复旦大学大数据学院
School of Data Science, Fudan University
魏忠钰

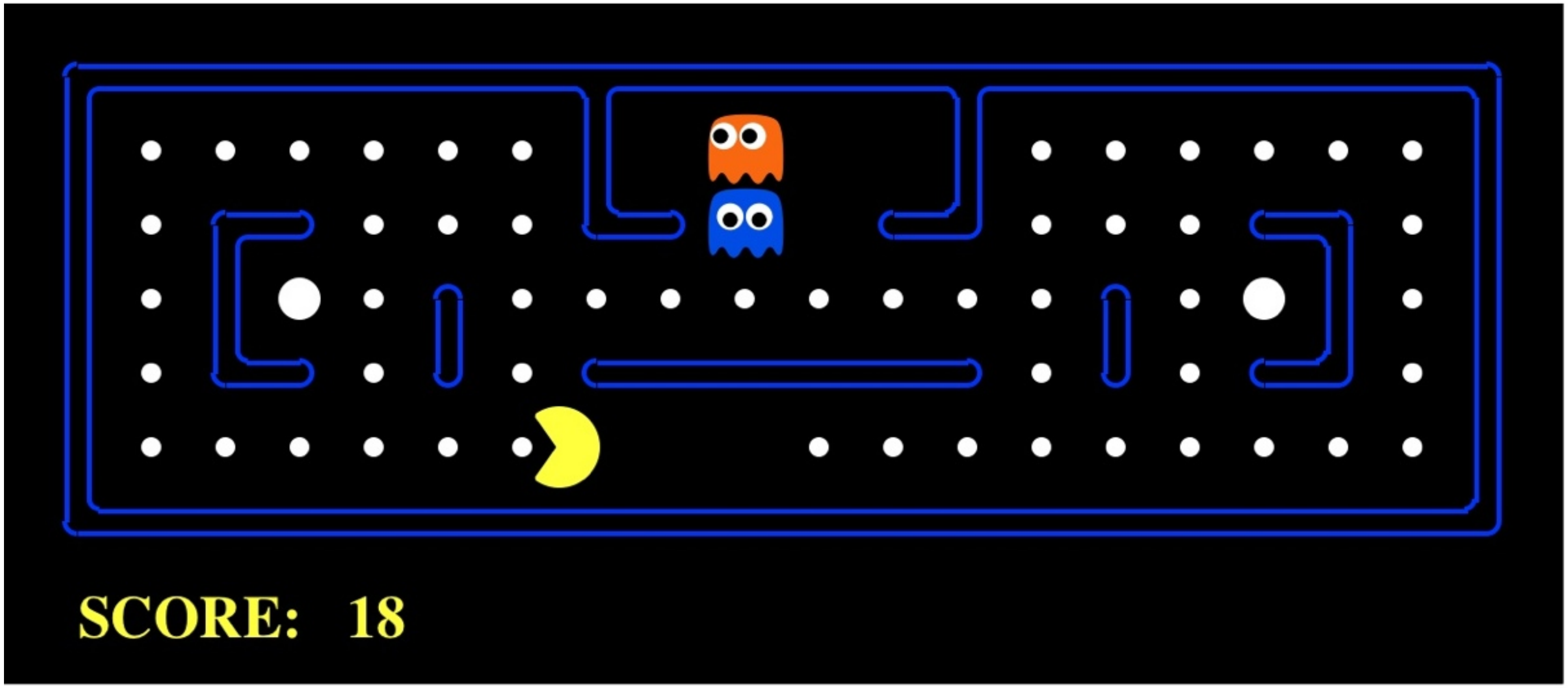
Search I

March 7th, 2018

Outline

- Search Problems
- Uninformed Search
  - Depth-First Search
  - Breadth-First Search
  - Uniform-Cost Search

Real world task - Pac-man

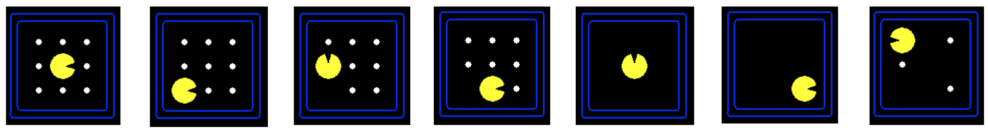


Search problems

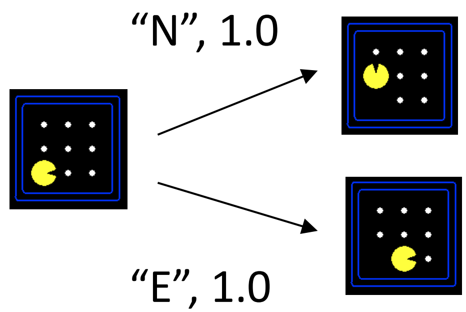
- A search problem consists of:
  - A state space
  - A successor function (with actions, costs)
  - A start state
  - A goal test

Search Problems - Pac-man

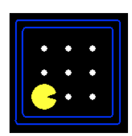
- A search problem consists of:
  - A state space



  - A successor function (with actions, costs)

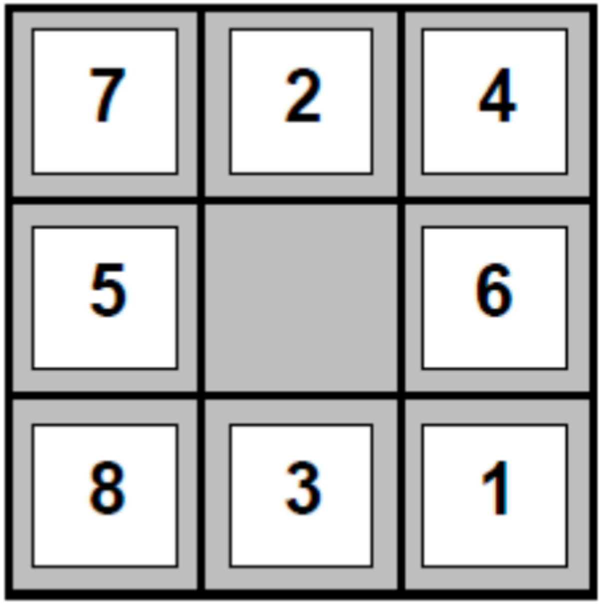


  - A start state

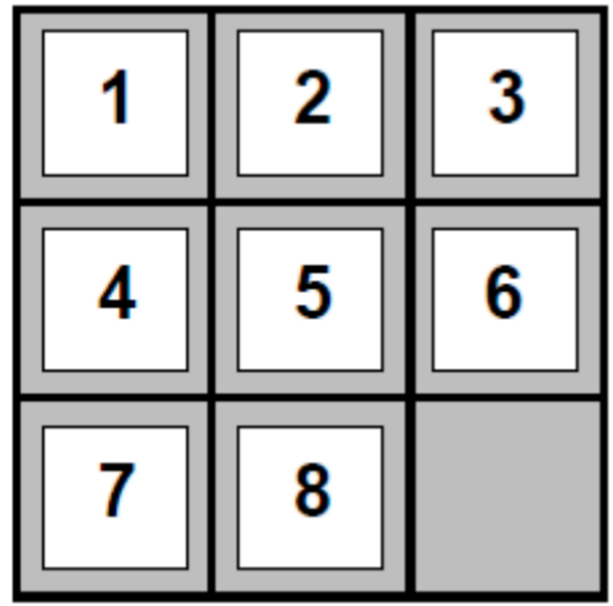


  - A goal test
    - Move to a specific position

Real world task : the 8-puzzle

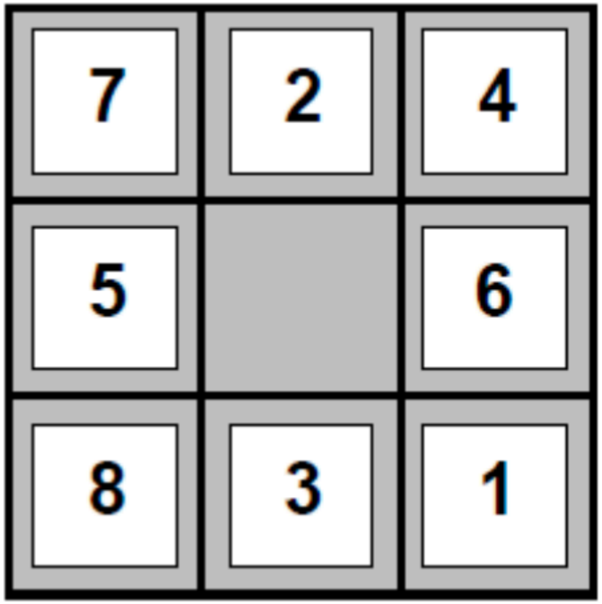


Start State

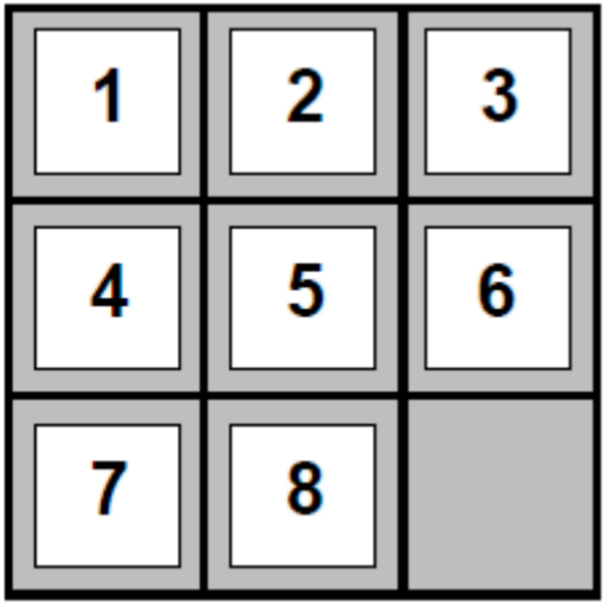


Goal State

Search Problems: the 8-puzzle



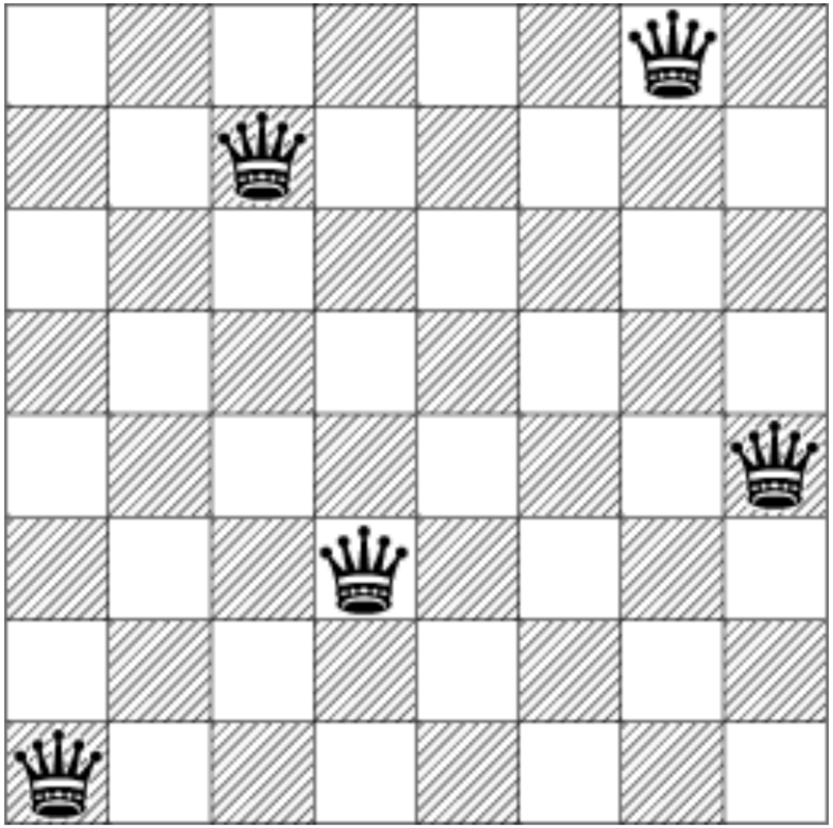
Start State



Goal State

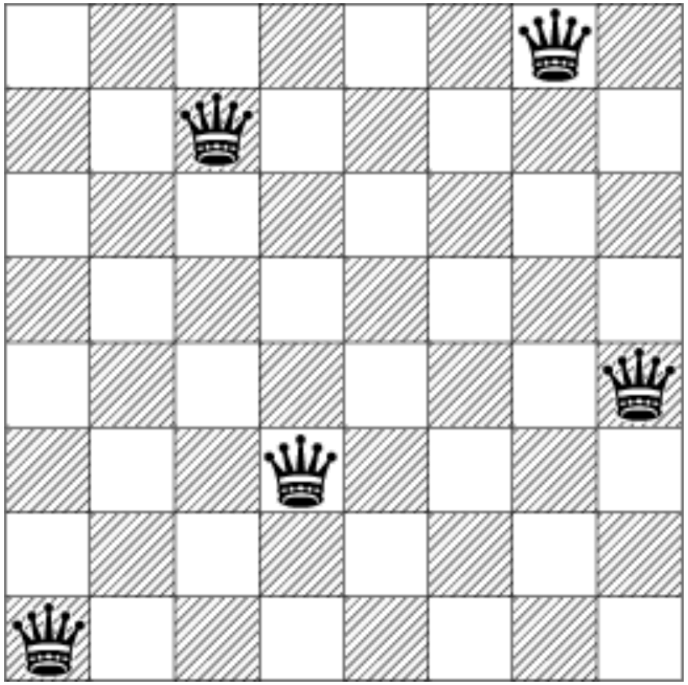
- State space
  - integer locations of 8 tiles.
- Successor function:
  - Move blank (up, down, right, left)

Real Task: 8-Queens Puzzle



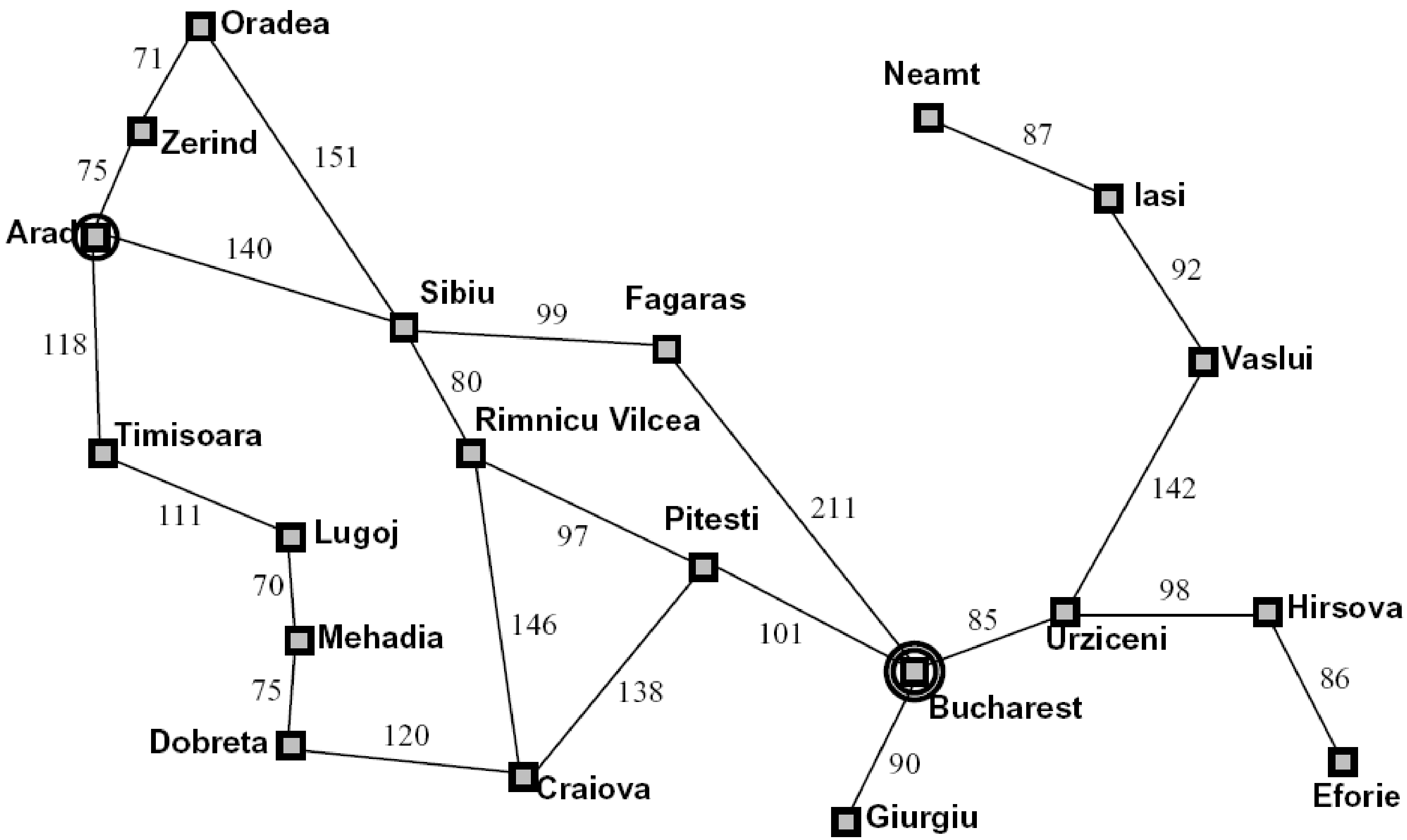
- Place 8 queens on a chessboard so that no two queens attack each other.
- A queen attacks any piece in the same row, column or diagonal.
- 3 more queens missing

Search Problem: 8-Queens Puzzle

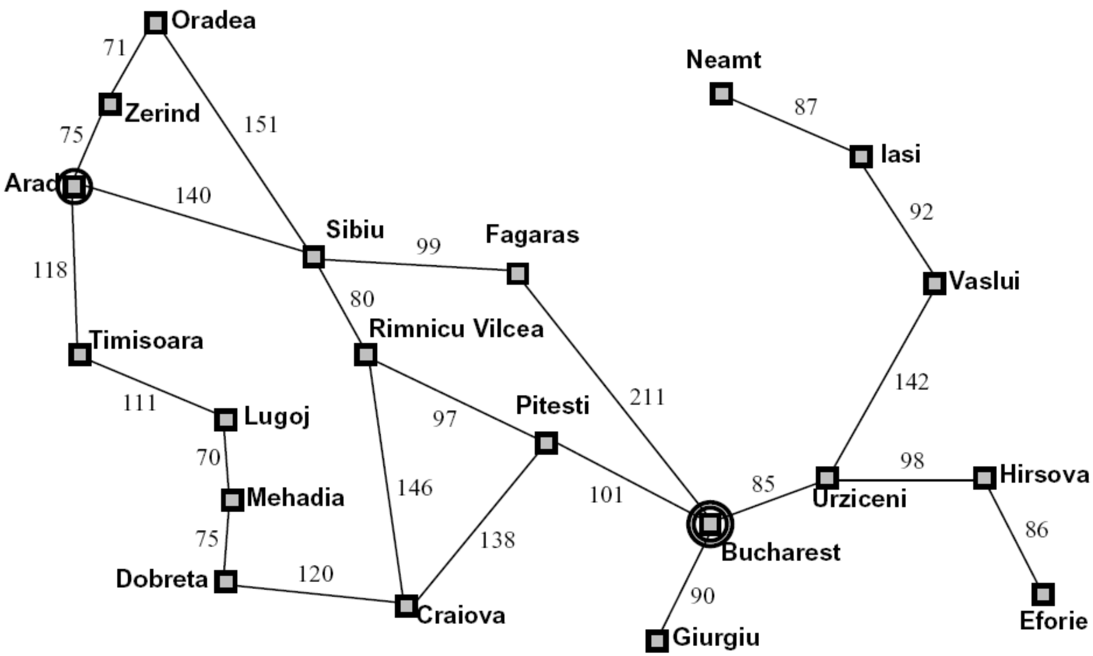


- **State space**: any arrangement of 0 to 8 queens on board
- **Successor function:** add a queen to any square
- **Start state**: blank board
- **Goal test**: 8 queens on board, non attacked

Real world task : Traveling in Romania



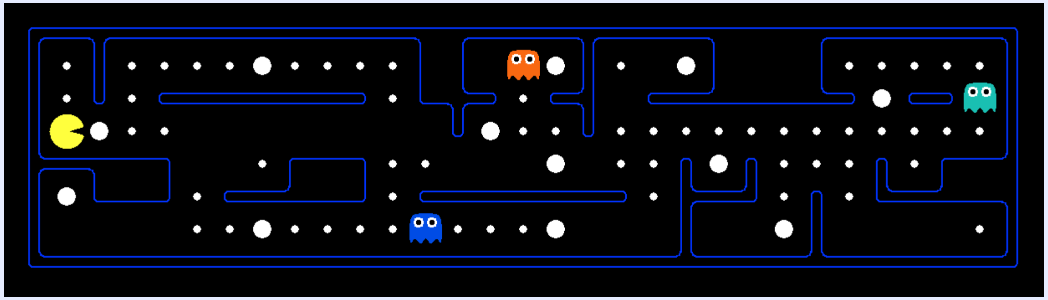
Search Problems: Traveling in Romania



- State space
  - Cities
- Successor function:
  - Roads: Go to adjacent city with cost = distance
- Start state:
  - Arad
- Goal test:
  - Is state == Bucharest?

What is in a State Space?

The world state includes every last detail of the environment

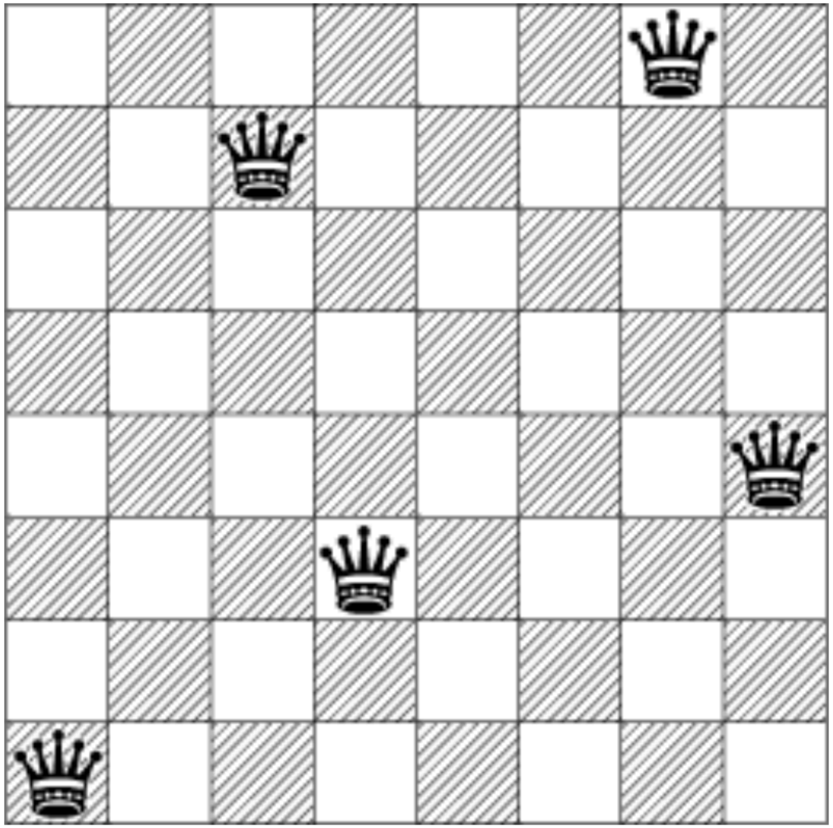


A search state keeps only the details needed for planning (abstraction)

- Problem: Pathing
  - States: (x,y) location
  - Actions: NSEW
  - Successor: update location only
  - Goal test: is (x,y)=END
- Problem: Eat-All-Dots
  - States: {(x,y), dot booleans}
  - Actions: NSEW
  - Successor: update location and possibly a dot boolean
  - Goal test: dots all false

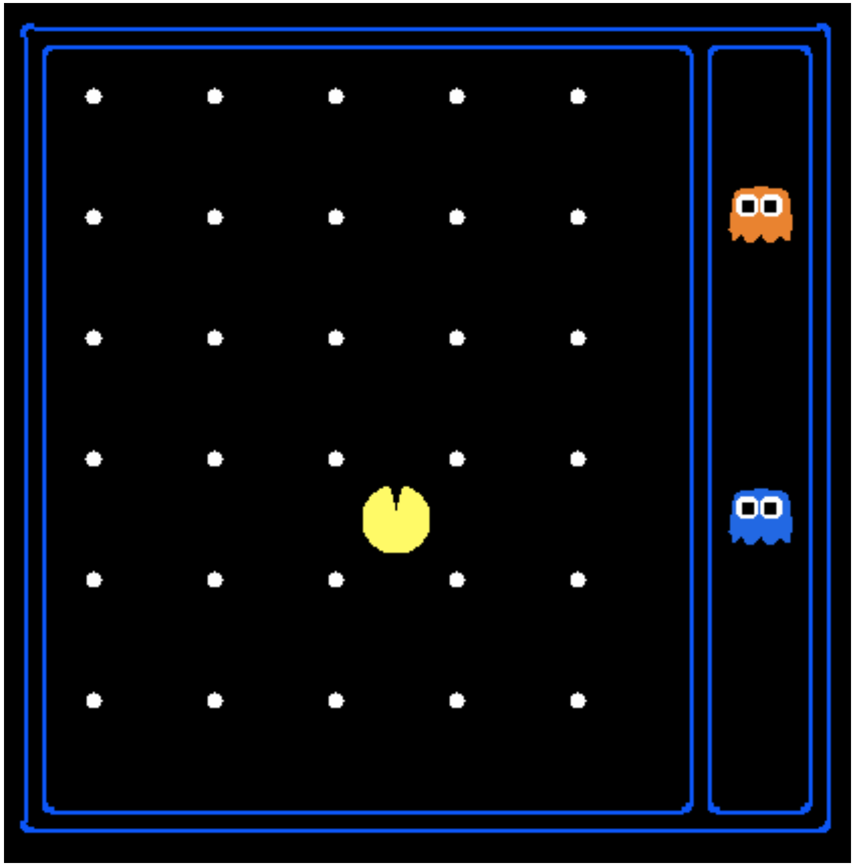
State Space Sizes?

- World state:
  - Board blanks: 64
  - Queen number: 8
- How many
  - World states?
  - 64^8

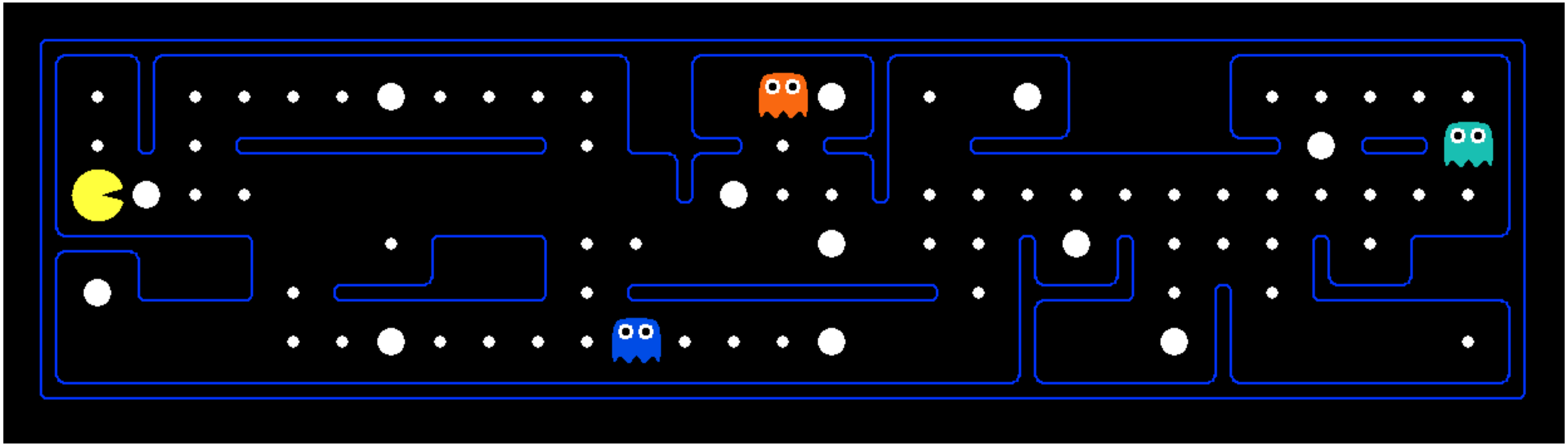


State Space Sizes?

- World state:
  - Agent positions: 120
  - Food count: 30
  - Ghost positions: 12
  - Agent facing: NSEW
- How many
  - World states?
    $120 \times (2^{30}) \times (12^{2}) \times 4$
  - States for pathing?
    120
  - States for eat-all-dots?
    $120 \times (2^{30})$

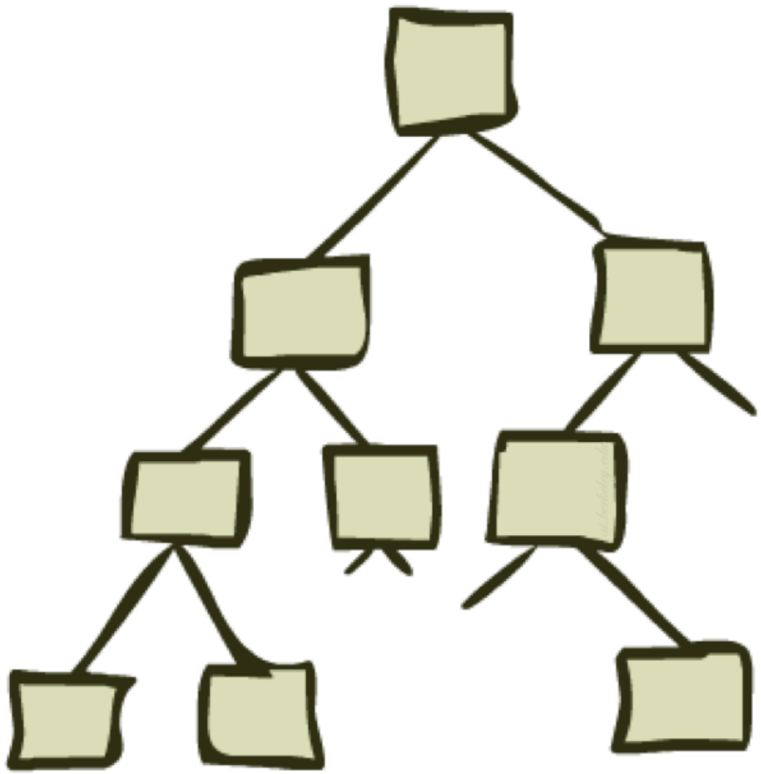


Safe Passage



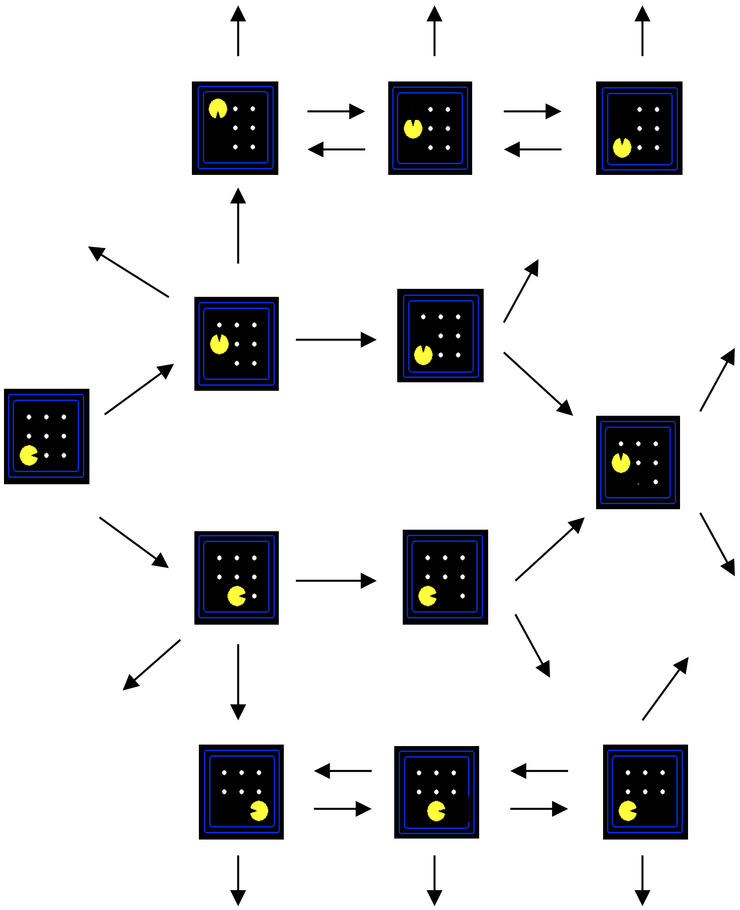
- Problem: eat all dots while keeping the ghosts **perma-scared**
- What does the state space have to specify?
  - (agent position, dot booleans, power dot booleans, remaining scared time)

State Space Graphs and Search Trees

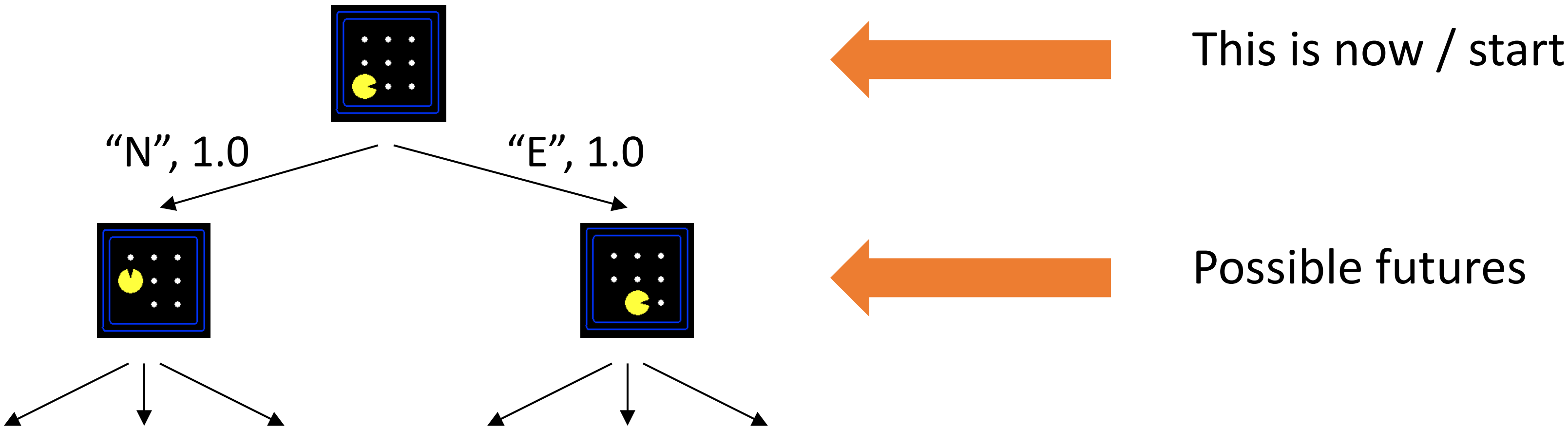


State Space Graph

- State space graph: A mathematical representation of a search problem
  - Nodes are (abstracted) world configurations
  - Arcs represent successors (action results)
  - The goal test is a set of goal nodes (maybe only one)
- In a state space graph, each state occurs only once!
- We can rarely build this full graph in memory (it's too big), but it's a useful idea



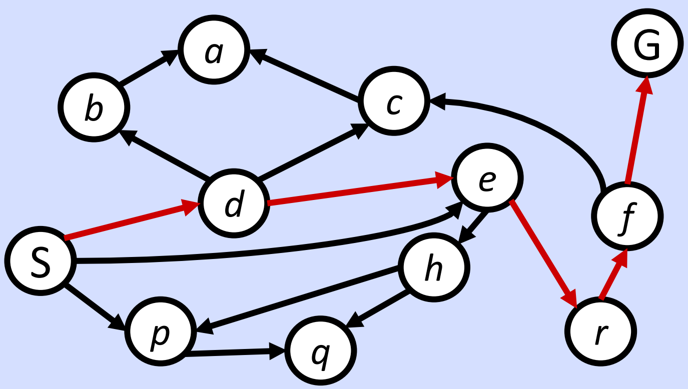
Search Trees



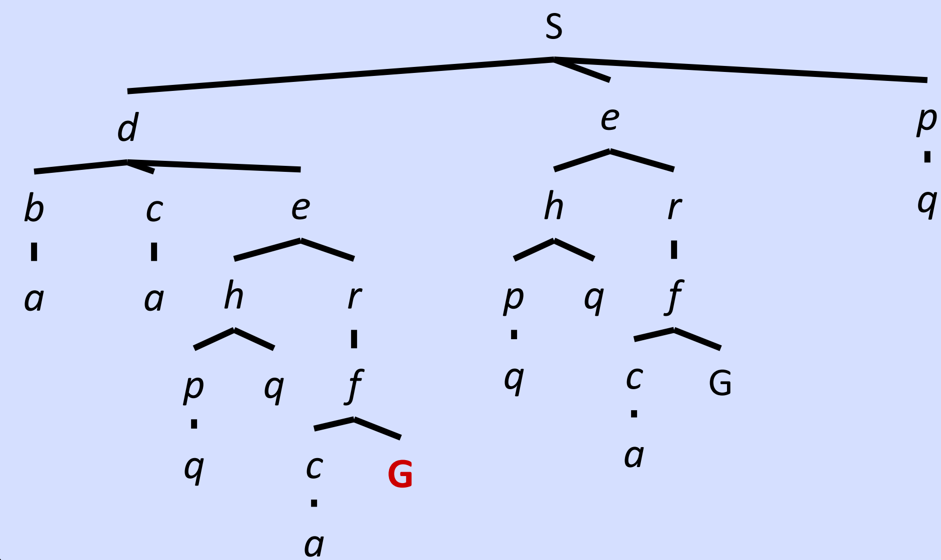
- A search tree:
  - A “what if” tree of plans and their outcomes
  - The start state is the root node
  - Children correspond to successors
  - Nodes show states, but correspond to PLANS that achieve those states
  - For most problems, we can never actually build the whole tree

State Space Graphs vs. Search Trees

State Space Graph



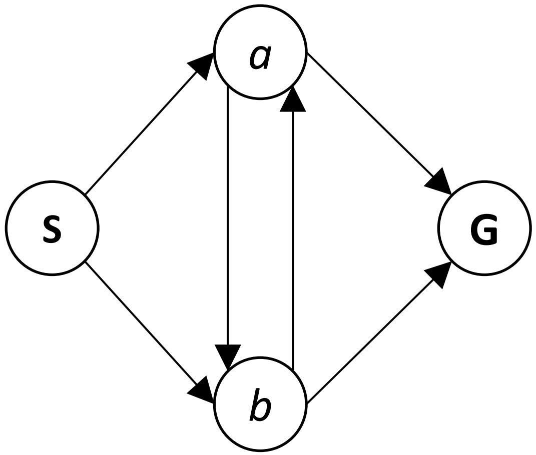
Search Tree



- *Each NODE in the search tree is an entire PATH in the state space graph.*

State Space Graphs vs. Search Trees

Consider this 4-state graph:

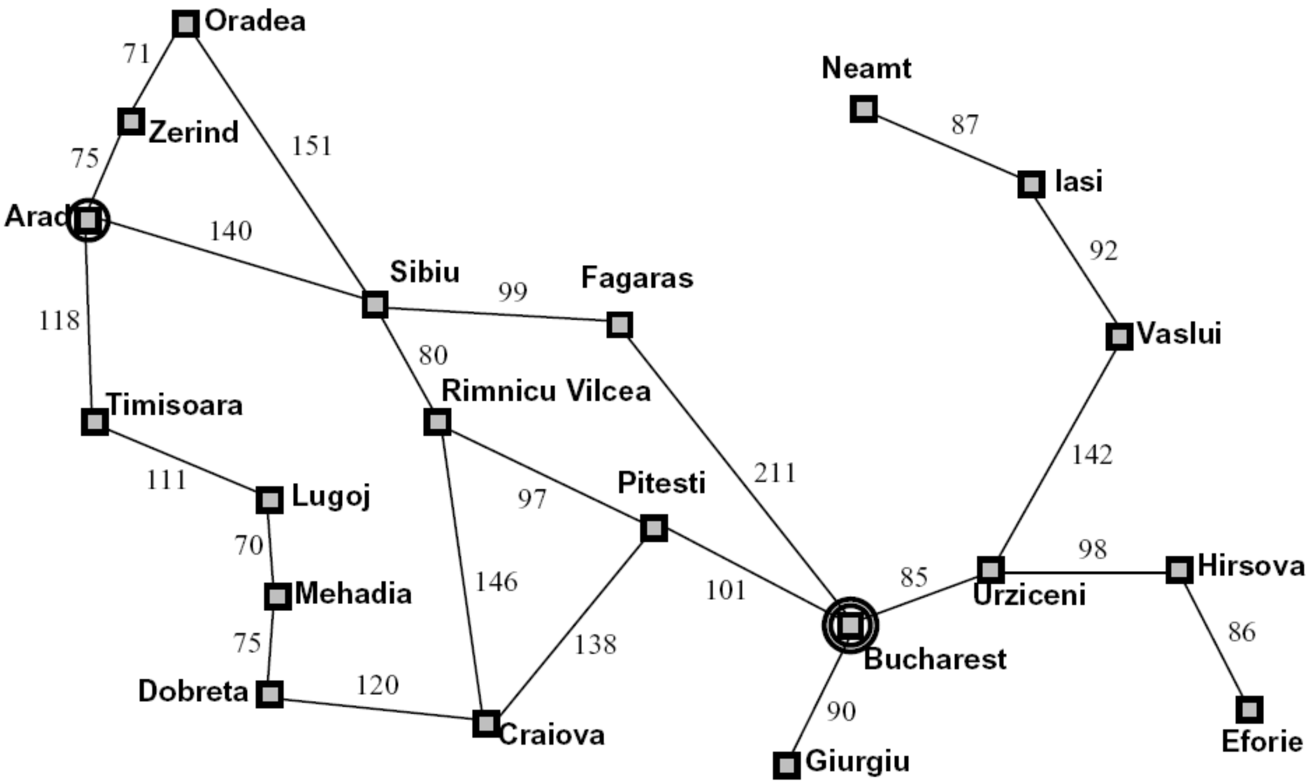


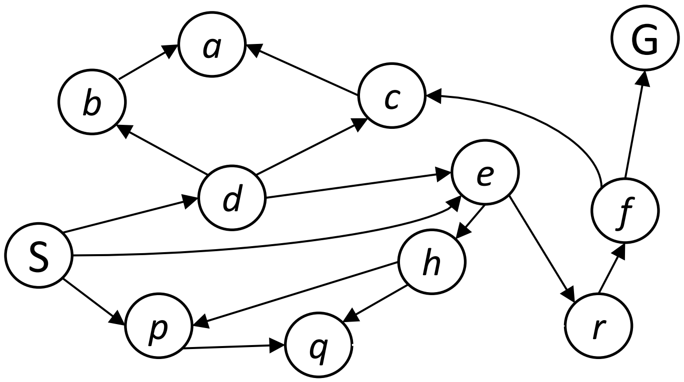
How big is its search tree (from S)?

Important: Lots of repeated structure in the search tree!

- Uninformed Search
  - Depth-First Search
  - Breadth-First Search
  - Uniform-Cost Search

Search Example: Romania

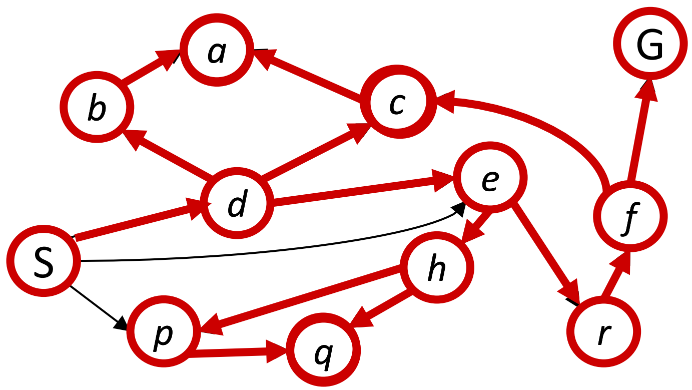


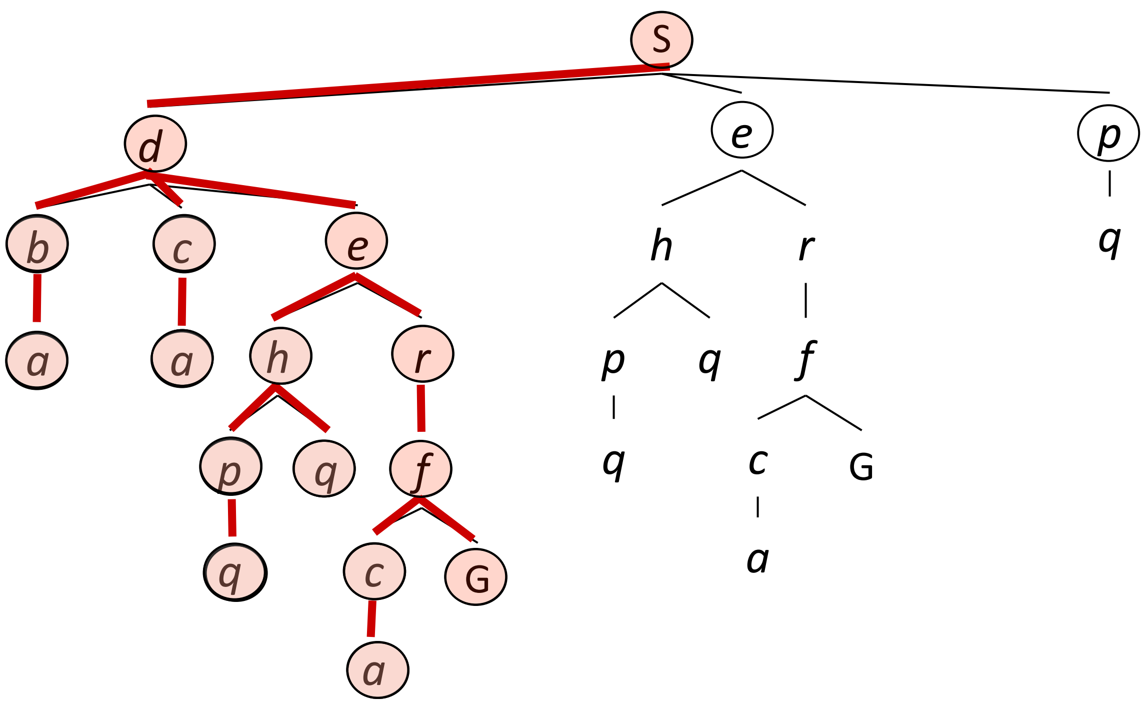


Depth-First Search

Strategy: expand a deepest node first

Implementation: Fringe is a LIFO stack



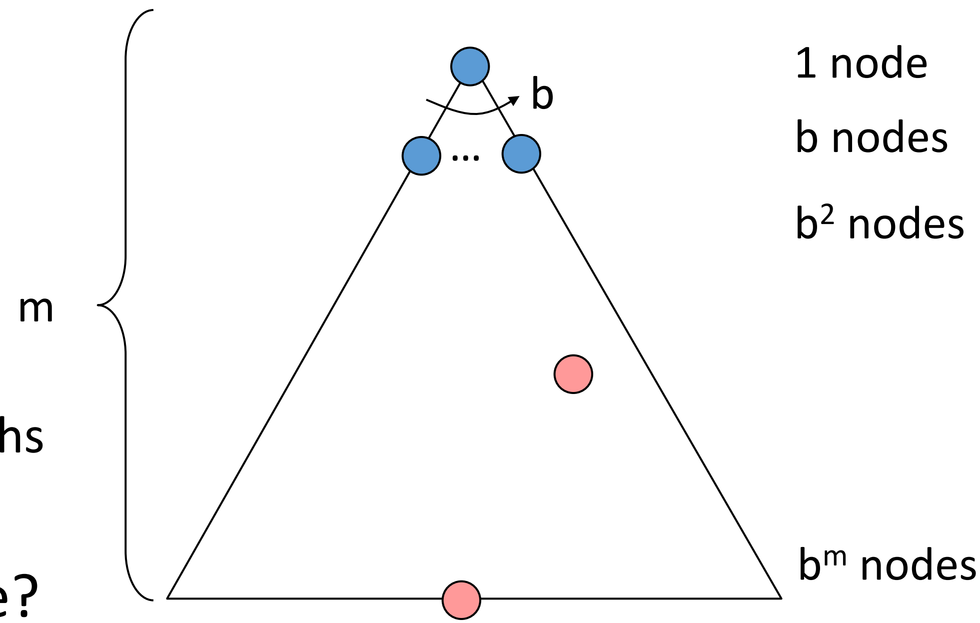


Search Algorithm Properties

- Complete: Guaranteed to find a solution if one exists?
- Optimal: Guaranteed to find the least cost path?
- Time complexity?
- Space complexity?

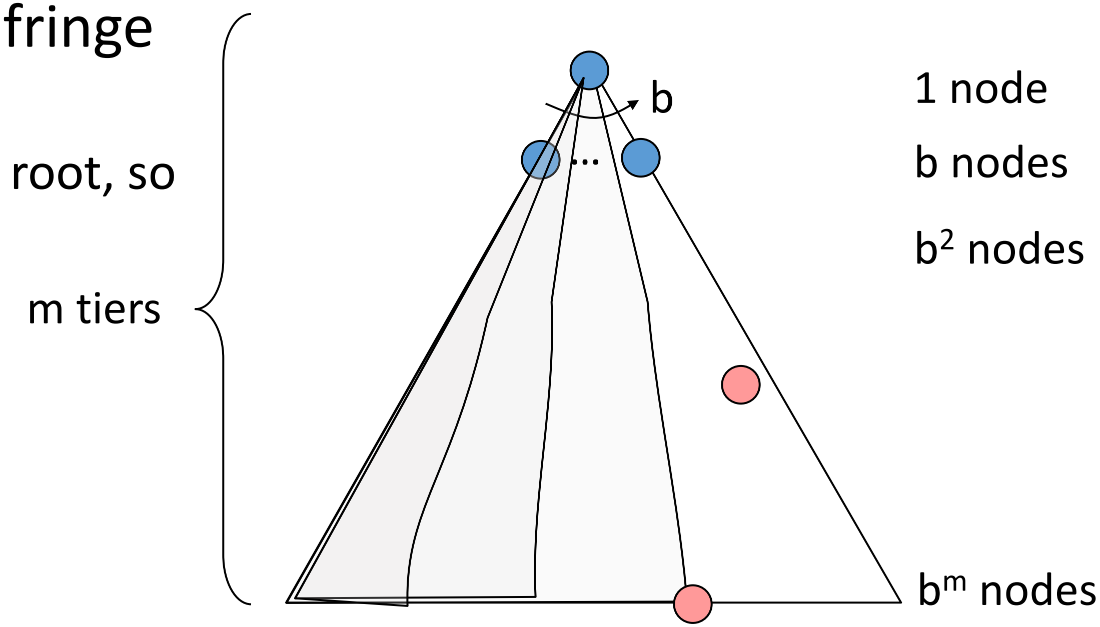
- Cartoon of search tree:
  - b is the branching factor
  - m is the maximum depth
  - s is the solutions at various depths

- Number of nodes in entire tree?
  - $1 + b + b^2 + ... b^m = O(b^m)$



Depth-First Search (DFS) Properties

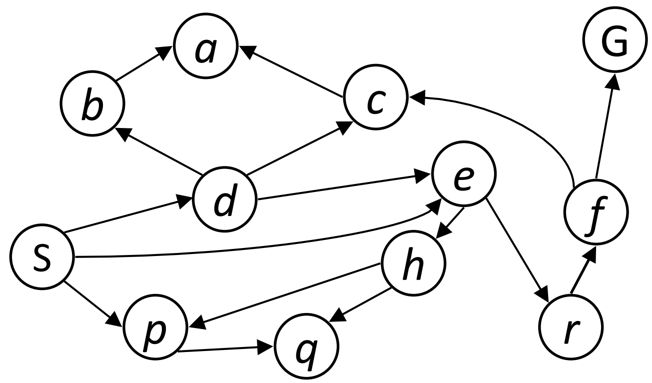
- What nodes DFS expand (Time Complexity)?
  - Some left prefix of the tree.
  - If m is finite, takes time $O(b^m)$
- How much space does the fringe take (Space Complexity)?
  - Only has siblings on path to root, so $O(bm)$
- Is it complete?
  - No, m can be infinite
- Is it optimal?
  - No, it finds the "leftmost" solution, regardless of depth or cost



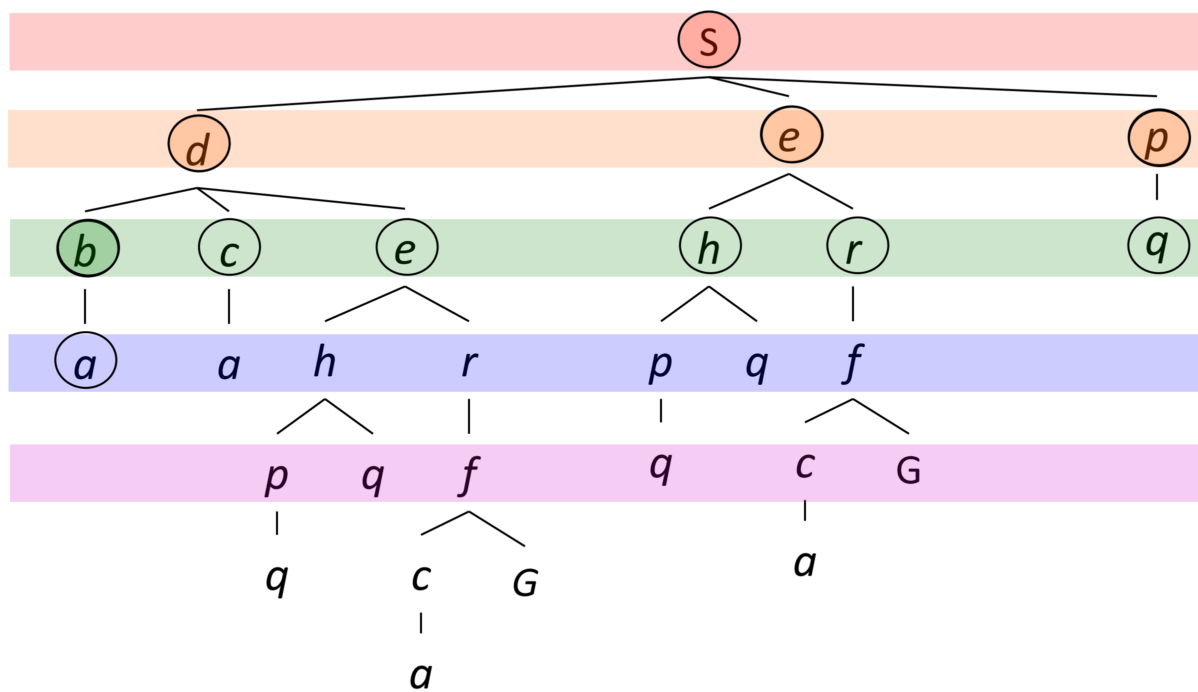
Breadth-First Search

Strategy: expand a shallowest node first

Implementation: Fringe is a FIFO queue



Search Tiers



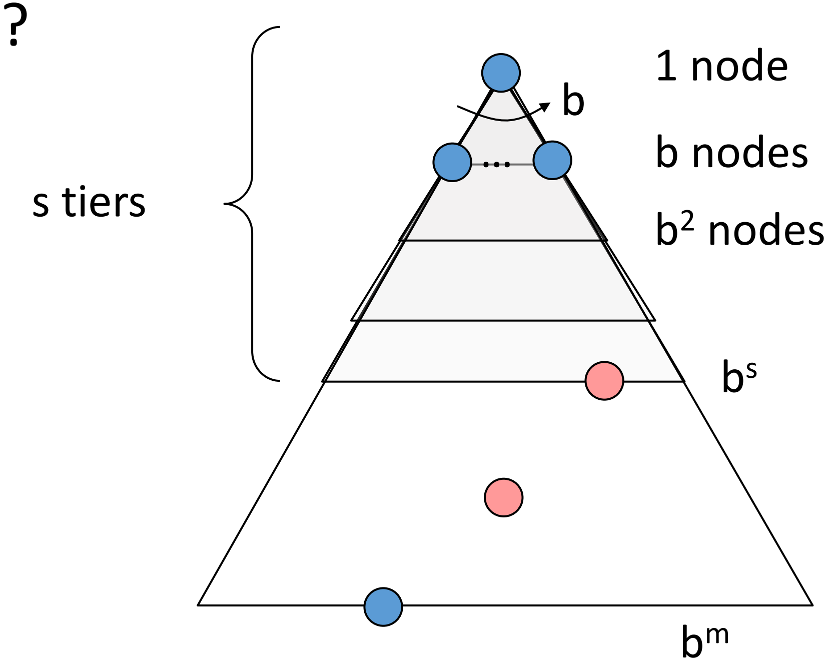
Breadth-First Search (BFS) Properties

- What nodes does BFS expand (Time Complexity)?
  - Processes all nodes above shallowest solution
  - Let depth of shallowest solution be s
  - Search takes time $O(b^s)$

- How much space does the fringe take?
  - Has roughly the last tier, so $O(b^s)$

- Is it complete?
  - yes

- Is it optimal?
  - Yes (if the cost is equal per step)



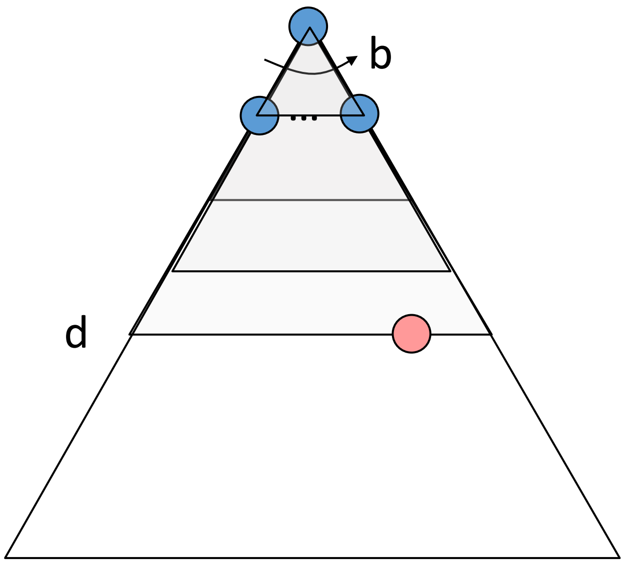
DFS vs BFS

- When will BFS outperform DFS?
  - The branch factor is relatively small.
  - The depth of the optimal solution is relatively shallow.

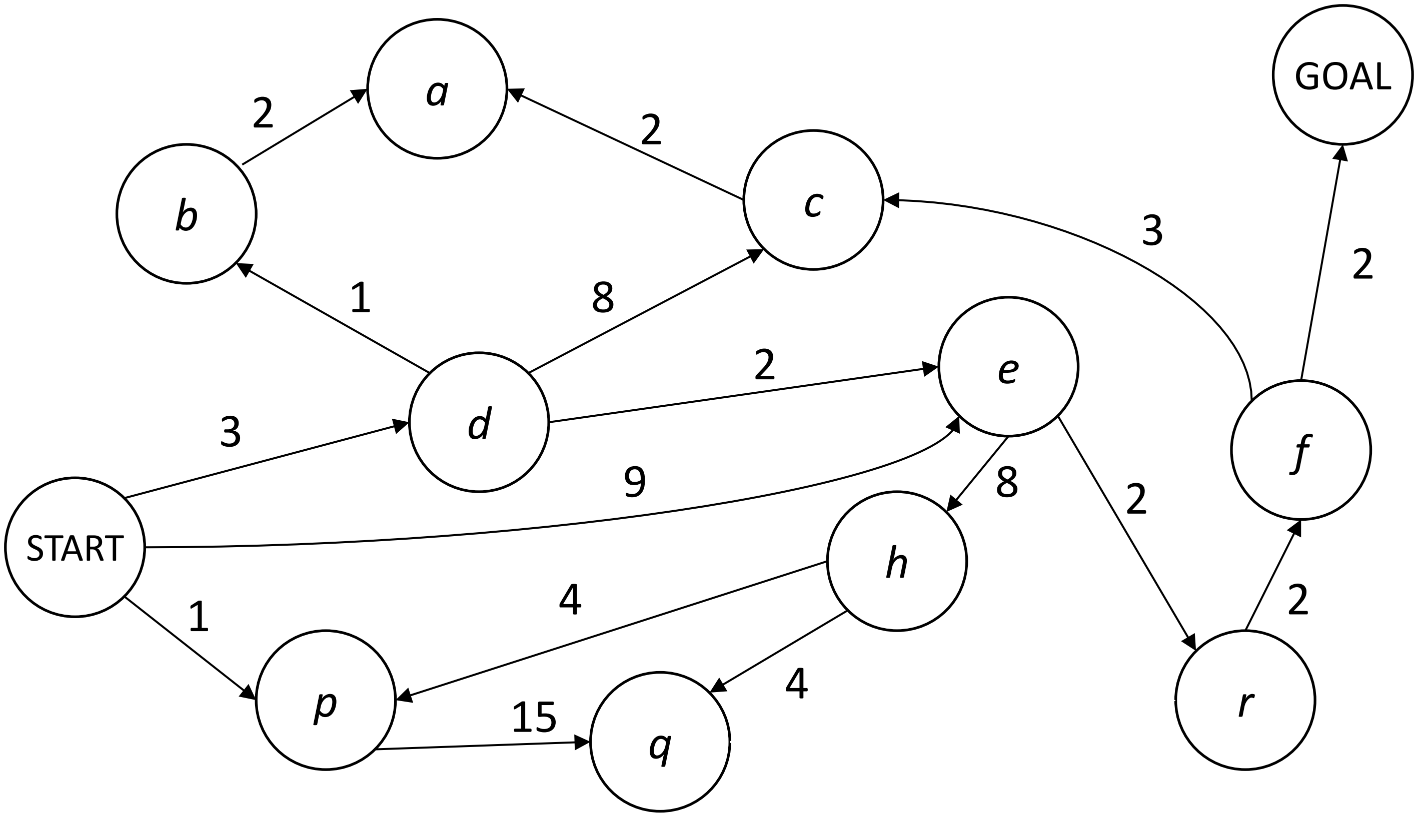
- When will DFS outperform BFS?
  - The tree is deep and the answer is frequent.

Iterative Deepening

- Idea: get DFS's space advantage with BFS's time / shallow-solution advantages
  - Run a DFS with depth limit 1. If no solution…
  - Run a DFS with depth limit 2. If no solution…
  - Run a DFS with depth limit 3. …..
- How many nodes does BFS expand?
  - $O(b^d)$
- How much space does the fringe take?
  - O(bd)
- Is it complete?
  - yes!
- Is it optimal?
  - Yes! (if the cost is equal per step)



Cost-Sensitive Search

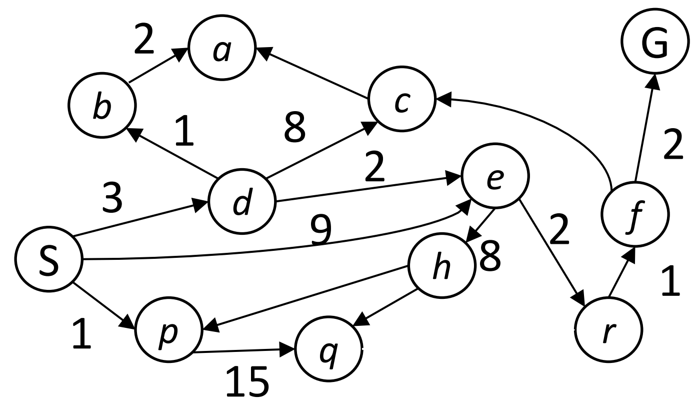


BFS finds the shortest path in terms of number of actions. It does not find the least-cost path. We will now cover a similar algorithm which does find the least-cost path.

Uniform Cost Search

Strategy: expand a cheapest node first:

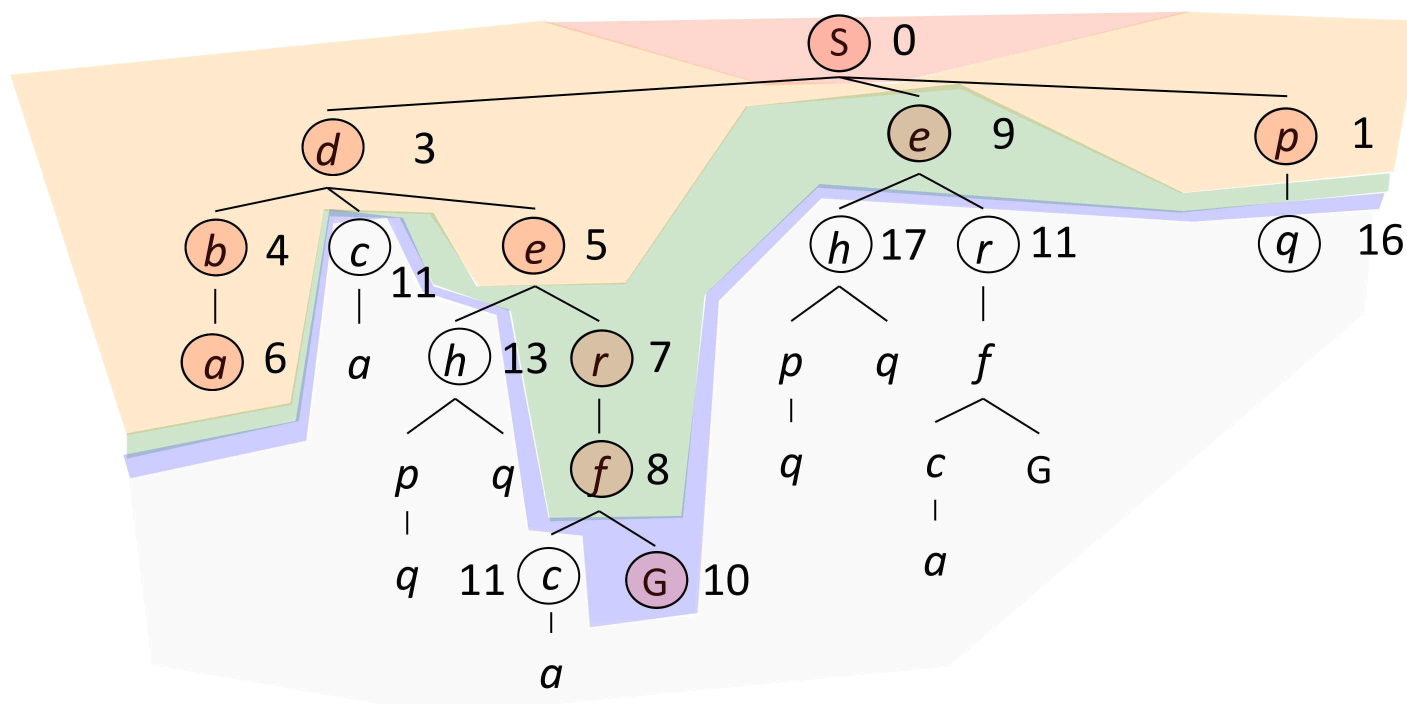
Fringe is a priority queue (priority: cumulative cost)



Cost contours

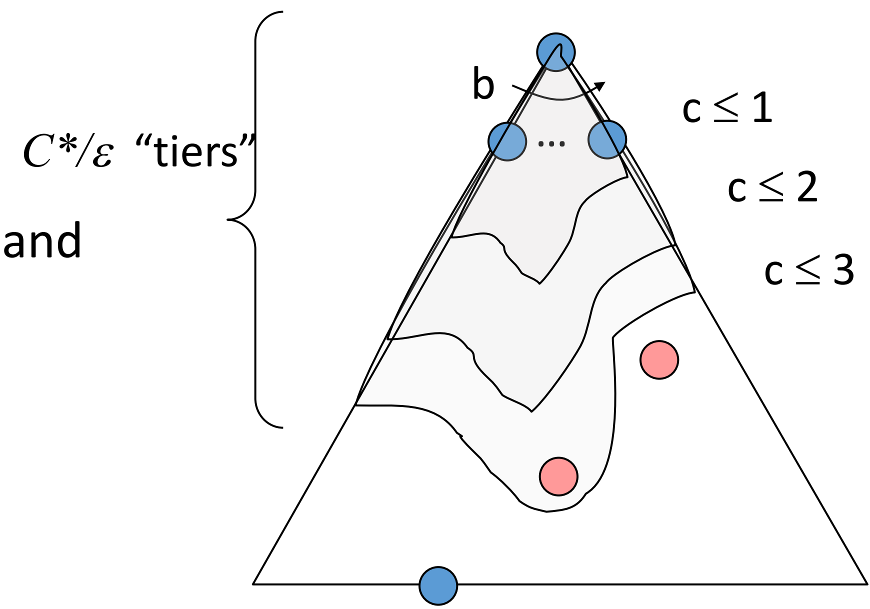
Suppose

$C^* = 10$ and ε equals 3



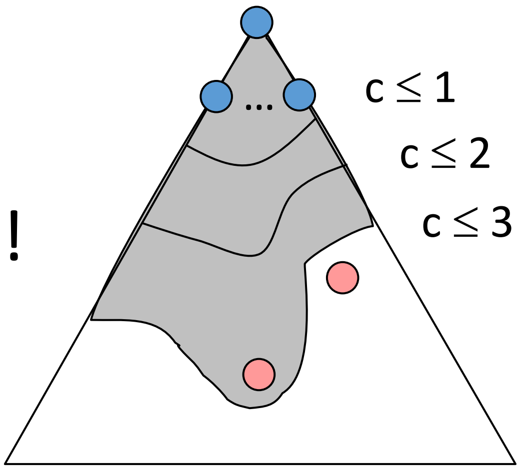
Uniform Cost Search (UCS) Properties

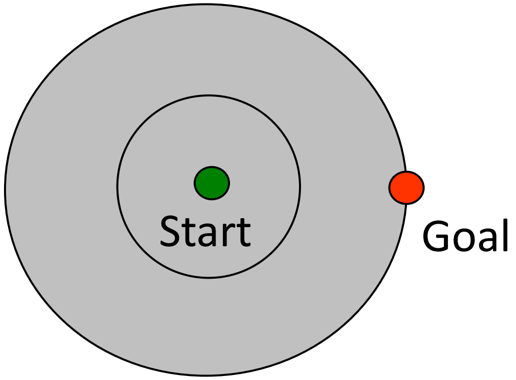
- What nodes does UCS expand?
  - Processes all nodes with cost less than cheapest solution!
  - If that solution costs C^* and arcs cost at least ε, then the "effective depth" is roughly C^*/ε
  - Takes time O(b$^{C^*/\varepsilon}$) (exponential in effective depth)
- How much space does the fringe take?
  - Has roughly the last tier, so O(b$^{C^*/\varepsilon}$)
- Is it complete?
  - Assuming best solution has a finite cost and minimum arc cost is positive, yes!
- Is it optimal?
  - Yes!



Uniform Cost Issues

- Remember: UCS explores increasing cost contours
- The good: UCS is complete and optimal!
- The bad:
  - Explores options in every “direction”
  - No information about goal location





Comparison

Algorithm	Complete?	Optimal?	Time?	Space?
DFS	N	N	O(b^{m})	O(bm)
BFS	Y	Y	O(b^{d})	O(b^{d})
IDS	Y	Y	O(b^{d})	O(bd)
UCS	Y	Y	O(b$^{C*/\varepsilon}$)	O(b$^{C*/\varepsilon}$)

For BFS, Suppose the branching factor *b* is finite and step costs are identical;

How bad is BFS?

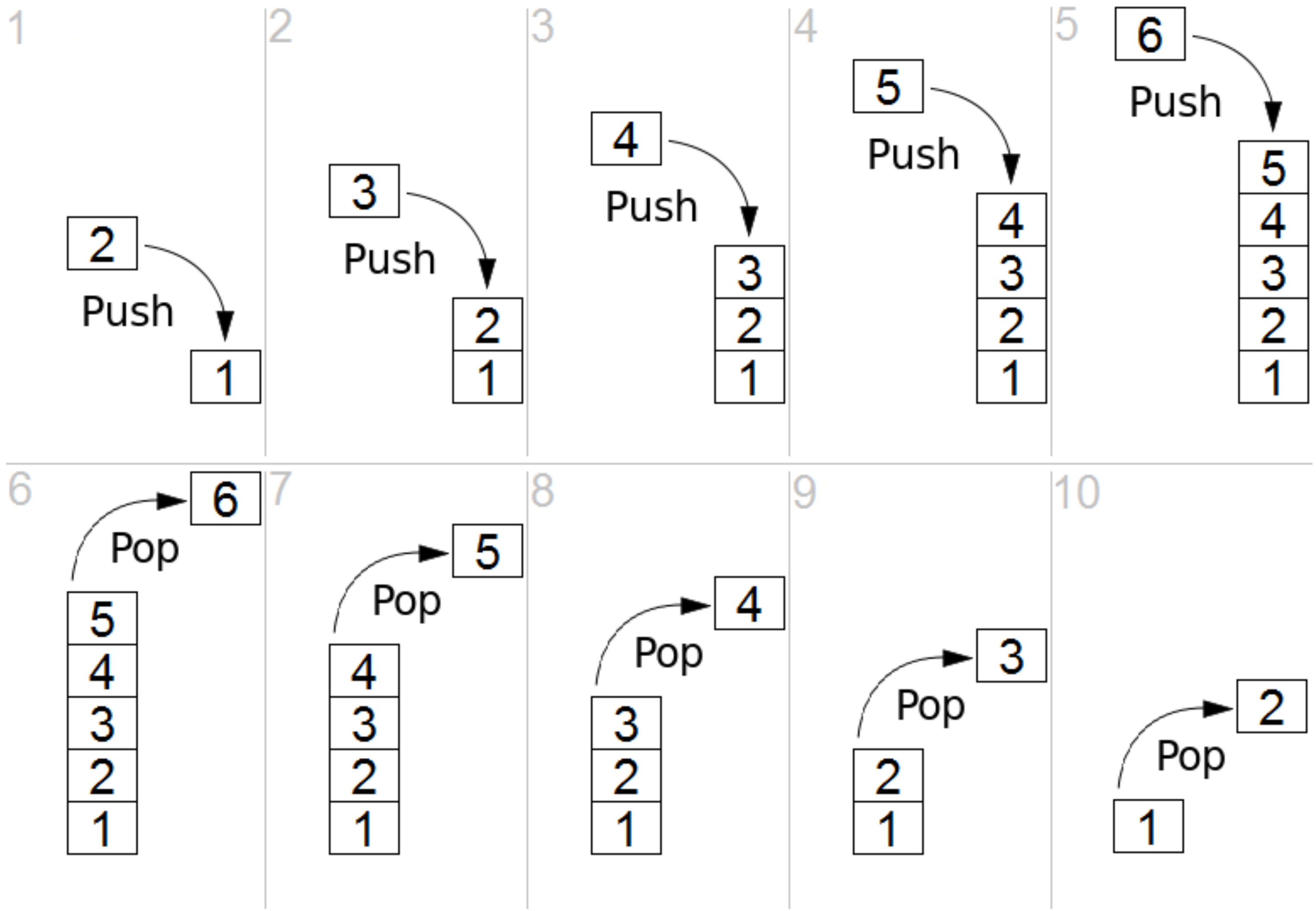
Depth	Nodes	Time	Memory
2	110	.11 milliseconds	107 kilobytes
4	11,110	11 milliseconds	10.6 megabytes
6	10^6	1.1 seconds	1 gigabyte
8	10^8	2 minutes	103 gigabytes
10	10^{10}	3 hours	10 terabytes
12	10^{12}	13 days	1 petabyte
14	10^{14}	3.5 years	99 petabytes
16	10^{16}	350 years	10 exabytes

Figure 3.13 Time and memory requirements for breadth-first search. The numbers shown assume branching factor $b = 10$; 1 million nodes/second; 1000 bytes/node.

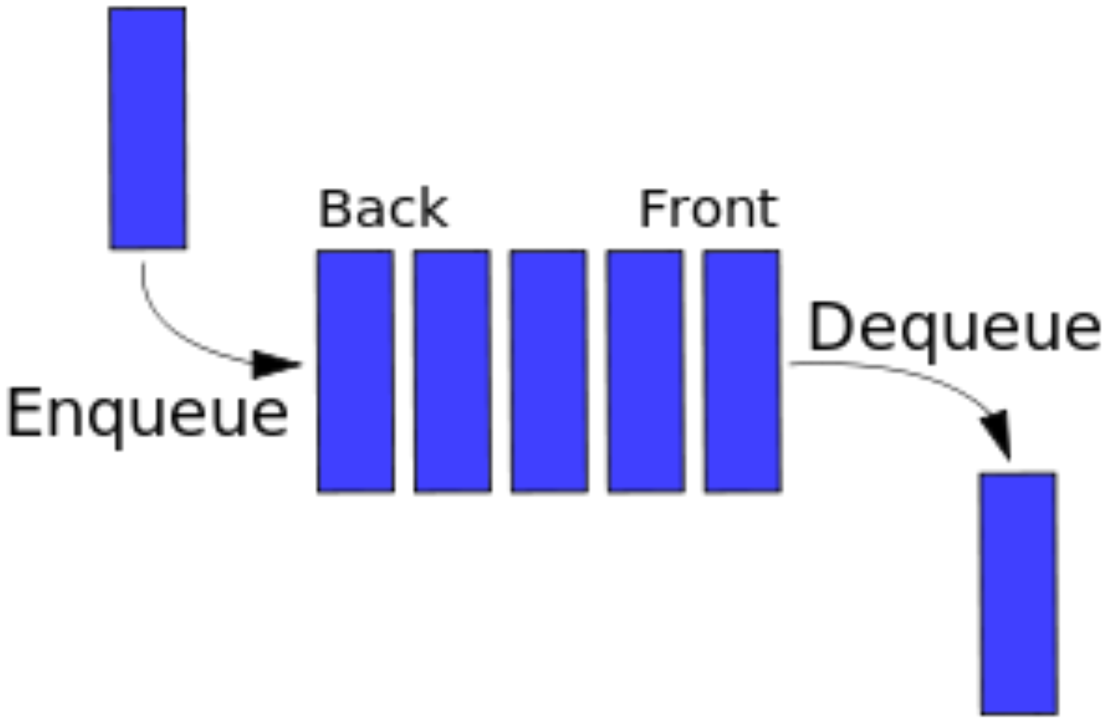
Data structure

- LIFO stack
- FIFO queue
- Priority queue

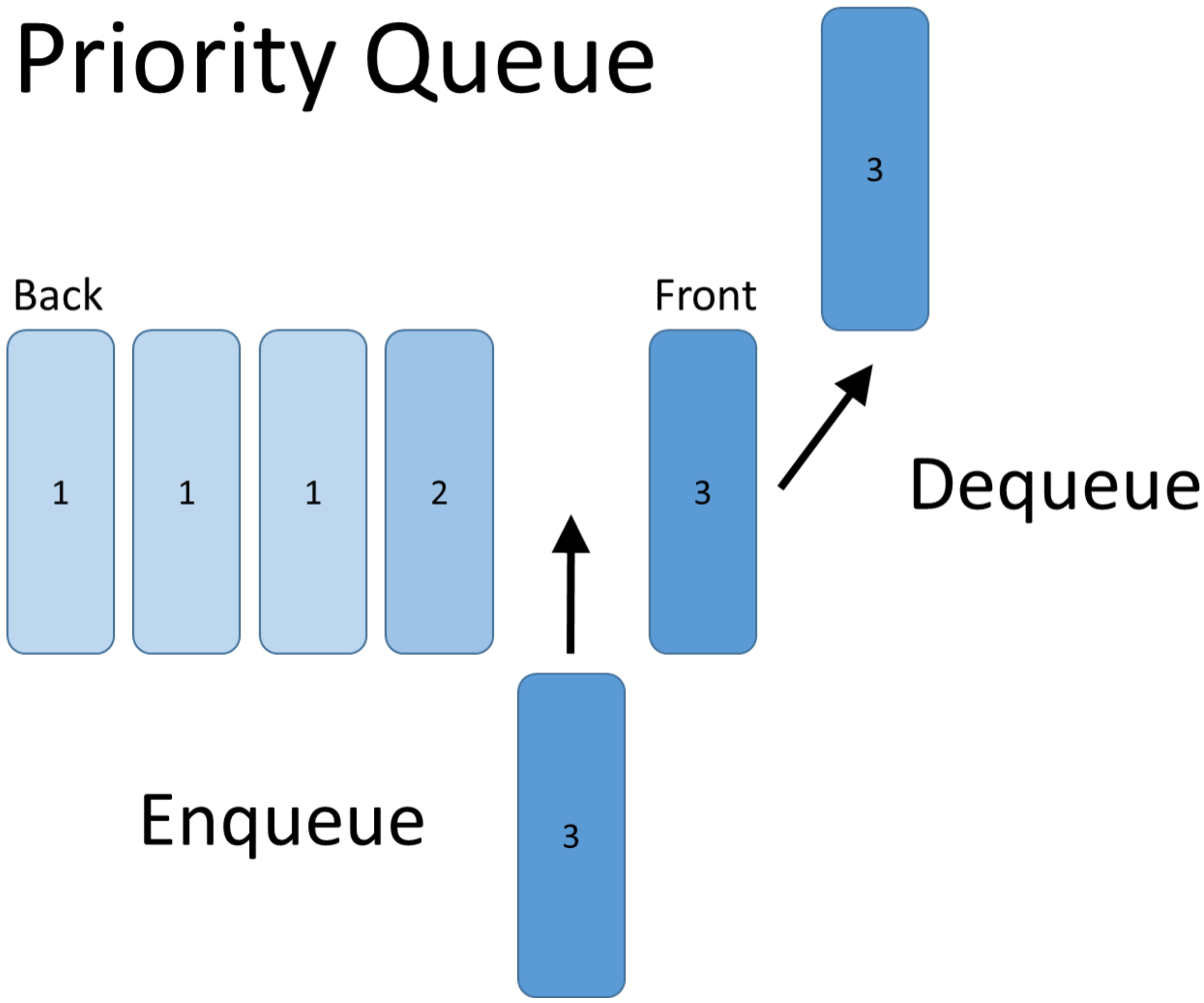
Last in First out Stack



FIFO queue



Priority queue



The One Queue

- All these search algorithms are the same except for fringe strategies
  - Conceptually, all fringes are priority queues (i.e. collections of nodes with attached priorities)
  - Can even code one implementation that takes a variable queuing object