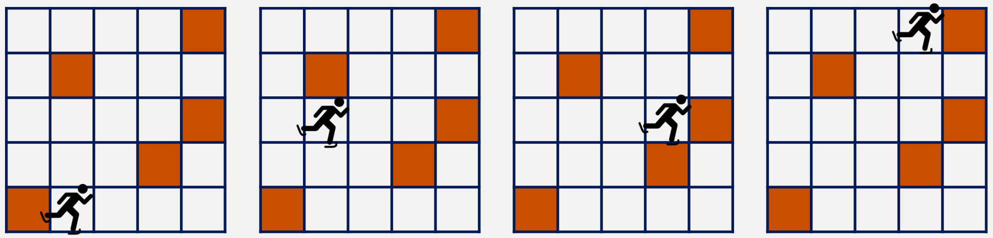
  - A successor function
  - A start state
  - A goal test

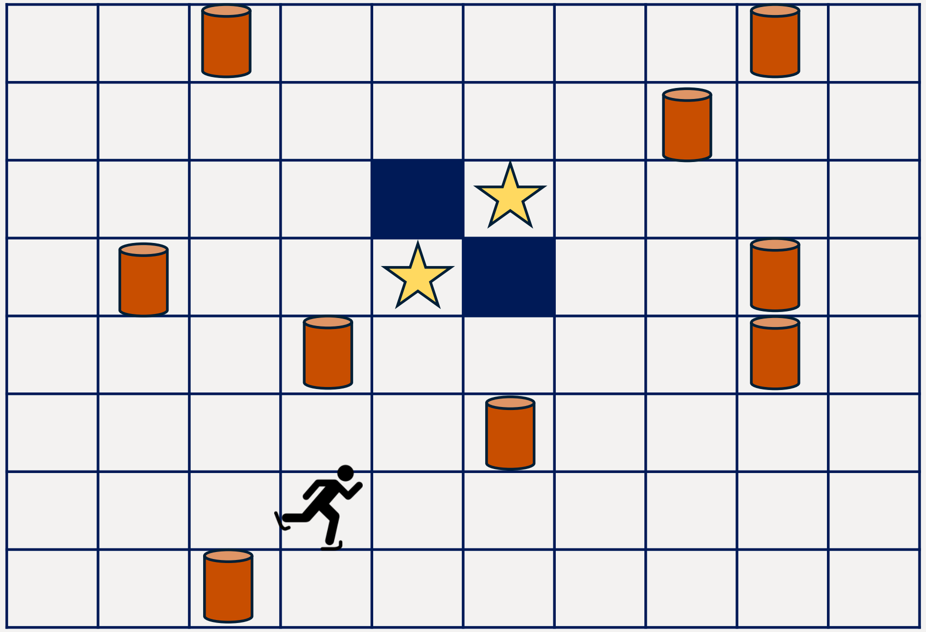


- How many successors for each state if:
  - Obstacles can't move?
  - Obstacles can move?

Duke

How large is the graph?

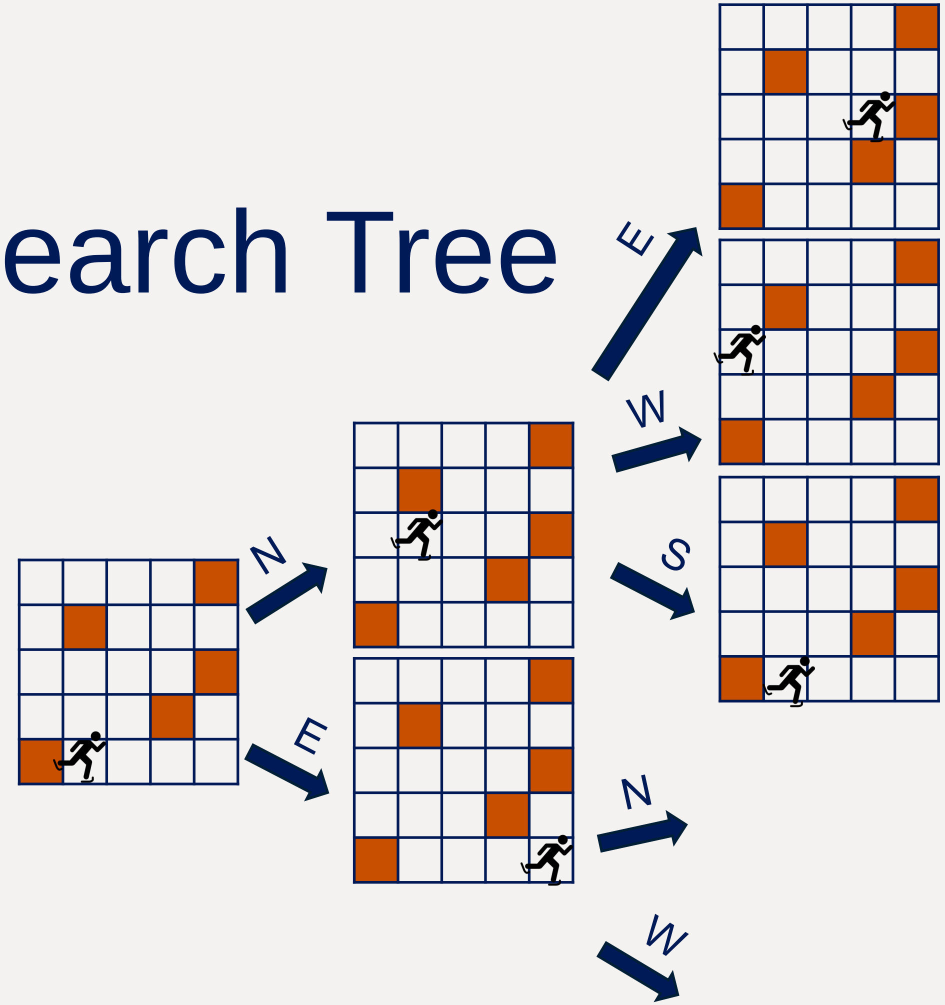




- Obstacles can't move:
  - 20 states (25 grids – 5 occupied)
- Obstacles can move (need CM7):
  - $\binom{25}{5} \times 20 \approx 10^6$ for the 5x5 grid
  - $\approx 10^{13}$ for the 10x8 grid

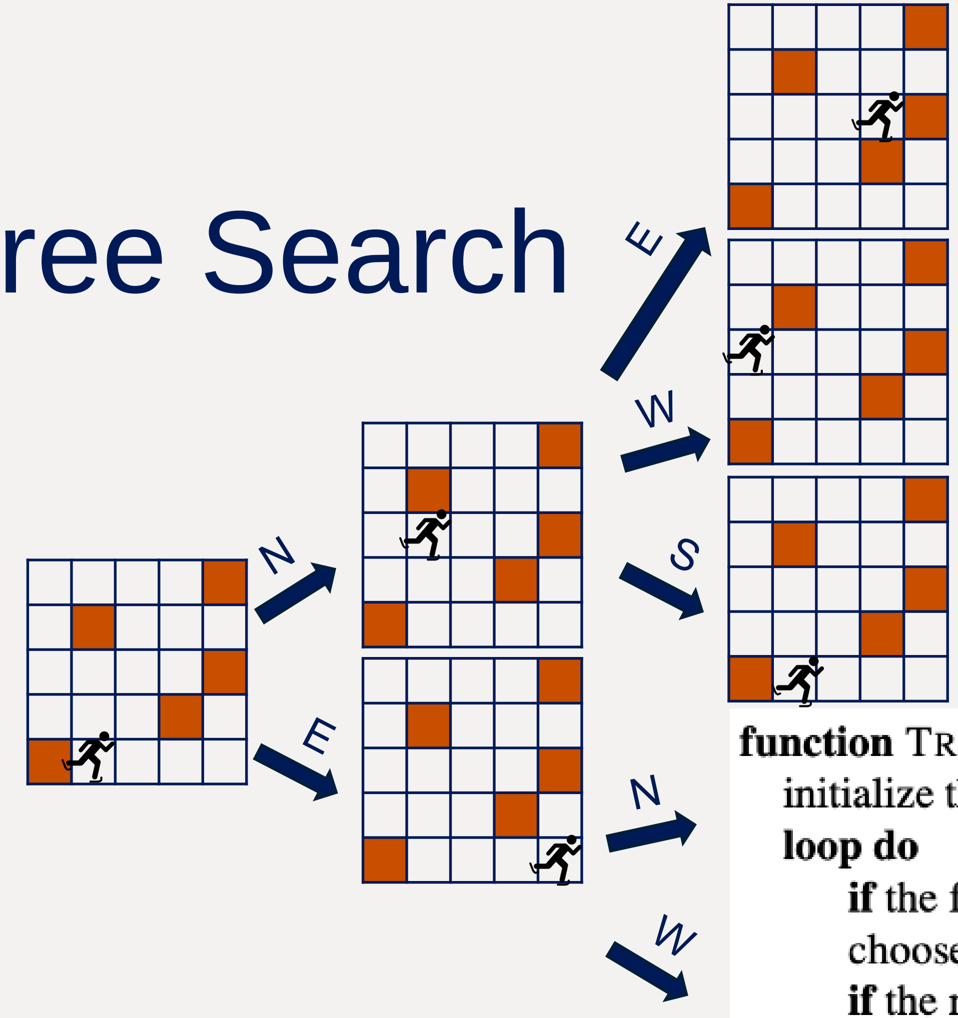
Duke

Search Tree



- Each vertex in the tree is both a state and a "plan of movement"
- Usually contains repeated states
- Is usually infinite

Tree Search

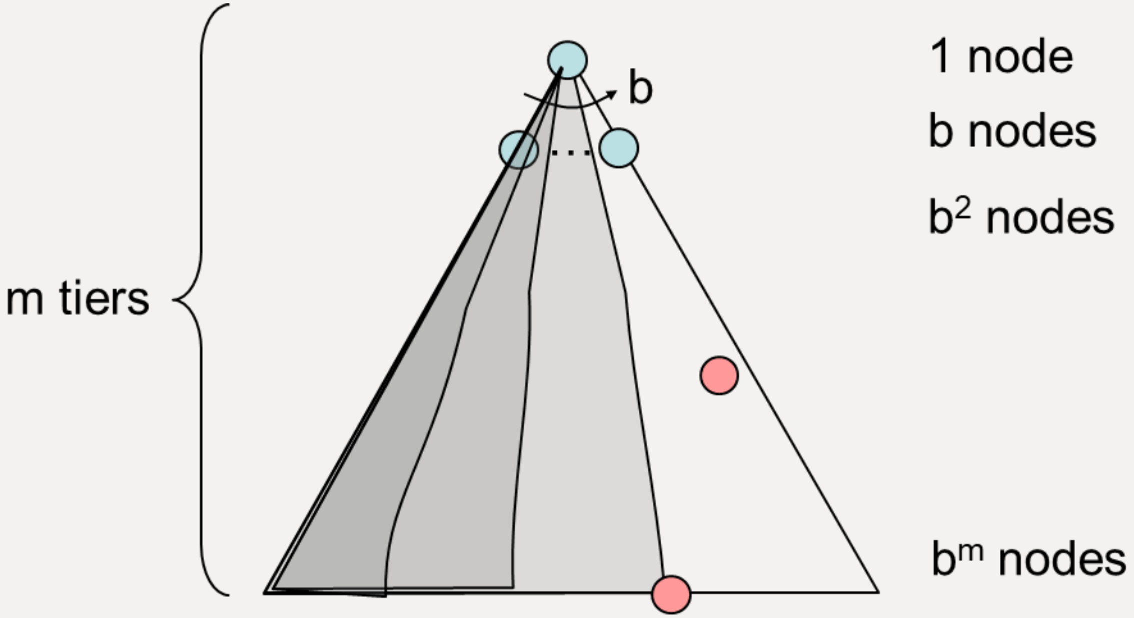


- Depth-first Search (DFS)
- Breadth-first Search (BFS)
- Iterative-Deepening (IDS)
- Uniform-Cost Search (UCS)

function TREE-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
  loop do
    if the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state **then return** the corresponding solution
    expand the chosen node, adding the resulting nodes to the frontier

Depth-first Search (DFS)

- Not guaranteed to stop (can cycle between repeated states)
- Not guaranteed to find the “optimal” solution
  - What is “optimal”? Need an explicit criteria, e.g., # of steps/actions



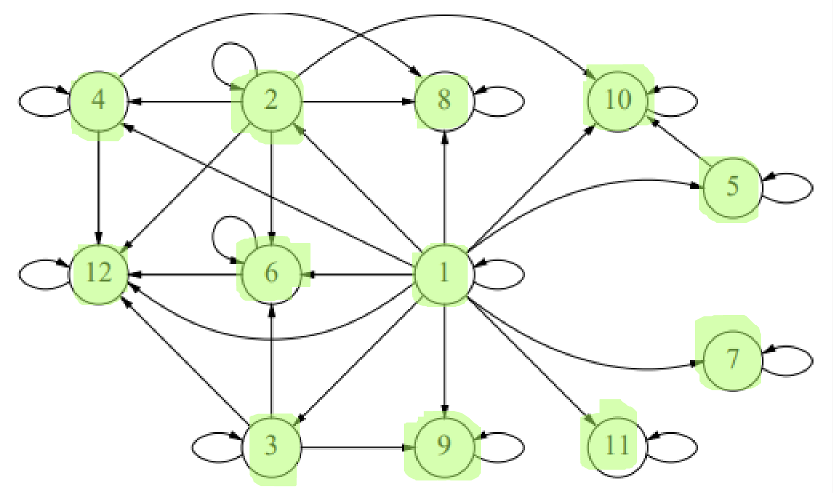
from Berkeley CS188 AI class

Topological Sorting a DAG using DFS

1. Add a “pseudo-vertex” u and edges from u to all real vertices
   - This guarantees one DFS can traverse the whole graph
2. Start a DFS from u
3. Append a vertex v to the order only after all children of v are already in the order
4. Remove u, then reverse the order of vertices

Topological Sorting a DAG using DFS

- Visit the smallest child first
- We don't need to add the fake vertex u here because vertex 1 has that property



- Global order:

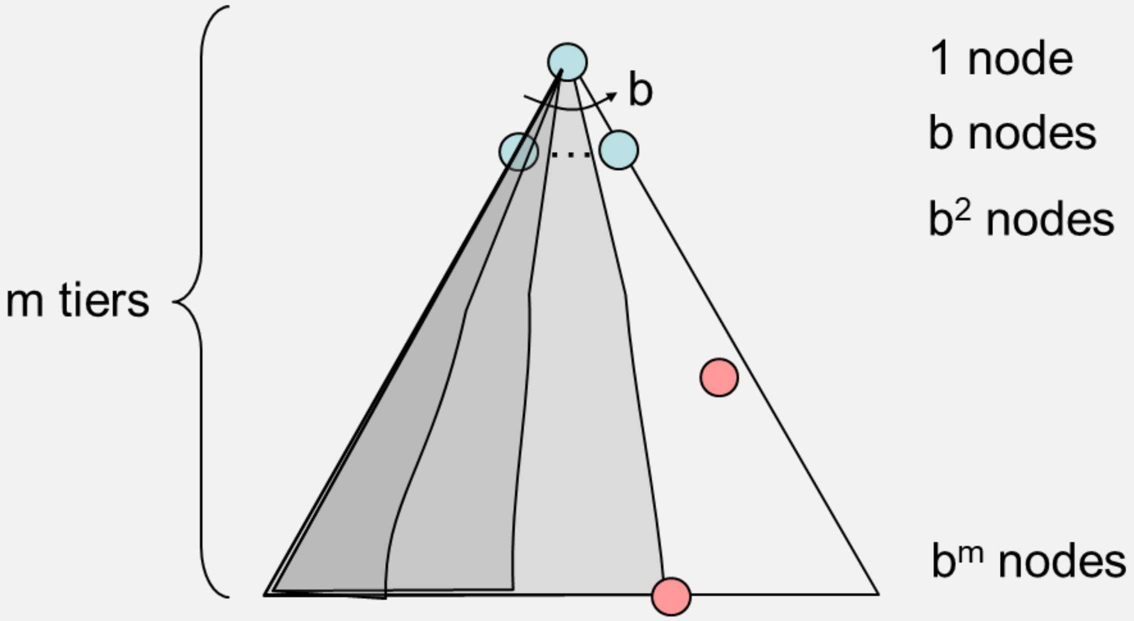
  8,12,4,6,10,2,9,3,5,7,11,1

- Topological order:

  1,11,7,5,3,9,2,10,6,4,12,8

Depth-first Search (DFS)

- Not guaranteed to stop
- Not guaranteed to
  find the “optimal” solution

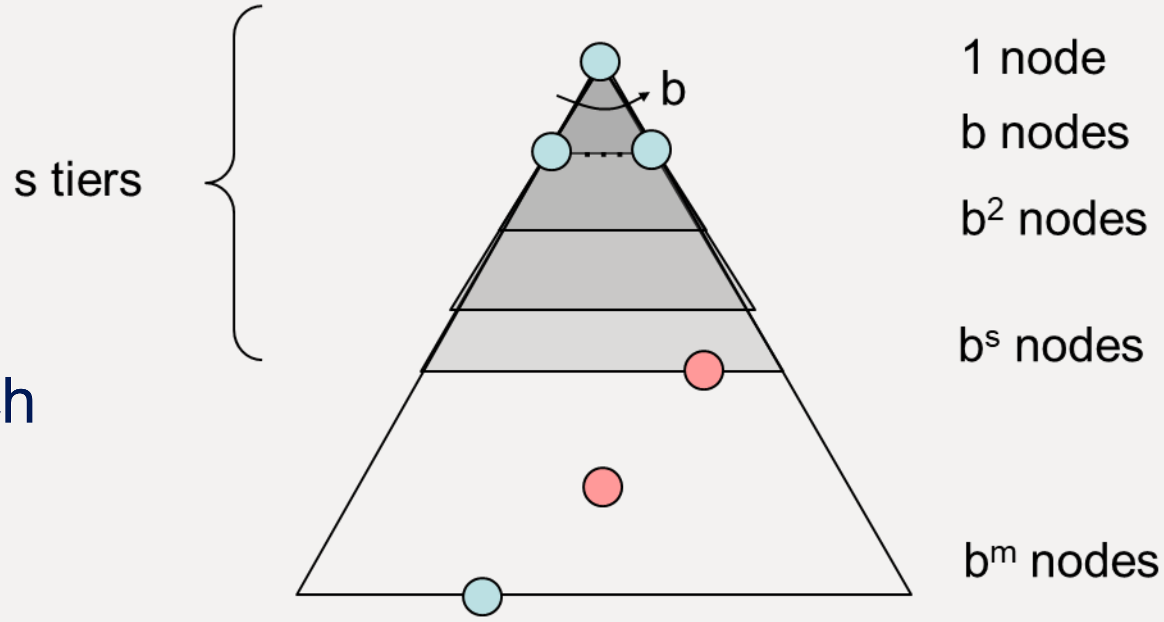


from Berkeley CS188 AI class

Duke

Breadth-first Search (BFS)

- Finds a solution (if exists) in finite steps
- Finds the “optimal” solution if each step costs the same

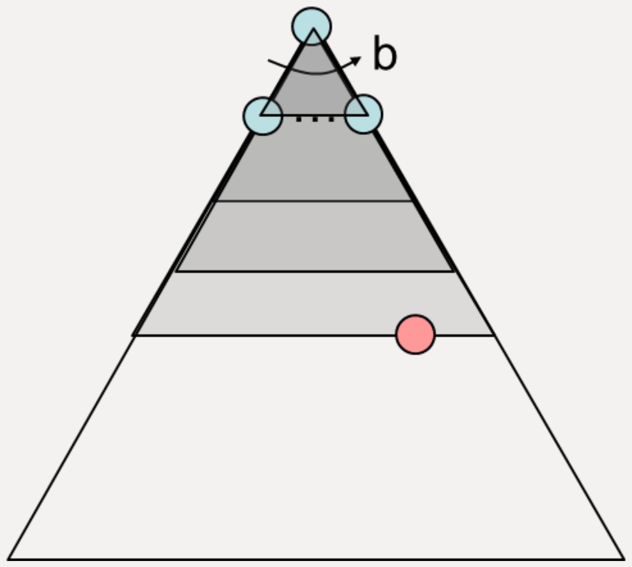


from Berkeley CS188 AI class

Duke

Iterative-Deepening (IDS)

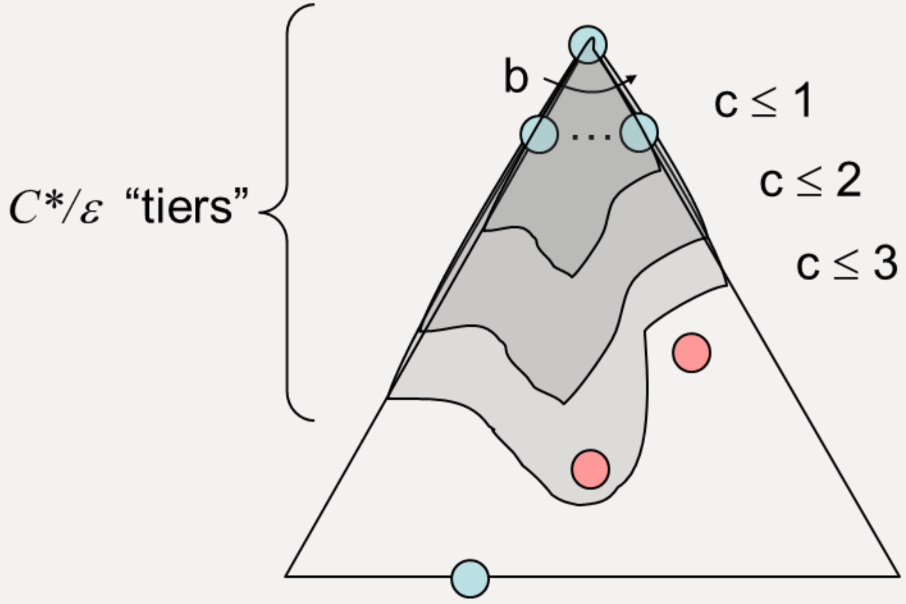
- Combines the best of two worlds
  - BFS’s time efficiency/ performance guarantee
  - DFS’s memory efficiency



from Berkeley CS188 AI class

Uniform Cost Search (UCS)

- Maintain a priority queue capturing the (cumulated) cost of each node
- Expand the lowest one in the frontier at each step
- Extends optimality to weighted-cost scenarios



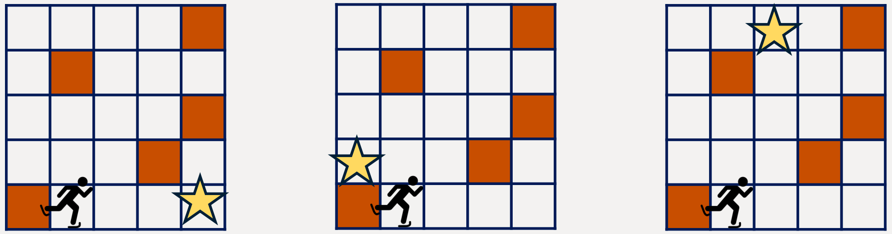
from Berkeley CS188 AI class

Tree Search

function TREE-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
  loop do
    if the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state **then return** the corresponding solution
    expand the chosen node, adding the resulting nodes to the frontier

- They really just differ in the frontier strategies

- They do the same thing, regardless of the goal



How to leverage more problem-specific knowledge?

- Read about informed searches/A* algorithm
- Take CS370

- State space graph and search trees are nice… for discrete problems
- What if the problem was continuous?

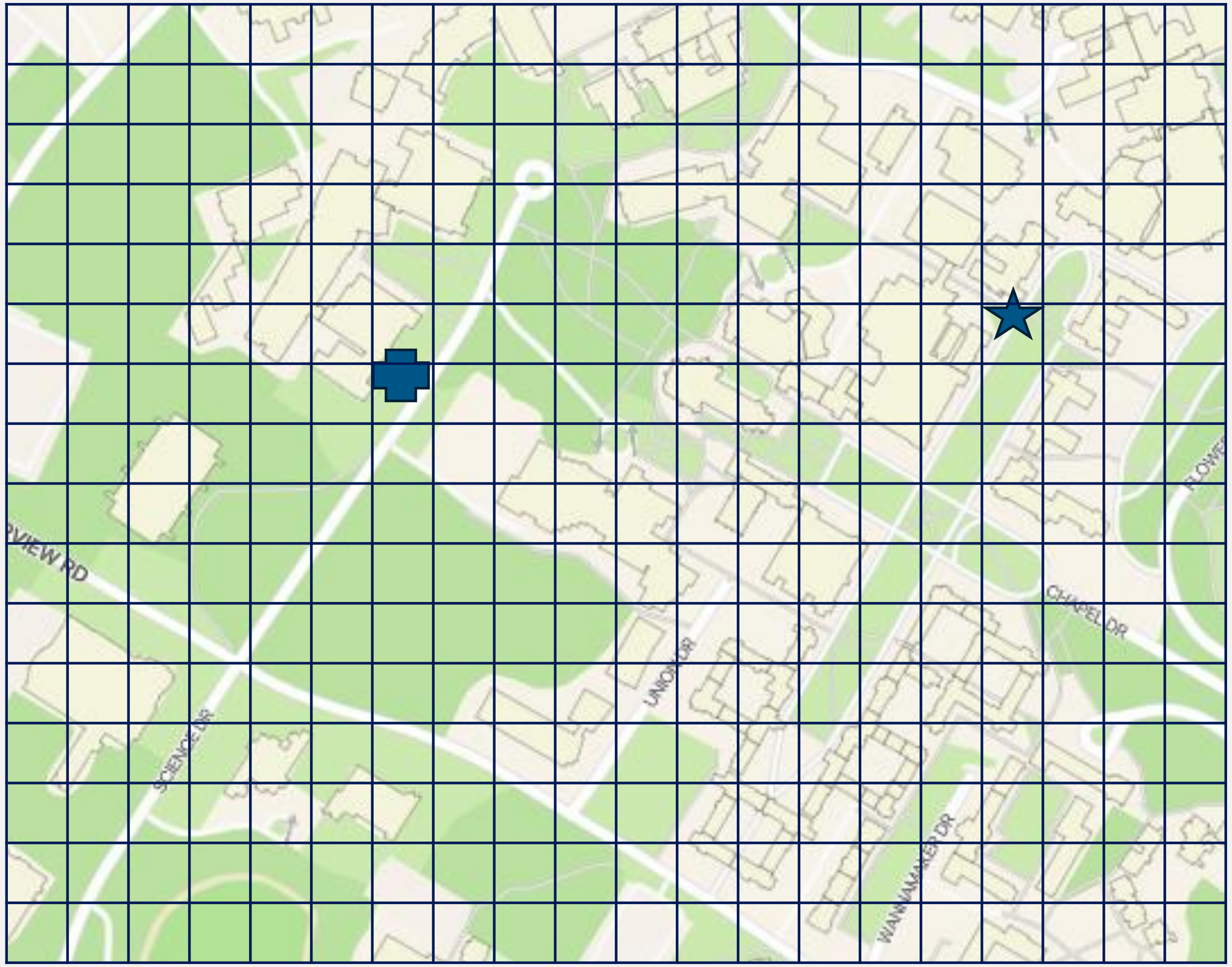
Motion planning

- Let’s fly a drone around campus
- Need not follow walkpaths
- But can’t fly over buildings

Approach 1: Grid

- “Discretize the world”

- How granularized?
- What are the possible moving patterns? Just NSEW?
- How to deal with partially-occupied grids?



Duke

Approach 2: Visibility Graph

- Focuses on obstacles
  - (For now) treat the drone as a single point of mass
  - Abstract all obstacles as convex polygons
  - Add vertices to all corners
  - Create edges between all unobstructed pairs of vertices
  - Run shortest path algorithms on resulting graph

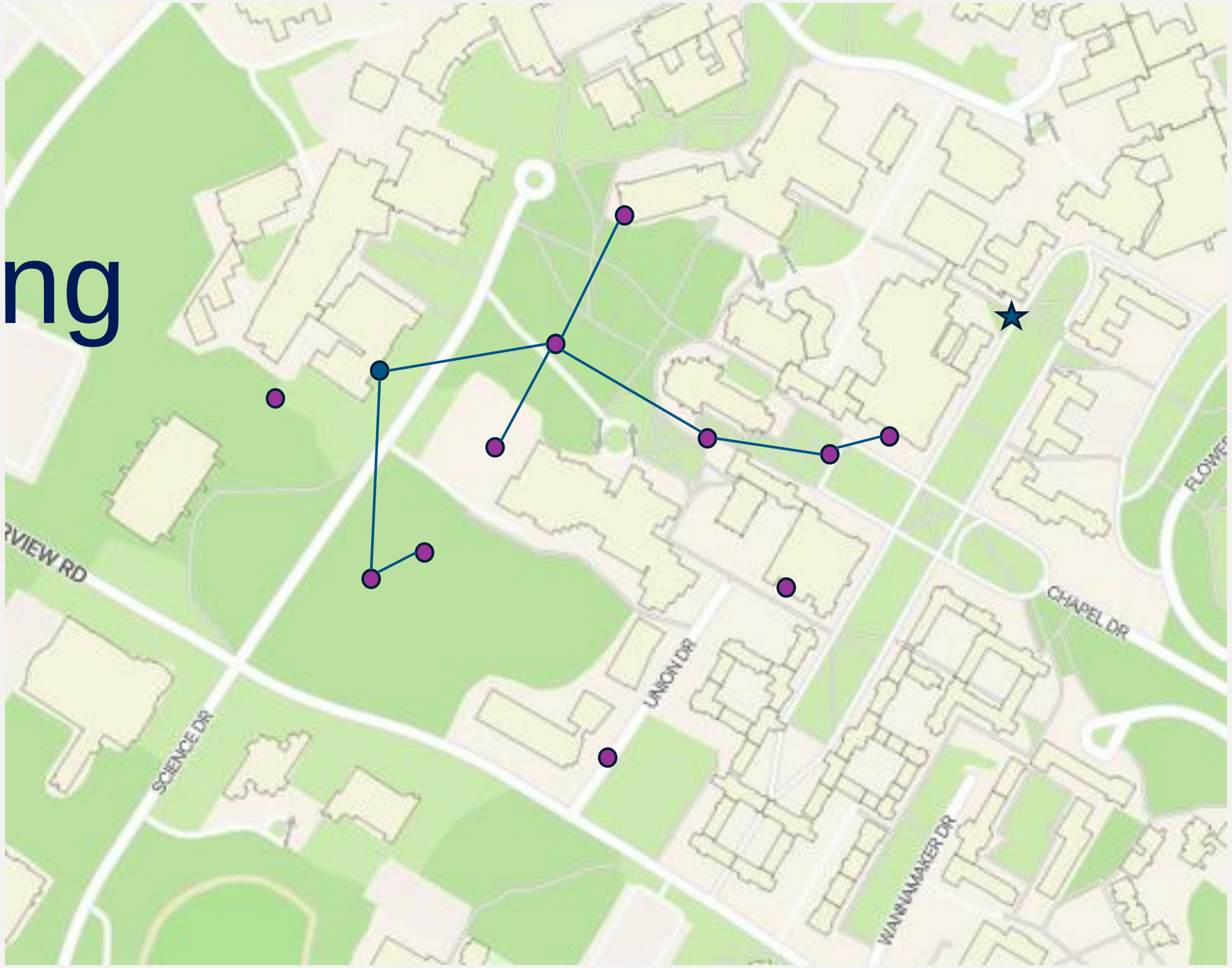
Approach 2: Visibility Graph

- Drone is not a point…
- No problem! We just leave some padding around obstacles
- However: hard to compute when have many obstacles with complicated shapes

Duke

Approach 3: Random Sampling

- Sample some location
  - Determine the nearest known location closest to the new one
  - If unobstructed, keep; otherwise discard



Duke

Many other approaches exist

- Hybrid ideas of these…
- Completely different ones…
- The first step is usually come up with a good model/abstraction of the real-world problem

Duke