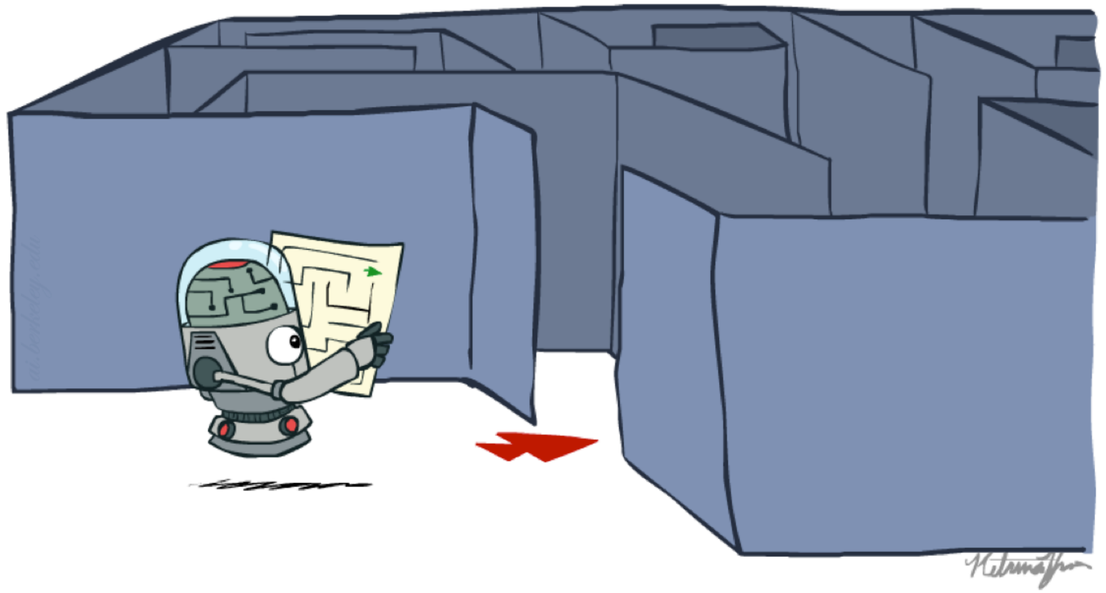
PURDUE CS47100

INTRODUCTION TO AI

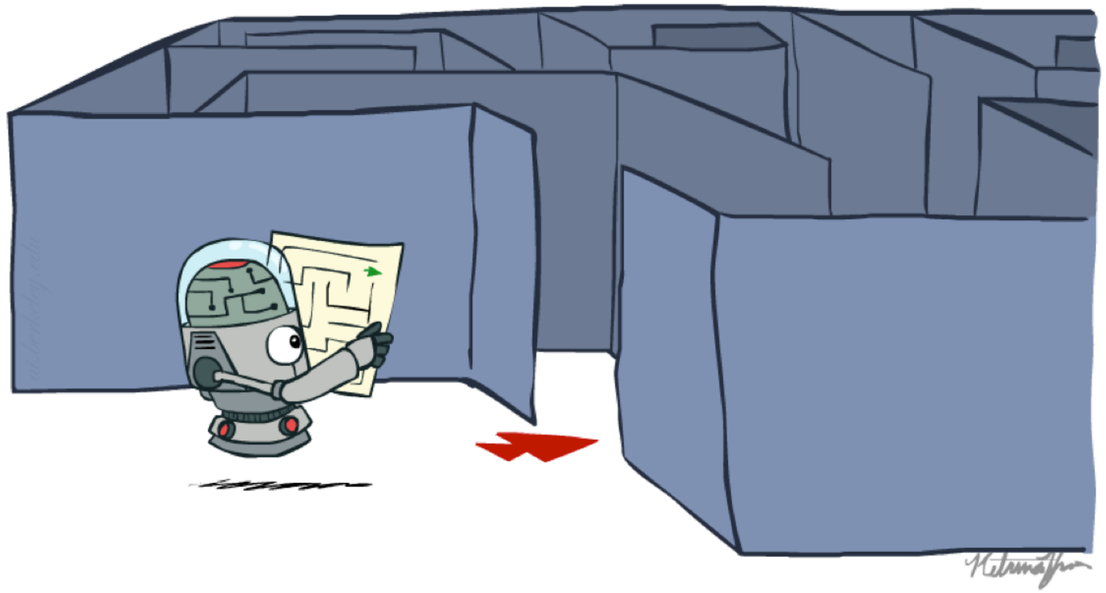
RECAP: SEARCH

- Search problem:
  - States and initial state
  - Actions
  - Transition model
  - Goal test
  - Step cost and path cost

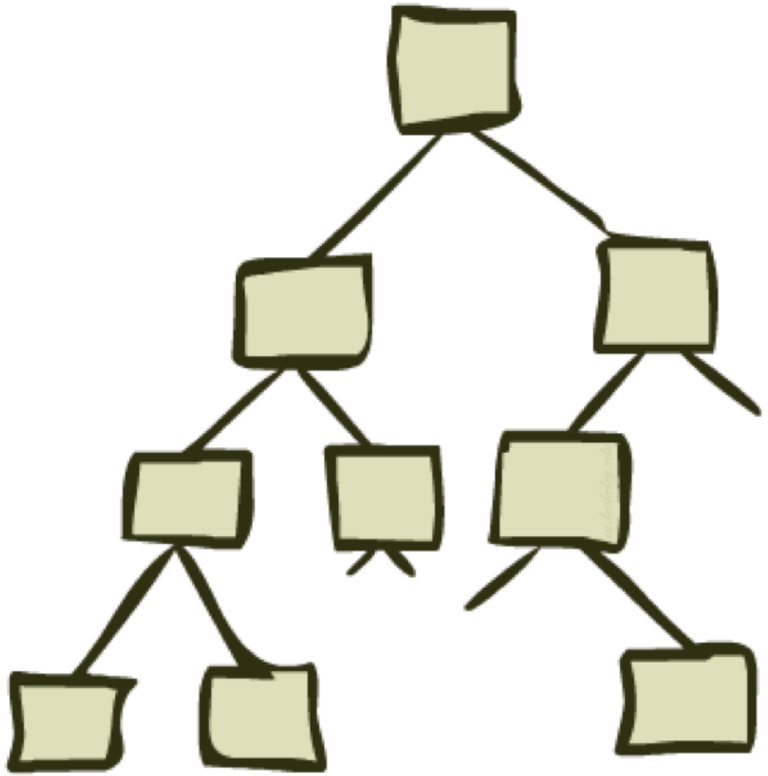


RECAP: SEARCH

- A search state should contain:
  - All the information needed to find a solution
  - Nothing but this information
- Watch out for efficiency: Try to define states and actions in a way that decrease the size of the state space as much as possible

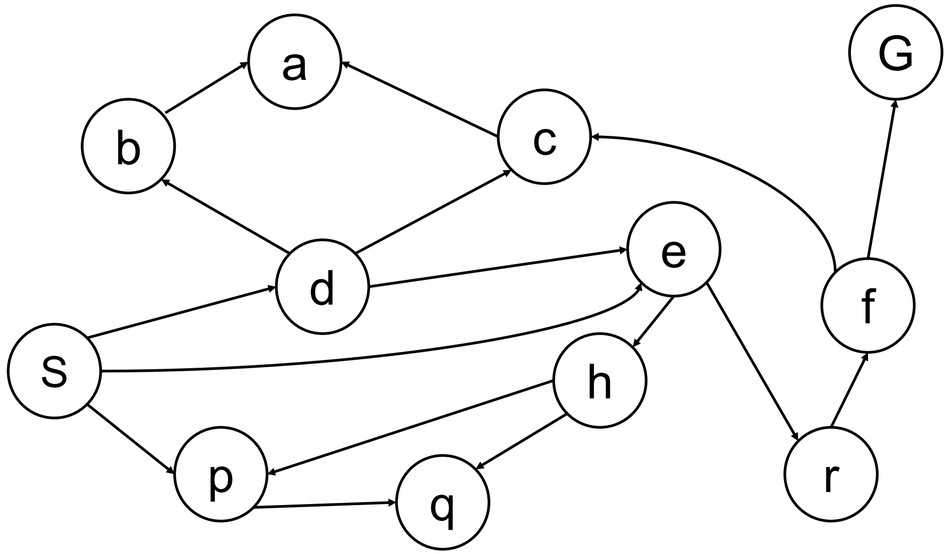


STATE SPACE GRAPHS AND SEARCH TREES



STATE SPACE GRAPHS

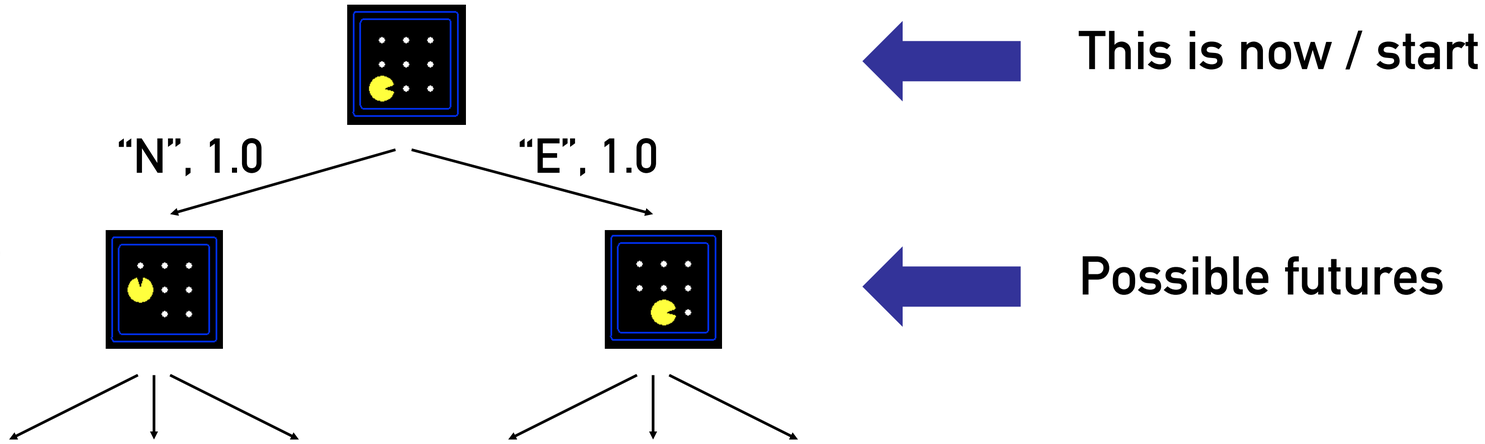
- **State space graph**: A mathematical representation of a search problem
  - Nodes are (abstracted) world configurations
  - Edges represent successors (results of actions)
  - The goal test is a set of goal nodes (maybe only one)
- In a state space graph, each state occurs only once
- We can rarely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



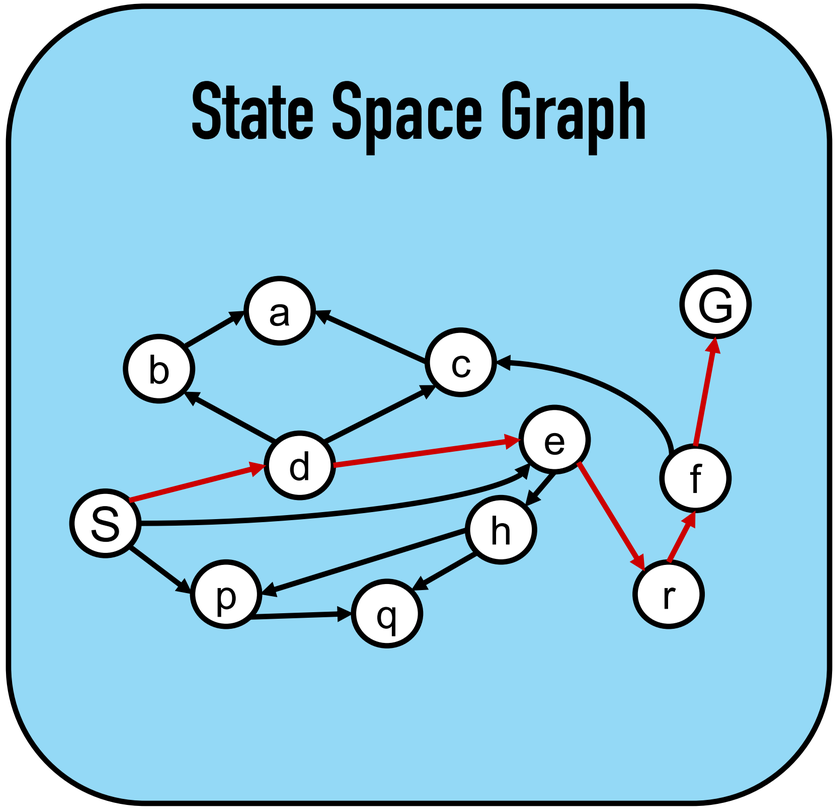
Example search graph

SEARCH TREES

- A search tree:
  - A "what if" tree of actions and their outcomes
  - Root node = initial state
  - Children correspond to successors
  - Nodes show states, branches correspond to actions that lead to those states
  - For most problems, we can never actually build the whole tree

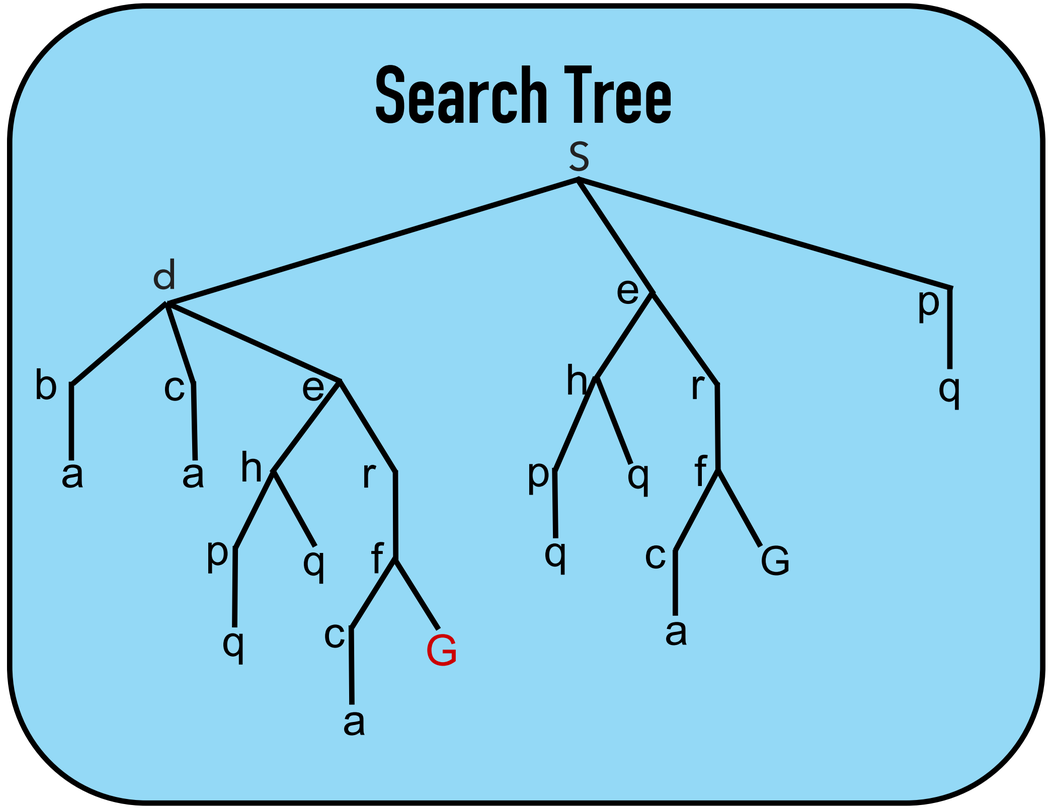


STATE SPACE GRAPHS VS. SEARCH TREES

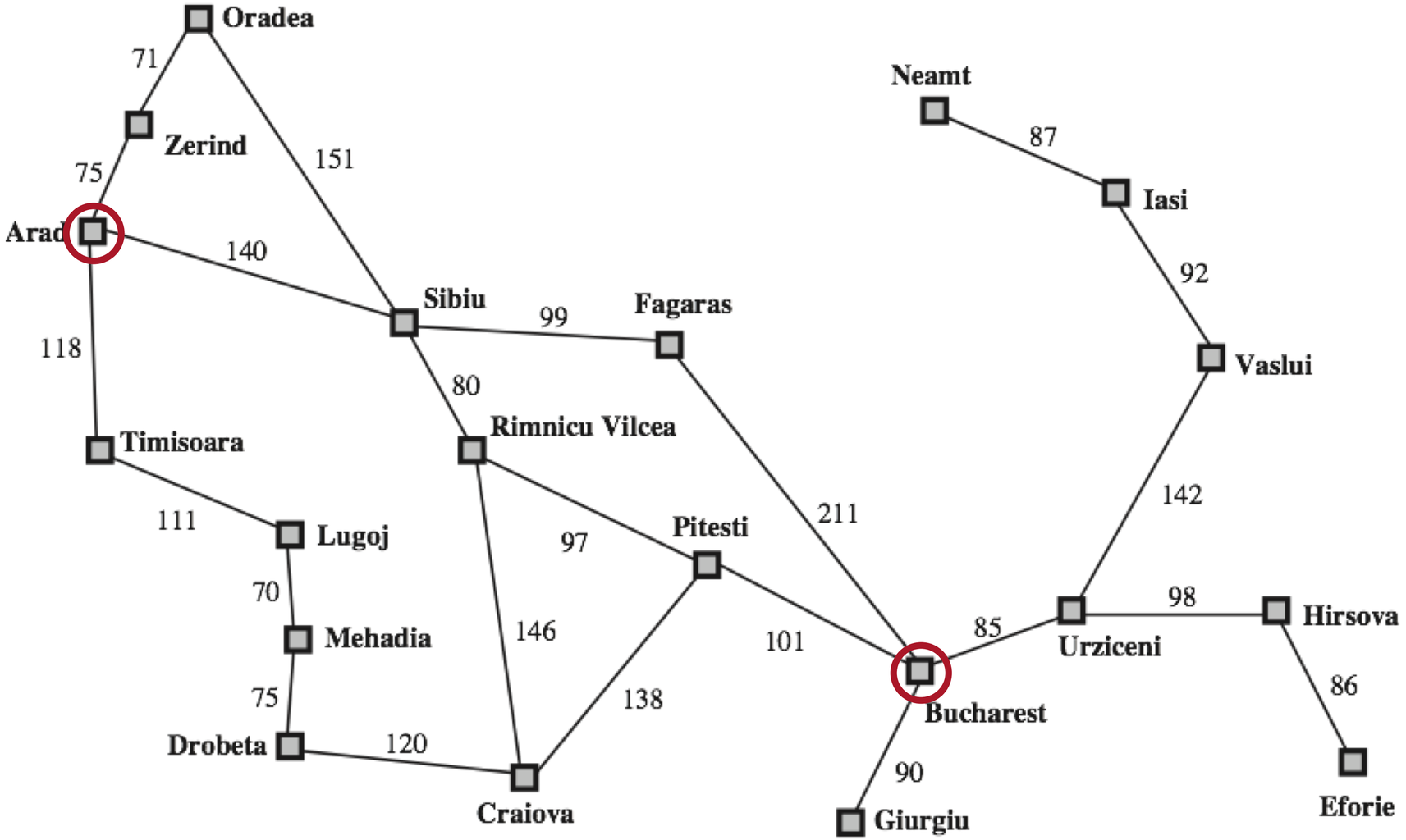


Each NODE in the search tree is reached by a PATH in the state space graph.

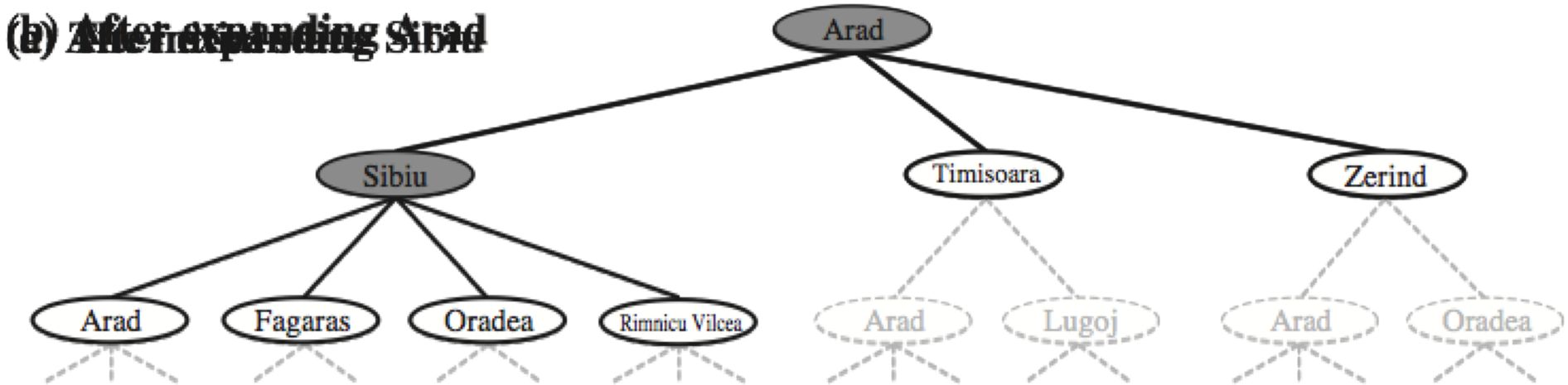
Algorithms construct both on demand – and they construct as little as possible.



SEARCH EXAMPLE: ROMANIA



SEARCHING WITH A SEARCH TREE



- Search:
  - Expand out potential plans (tree nodes)
  - Maintain a **fringe (frontier)** of partial plans under consideration
  - Try to expand as few tree nodes as possible

TREE SEARCH ALGORITHMS

```
function TREE_SEARCH (problem) return a solution, or failure

    fringe ← {initial_state}

    repeat

        if fringe = ∅ then return failure

        node ← POP(fringe)

        if GOAL_TEST(node)  then return the corresponding solution

        for a in Action(node):

            a' = Result(node, a)

            INSERT(fringe, a')
```

Search algorithms differ in how they pick the next node to expand from the fringe

HOW DO YOU EVALUATE A SEARCH ALGORITHM?

- Completeness – Does it always find a solution if one exists?
- Optimality – Does it find the best solution?
- Time complexity – How many nodes are processed (i.e., have ever been added to the fringe) to find a solution?
- Space complexity – How many nodes need to be stored in the fringe?

UNINFORMED SEARCH STRATEGIES

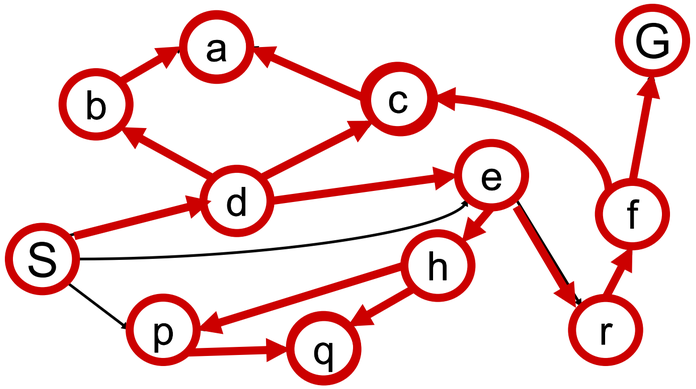
- The search algorithm has no additional information about states beyond what has been provided in the problem formulation
  - Depth-first search
  - Breadth-first search
  - Depth-limited search
  - Iterative deepening depth-first search
  - Bi-directional search
  - Uniform-cost search

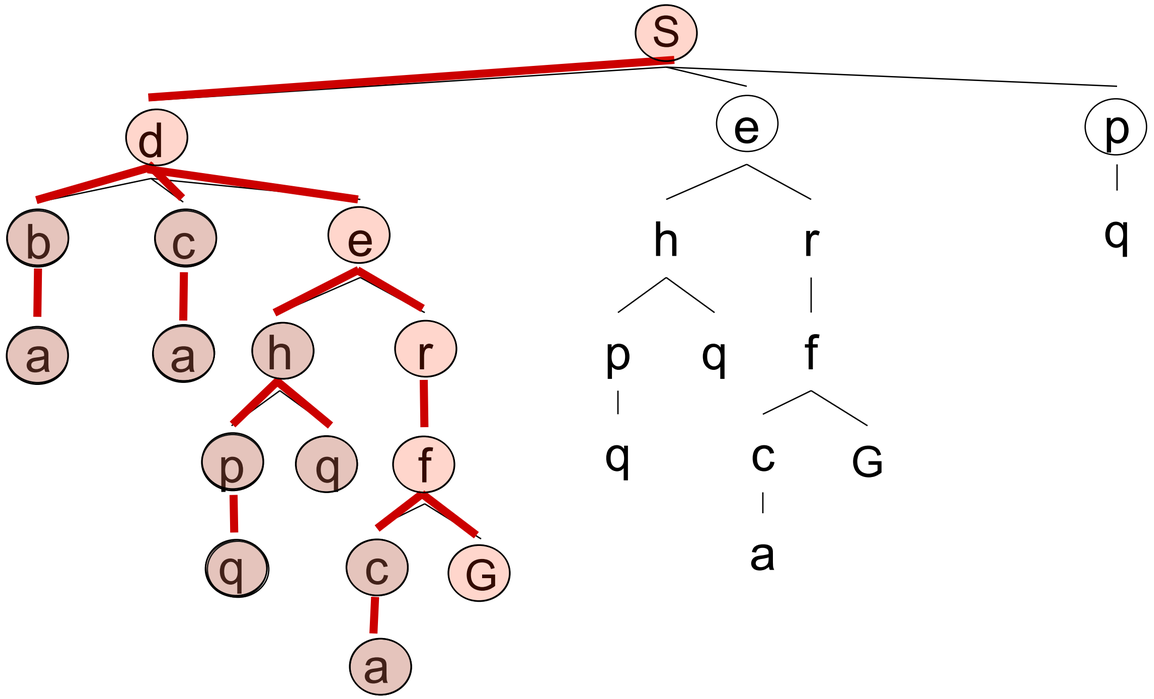
DEPTH-FIRST SEARCH

DEPTH-FIRST SEARCH

Strategy: expand a deepest node first

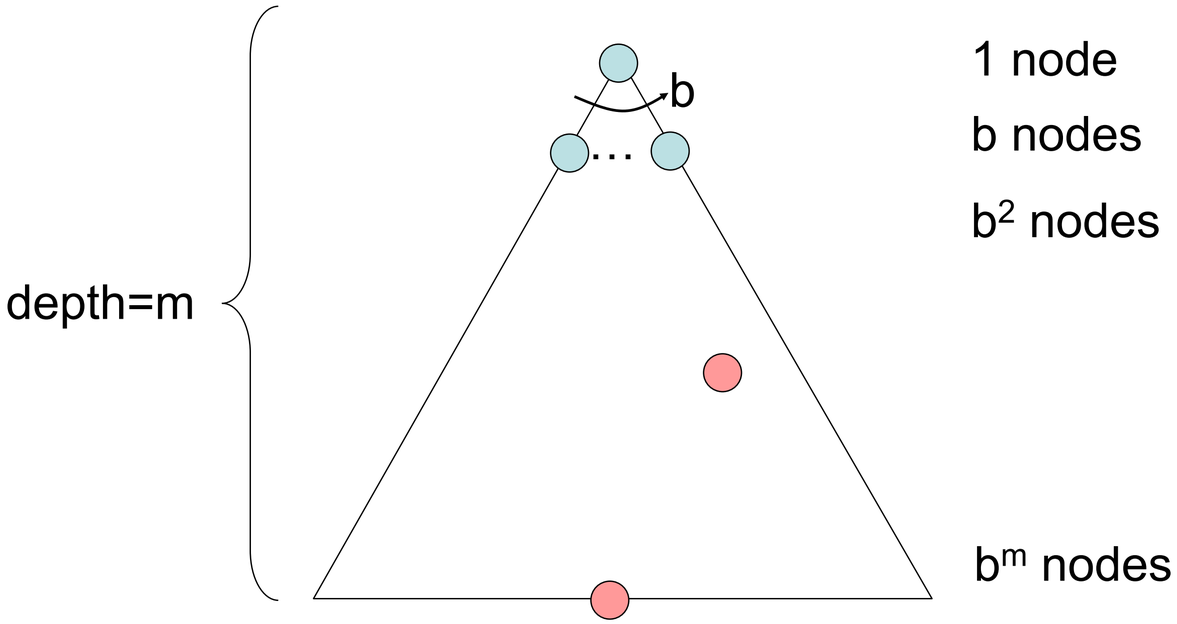
Implementation:
Fringe is a LIFO stack





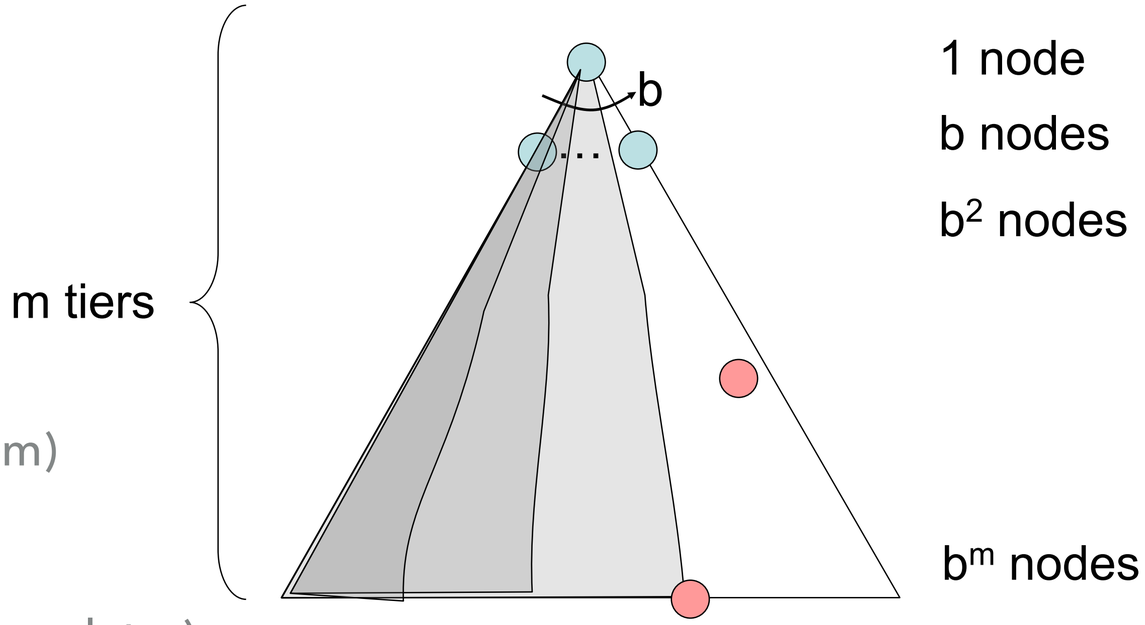
SEARCH ALGORITHM PROPERTIES

- Abstraction of search tree:
  - *b* is the branching factor
  - *m* is the maximum depth
  - solutions at various depths, let *d* be depth of shallowest goal
- Number of nodes in entire tree?
  - $1 + b + b^2 + \ldots b^m = O(b^m)$

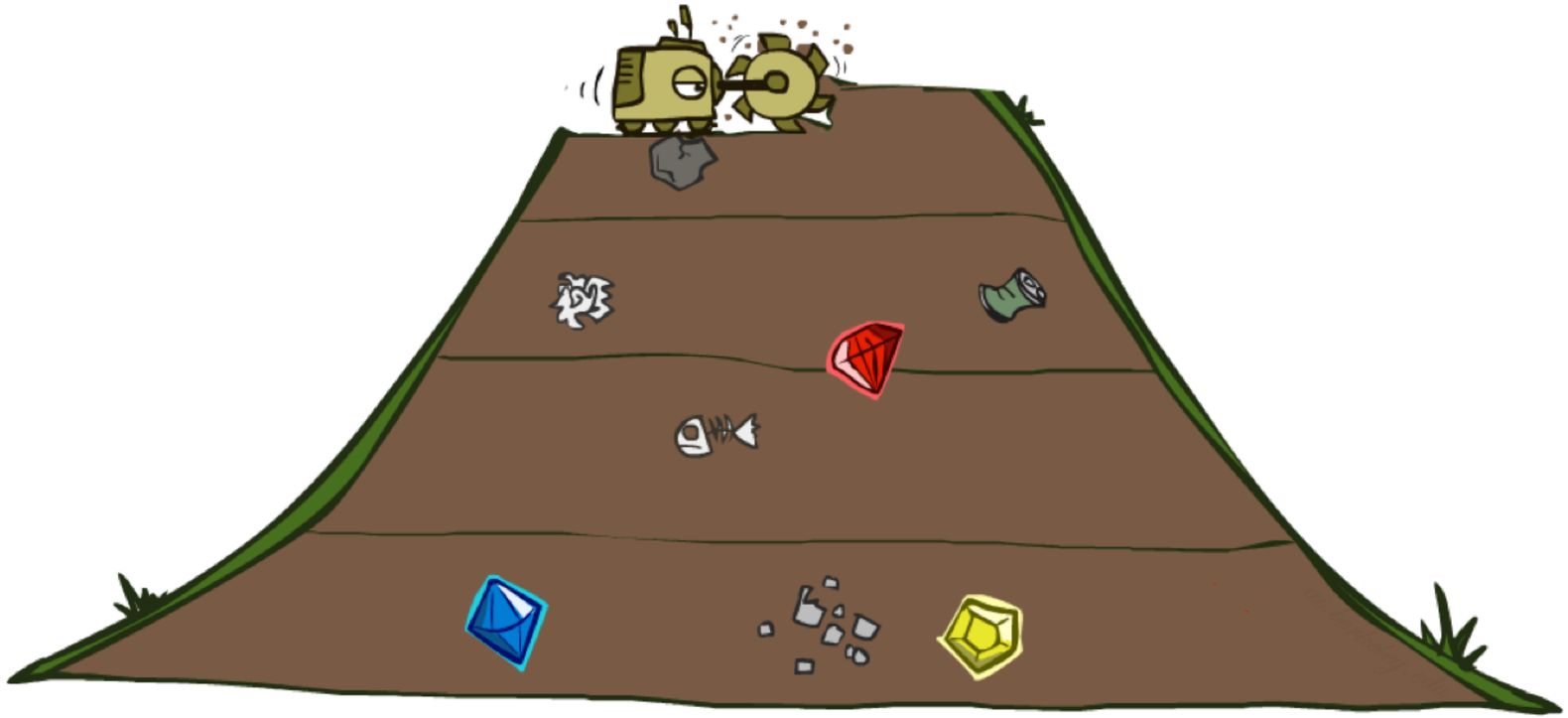


DEPTH-FIRST SEARCH (DFS) PROPERTIES

- Time complexity
  - DFS could process the whole tree (and m > d)
  - If m is finite, takes time $O(b^m)$
- Space complexity
  - Only need to store siblings on path to root, so O(bm)
- Is it complete?
  - m could be infinite, so only if we prevent cycles (more later)
- Is it optimal?
  - No, it finds the "leftmost" solution, regardless of depth or cost



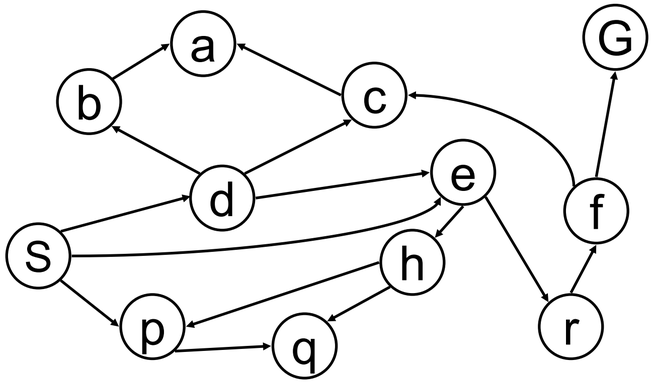
BREADTH-FIRST SEARCH



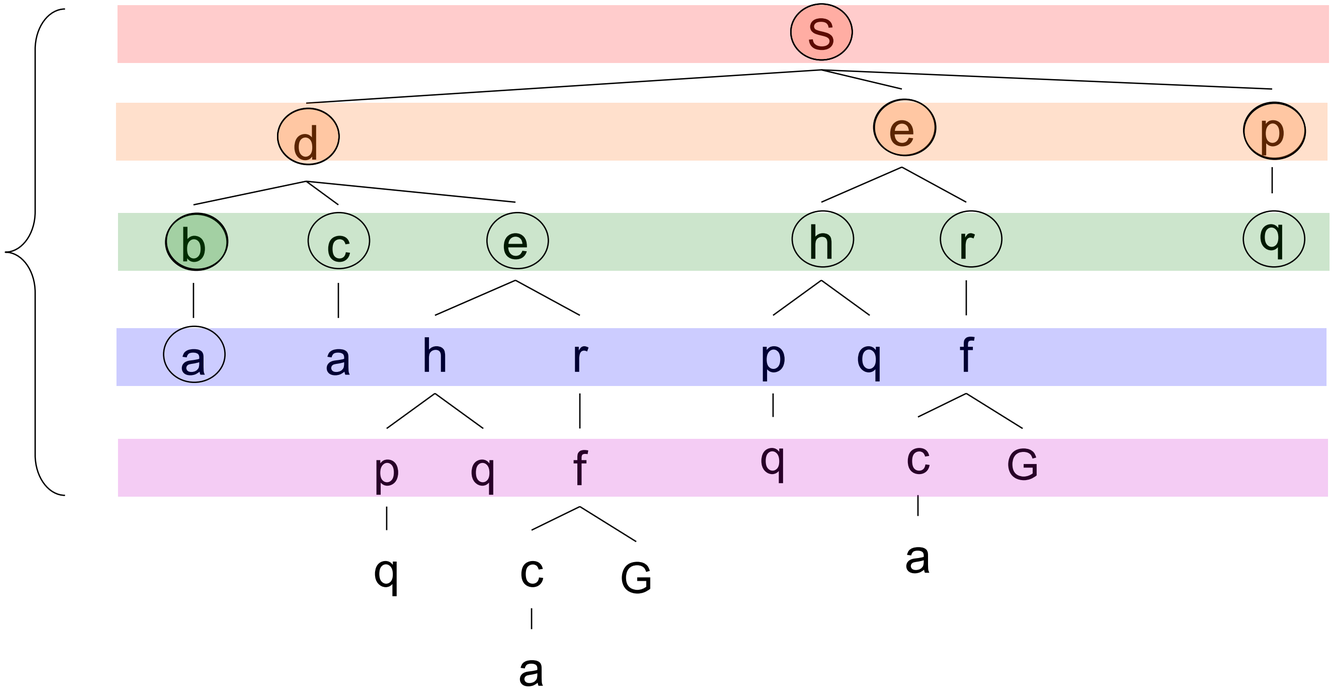
BREADTH-FIRST SEARCH

Strategy: expand a shallowest node first

Implementation: Fringe is a FIFO queue

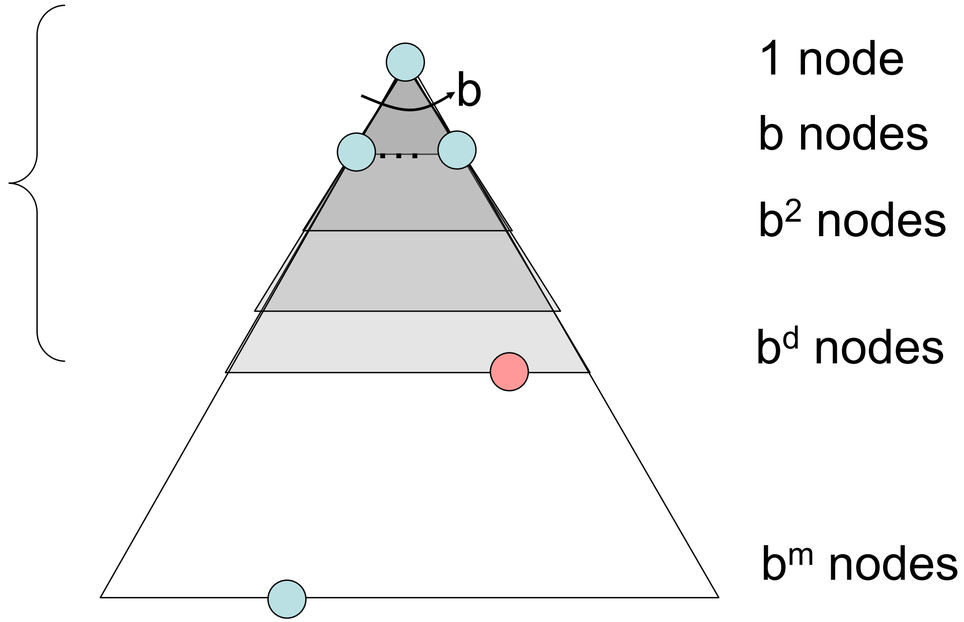


Search Tiers



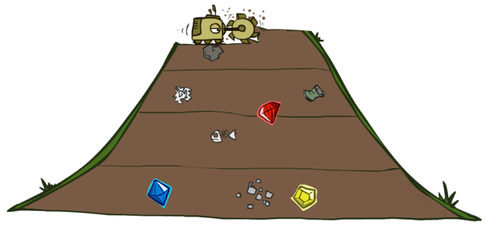
BREADTH-FIRST SEARCH (BFS) PROPERTIES

- Time complexity
  - BFS expands all nodes above shallowest solution
  - Let depth of shallowest solution be d
  - Search takes time $O(b^{d+1})$
- Space complexity
  - Needs to store roughly the successors of the last tier, so $O(b^{d+1})$
- Is it complete?
  - Yes, d must be finite if a solution exists
- Is it optimal?
  - Only if costs are all 1 (more on costs later)



DFS VS BFS

VS

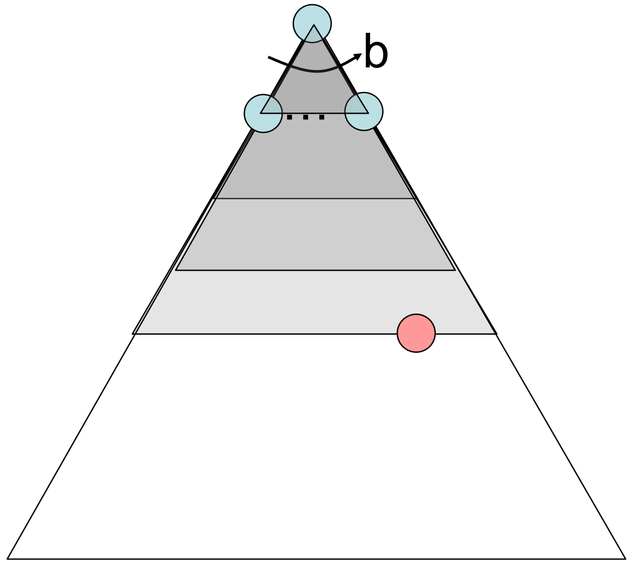


- When will BFS outperform DFS?

- When will DFS outperform BFS?

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. …..
- Isn't that wastefully redundant?
  - Generally most work happens in the bottom level searched, so not so bad
- Compare IDS, BFS (b = 10, d = 5)
  - N_{BFS} = 1 + 10 + 100 + 1,000 + 10,000 + 100,000 = 111,111
  - N_{IDS} = 6 + 50 + 400 + 3,000 + 20,000 + 100,000 = 123,456



BI-DIRECTIONAL SEARCH