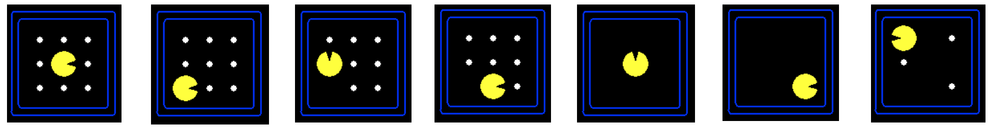
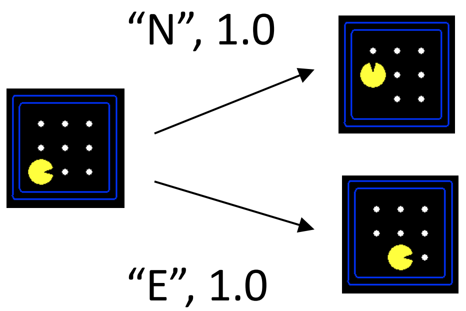
Search Problems

Search Problems

- A search problem consists of:
  - A state space



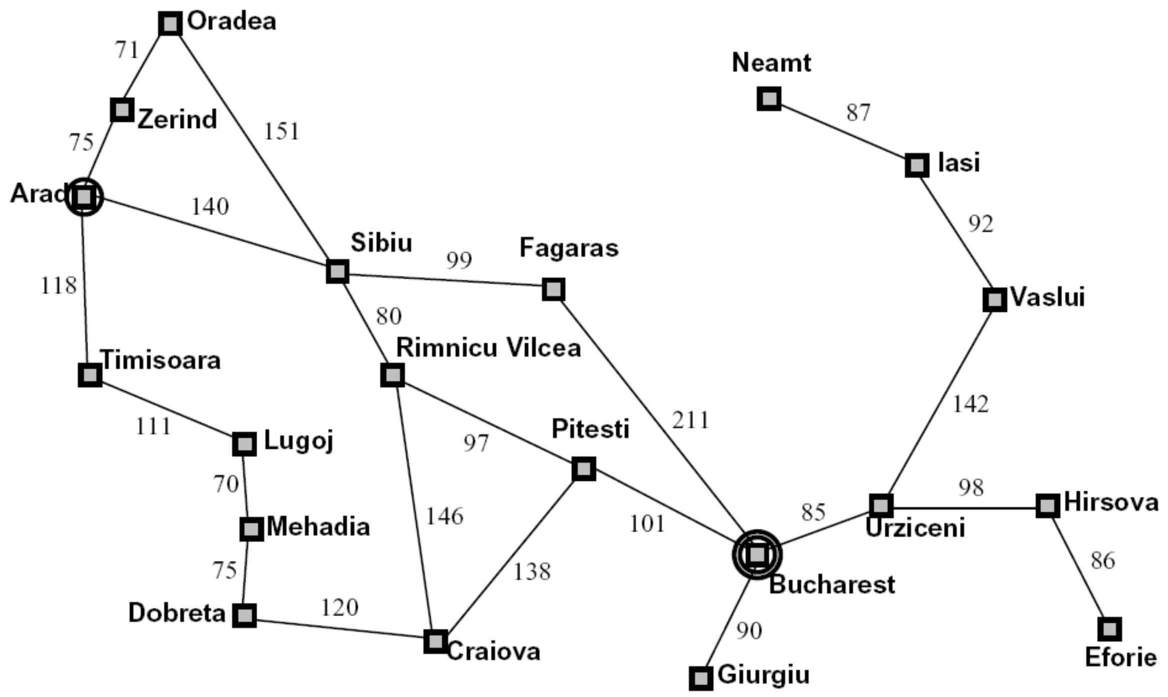
  - A successor function (with actions, costs)



  - A start state and a goal test

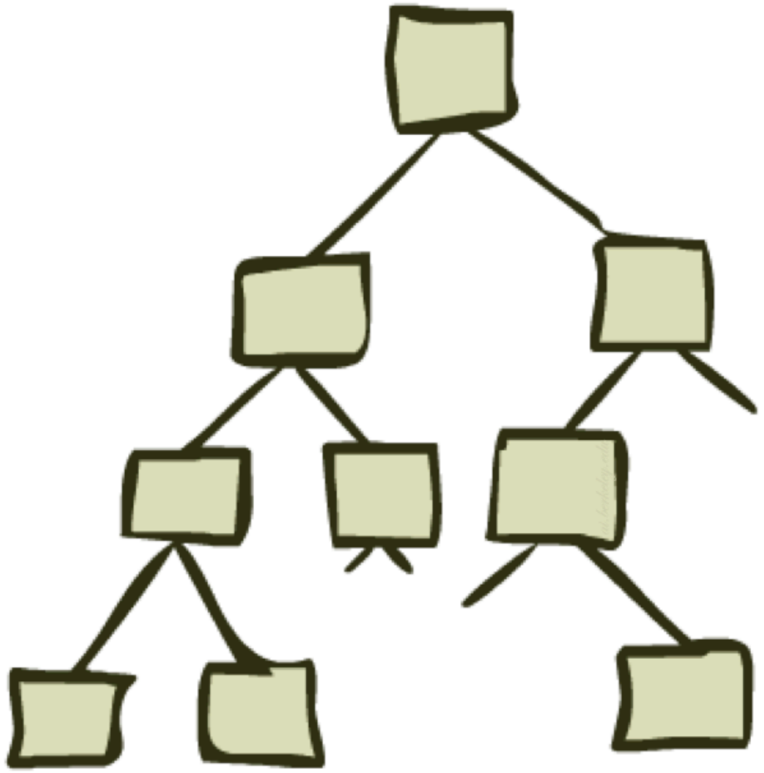
- A solution is a sequence of actions (a plan) which transforms the start state to a goal state

Example: Traveling in Romania



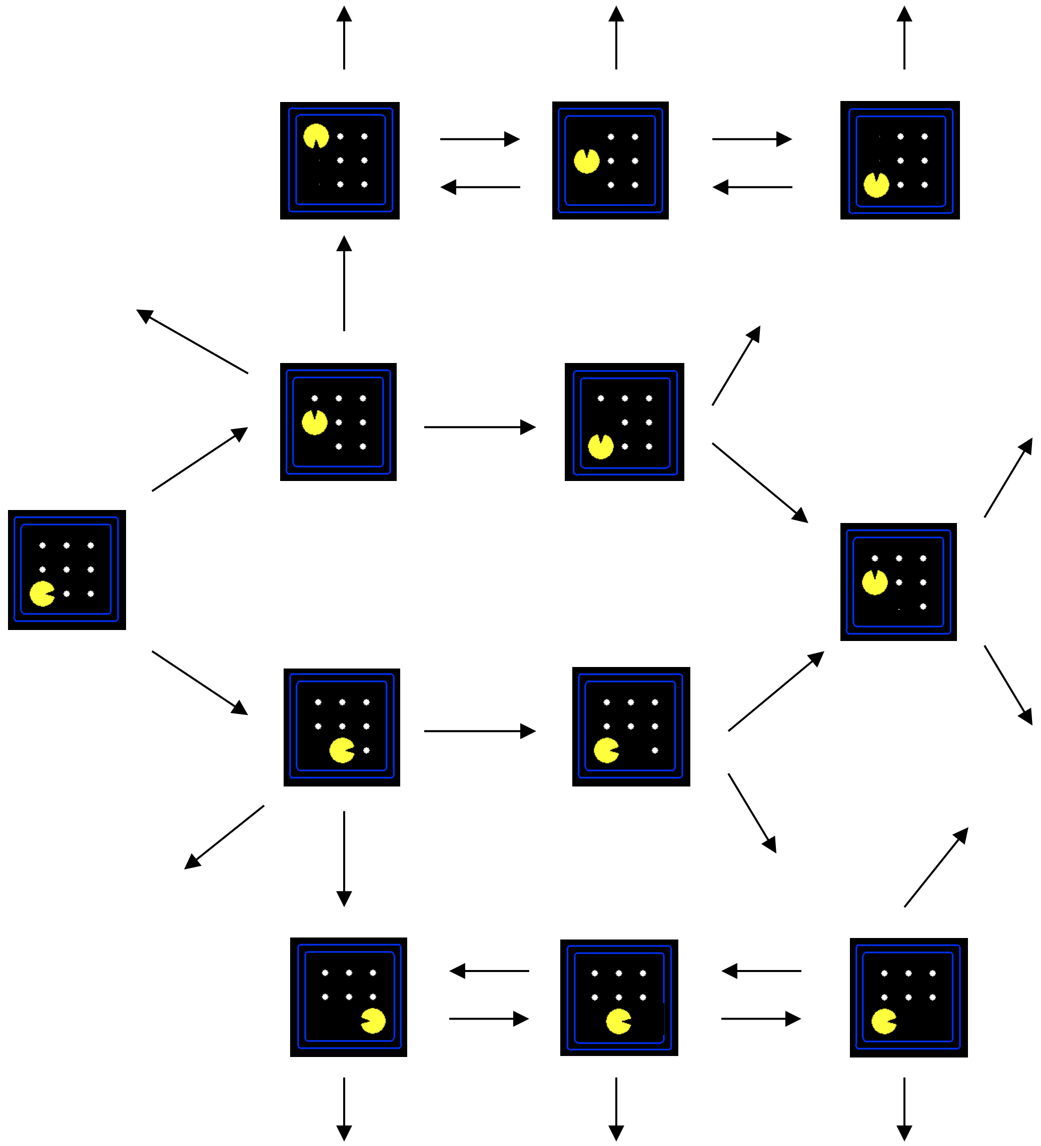
- State space:
  - Cities
- Successor function:
  - Roads: Go to adjacent city with cost = distance
- Start state:
  - Arad
- Goal test:
  - Is state == Bucharest?
- Solution?

State Space Graphs and Search Trees



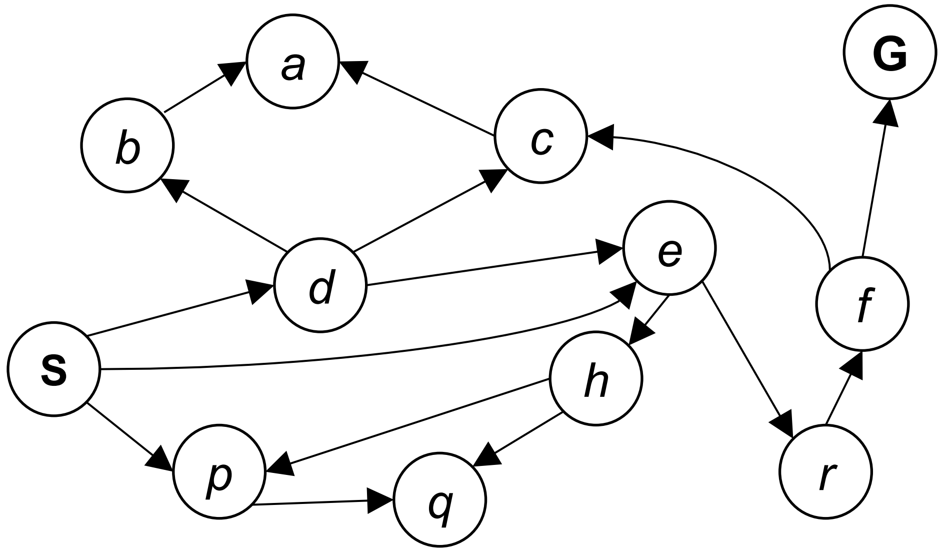
State Space Graphs

- 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



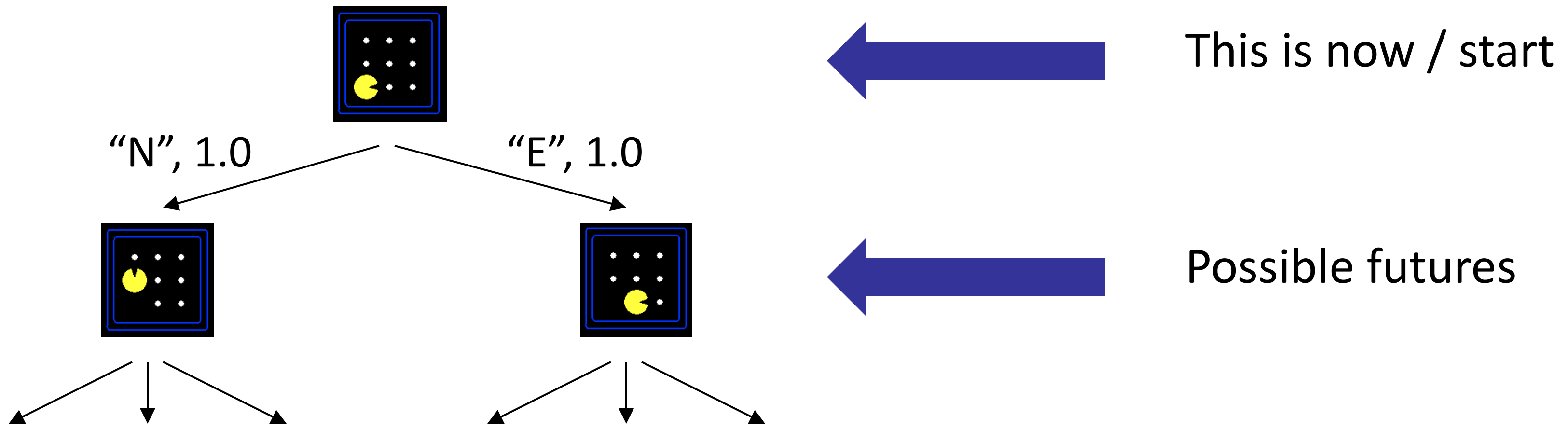
State Space Graphs

- 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 search 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



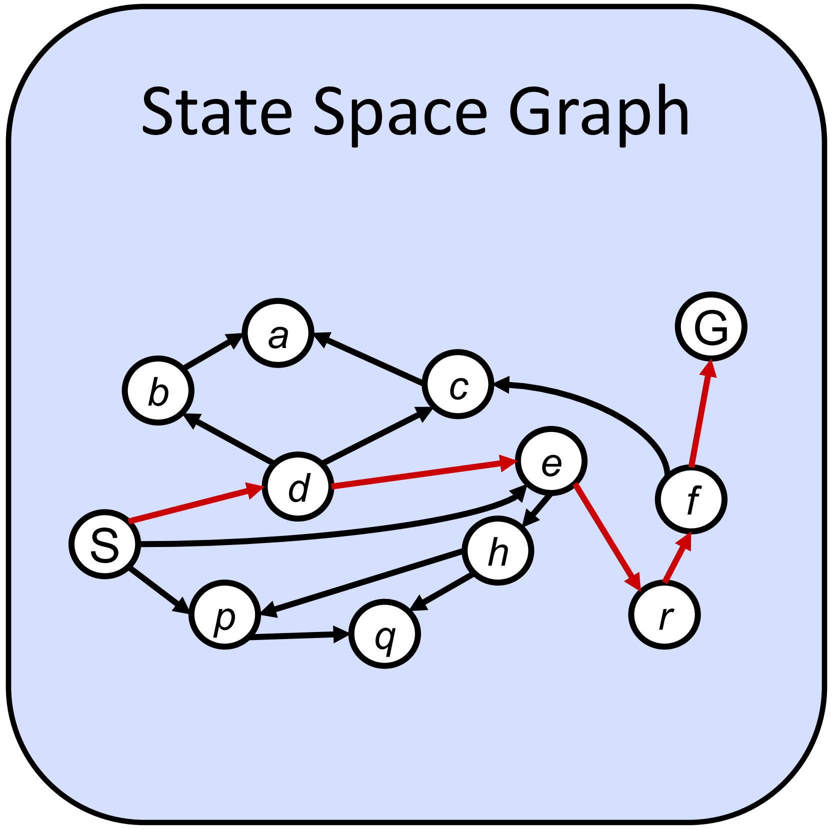
Tiny search graph for a tiny search problem

Search Trees



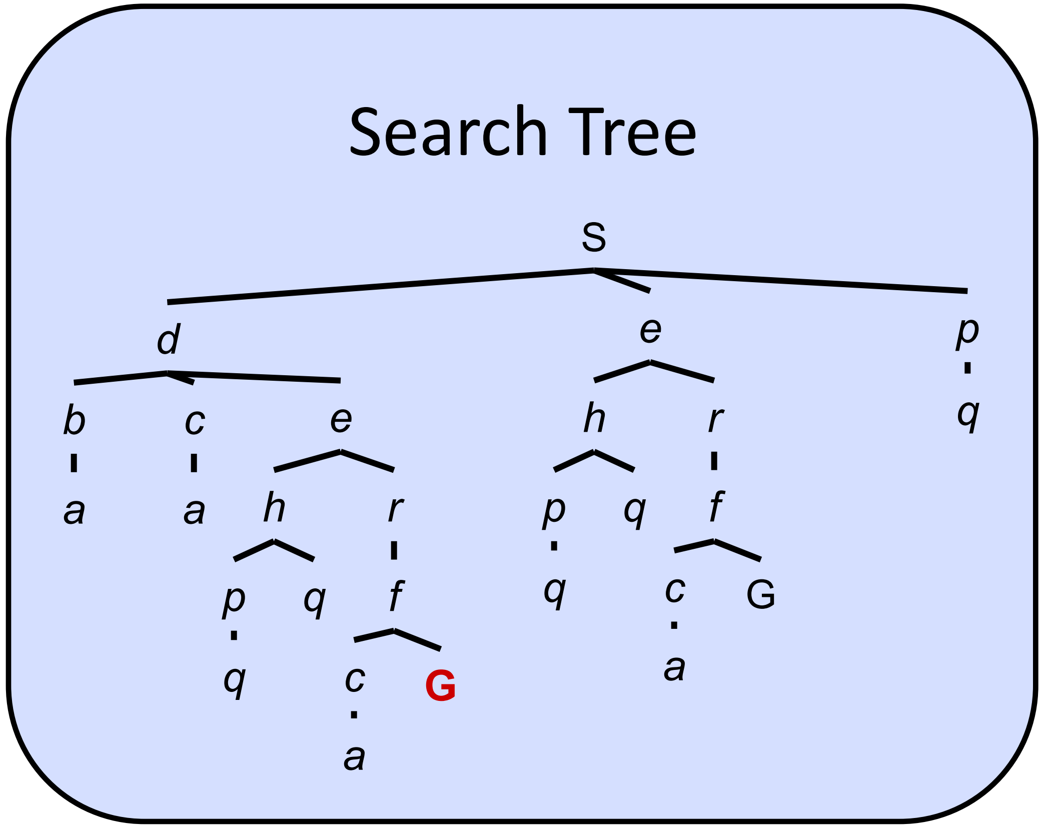
- A search tree:
  - A "what if" tree of plans and their outcomes
  - The start state is the root node
  - Children correspond to successors
  - Nodes show states, but correspond to PLANS that achieve those states
  - For most problems, we can never actually build the whole tree

State Space Graphs vs. Search Trees



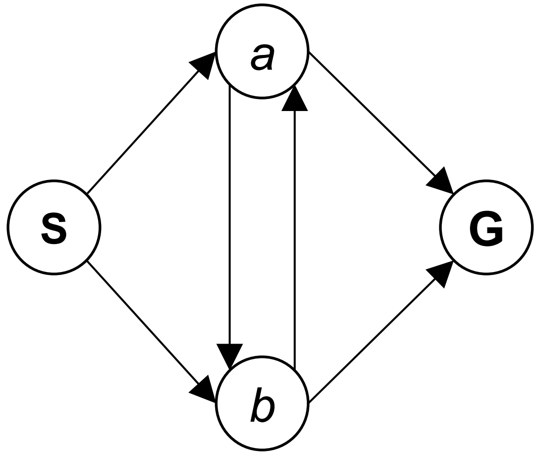
Each NODE in in the search tree is an entire PATH in the state space graph.

We construct both on demand – and we construct as little as possible.

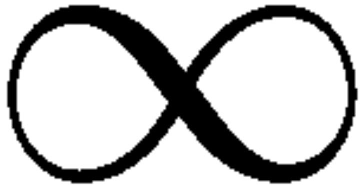


Quiz: State Space Graphs vs. Search Trees

Consider this 4-state graph:



How big is its search tree (from S)?



Important: Lots of repeated structure in the search tree!

Exercise

Q1: what will a state look like in this problem?

Possibly just the position of the player stored as an (X, Y) tuple.

Q2: What will the initial state look like in our example of this problem?

The initial player position (1, 1)

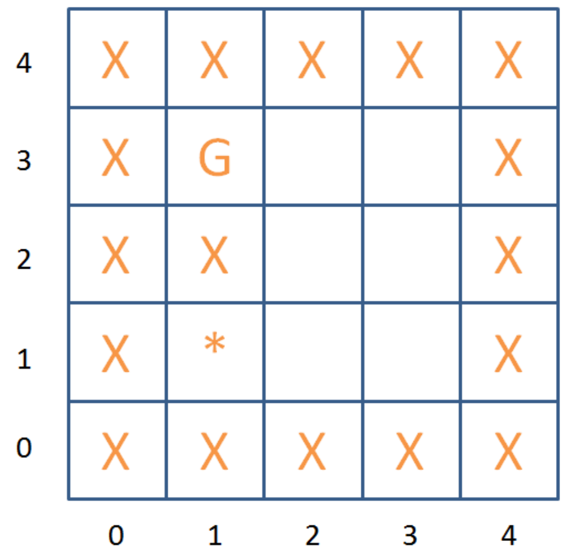
Q3: What will the goal state look like in our example of this problem?

The goal position (1, 3)

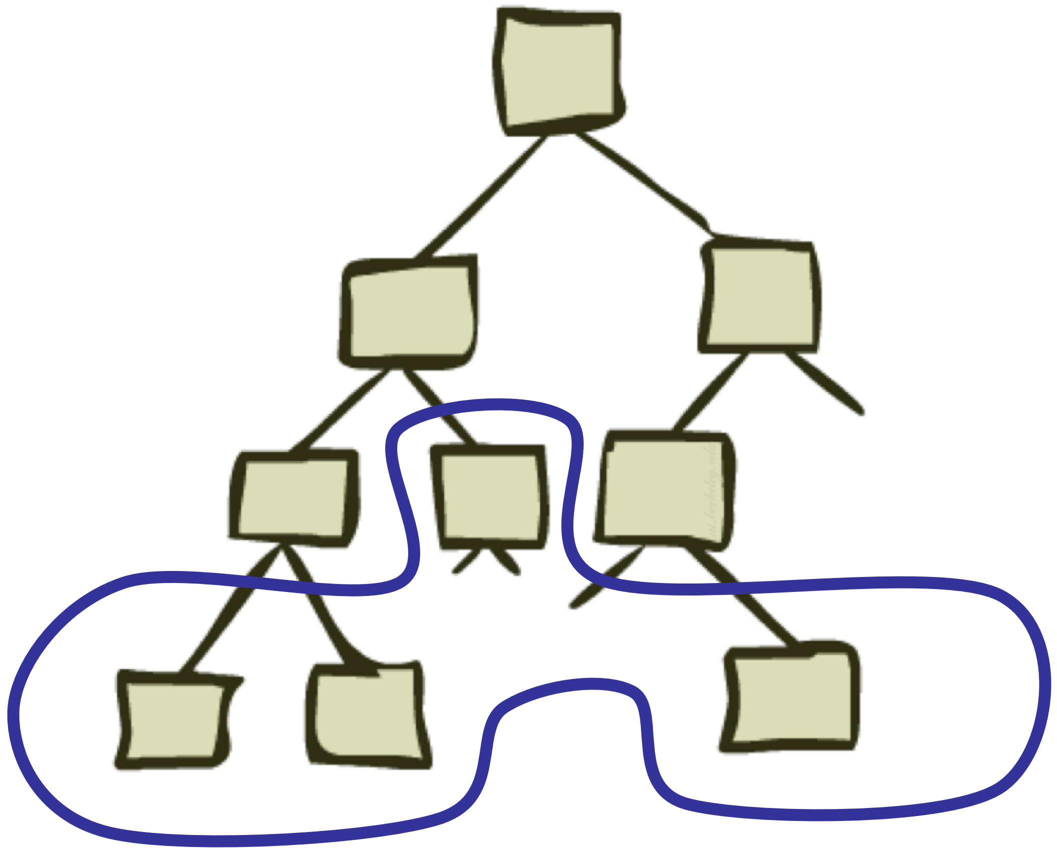
Q4: What will actions look like in this problem?

Any movement from a given state's player position to an adjacent, open or goal tile. This gives us transitions of:
Up: (0, +1), Down: (0, -1), Right: (+1, 0), Left: (-1, 0)

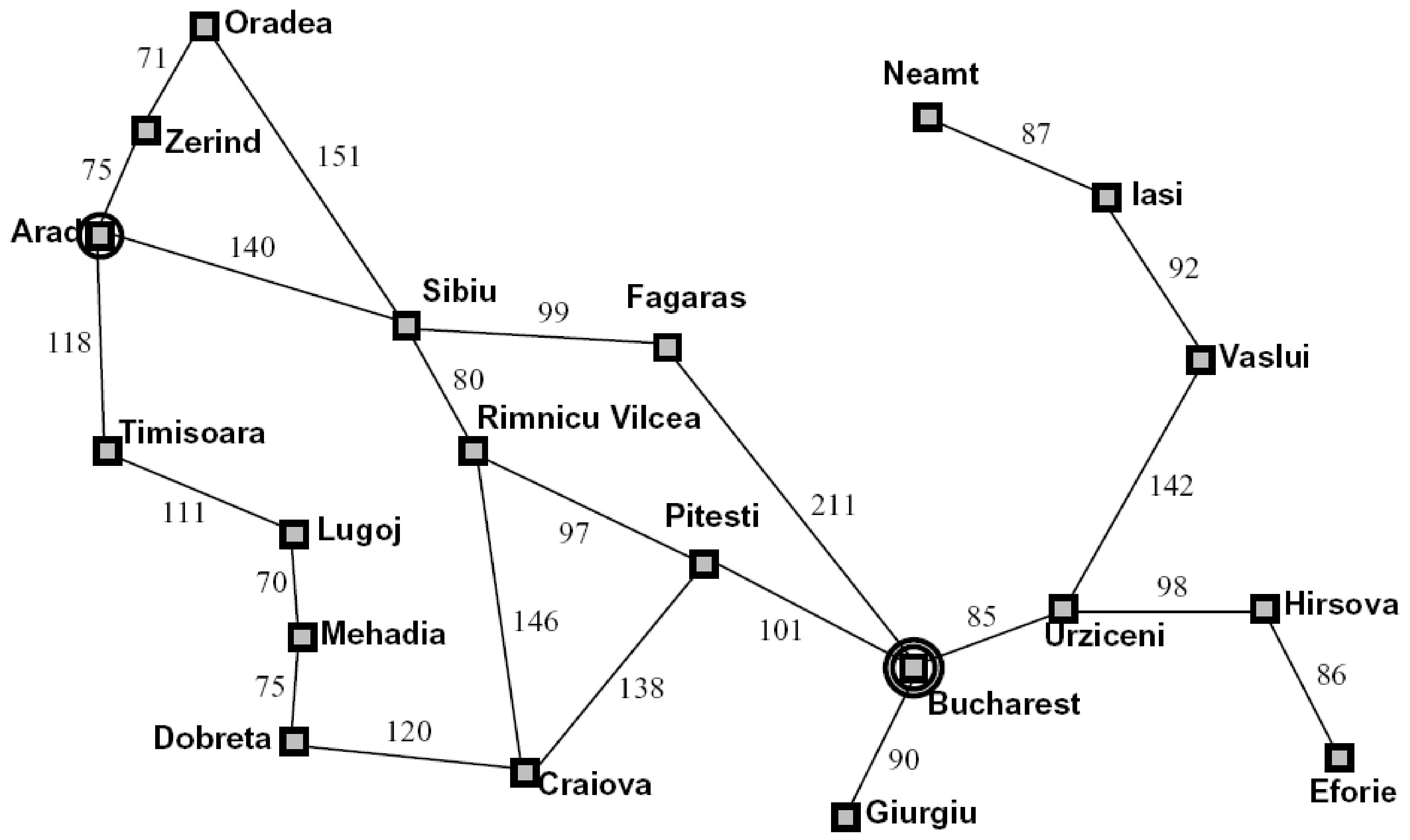
Legend:
X = Barrier (impassable)
* = Player
G = Goal



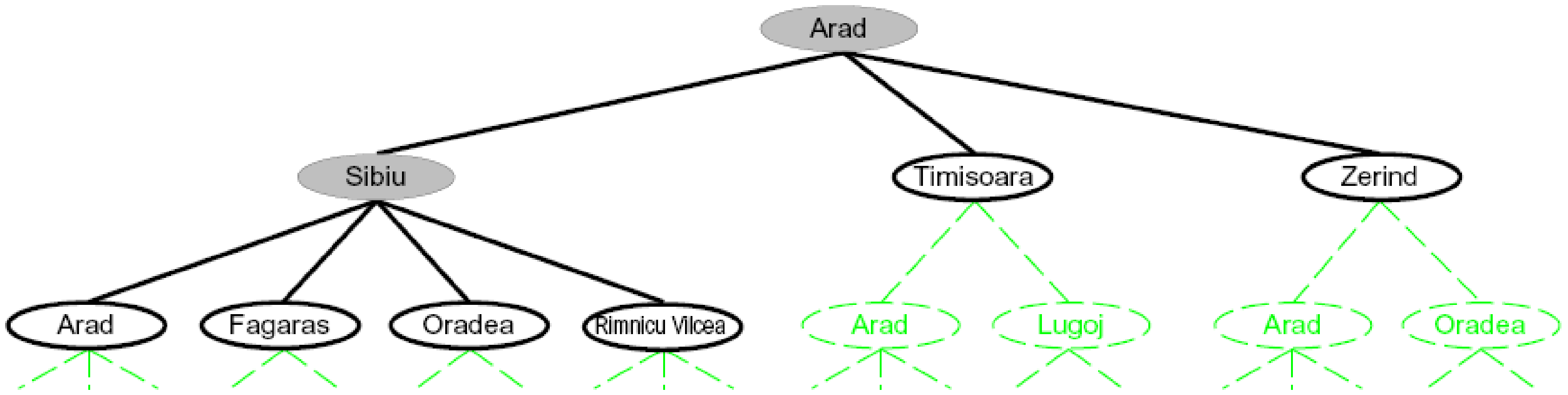
Tree Search



Search Example: Romania



Searching with a Search Tree



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

General Tree Search

```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the search tree
    end
```

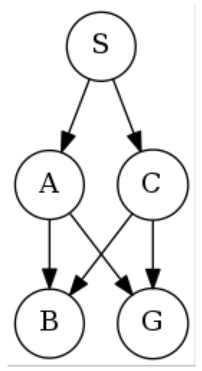
- Important ideas:
  - Fringe
  - Expansion
  - Exploration strategy
- Main question: which fringe nodes to explore?

LISP Search Tree Framework

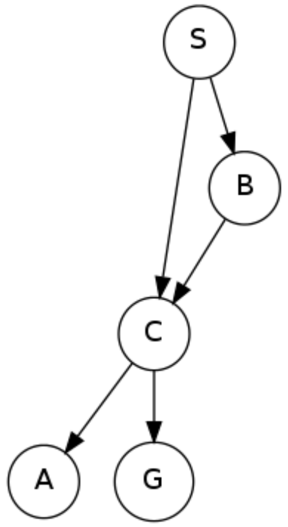
```
; ...for some expand-node function defined
 (defun BB-SEARCH (frontier)
     (cond
         ; Base case: ran out of nodes, return nil
         ; (failed to find goal state)
         ((null frontier) nil)
         ; Otherwise, more nodes on frontier, so keep
         ; expanding them
         (t (EXPAND-NODE frontier))
     )
 )

 ; ...for some goal-state check and
 ; choose-node functions defined
 (defun EXPAND-NODE (frontier)
     (let* ((next (choose-node frontier)))
         (cond
             ; Base case: found goal; done!
             ((goal-state next) next)
             ; Otherwise, expand and continue
             (t BB-SEARCH (append (remove next frontier))
                                  (successors next)
             )
         )
     )
 )
 ; Pseudo-code credit to Evan Lloyd
```

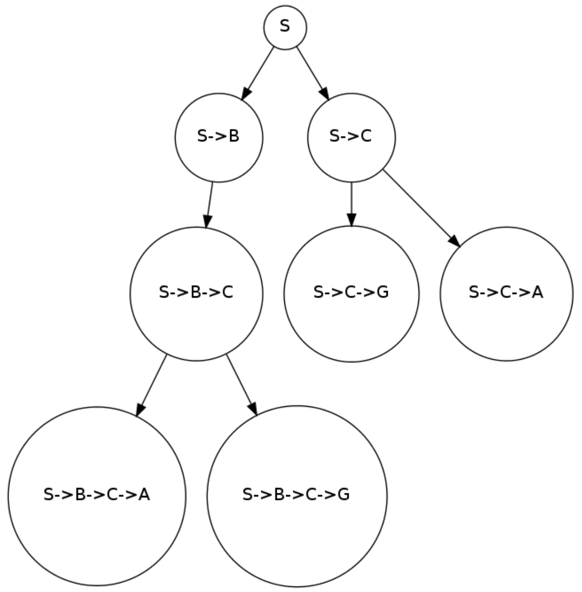
Exercise



7

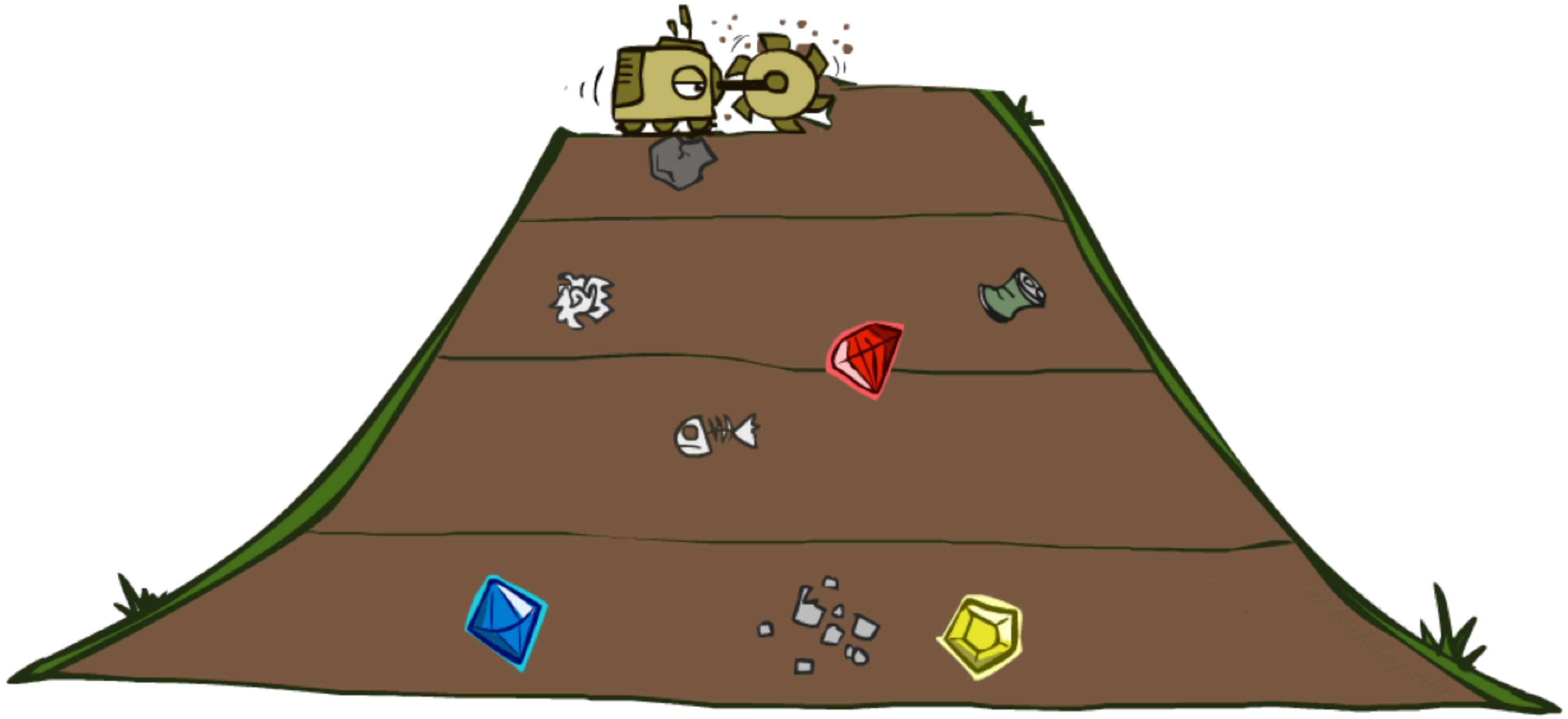


8



Q: How many nodes are in the complete search tree for the given state space graph? The start state is S. You may find it helpful to draw out the search tree on a piece of paper.

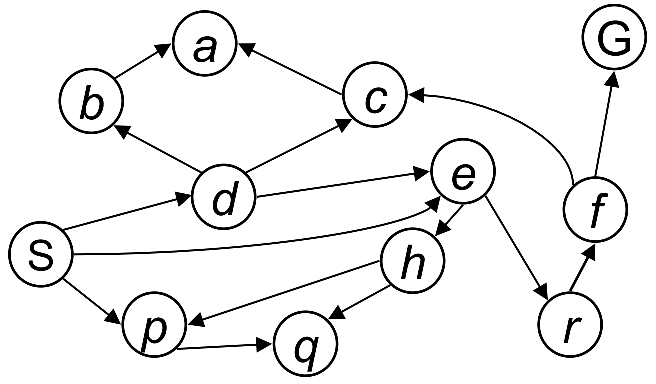
Breadth-First Search

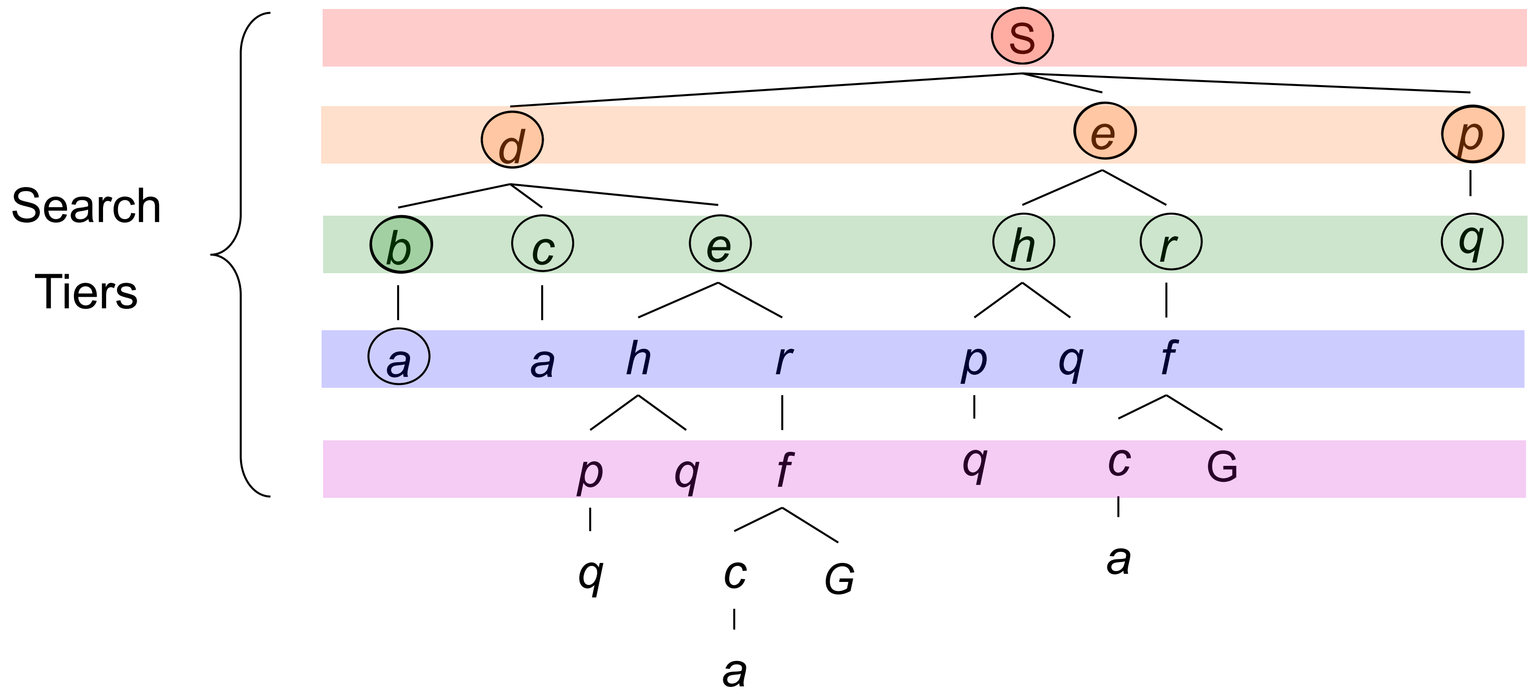


Breadth-First Search

Strategy: expand a shallowest node first

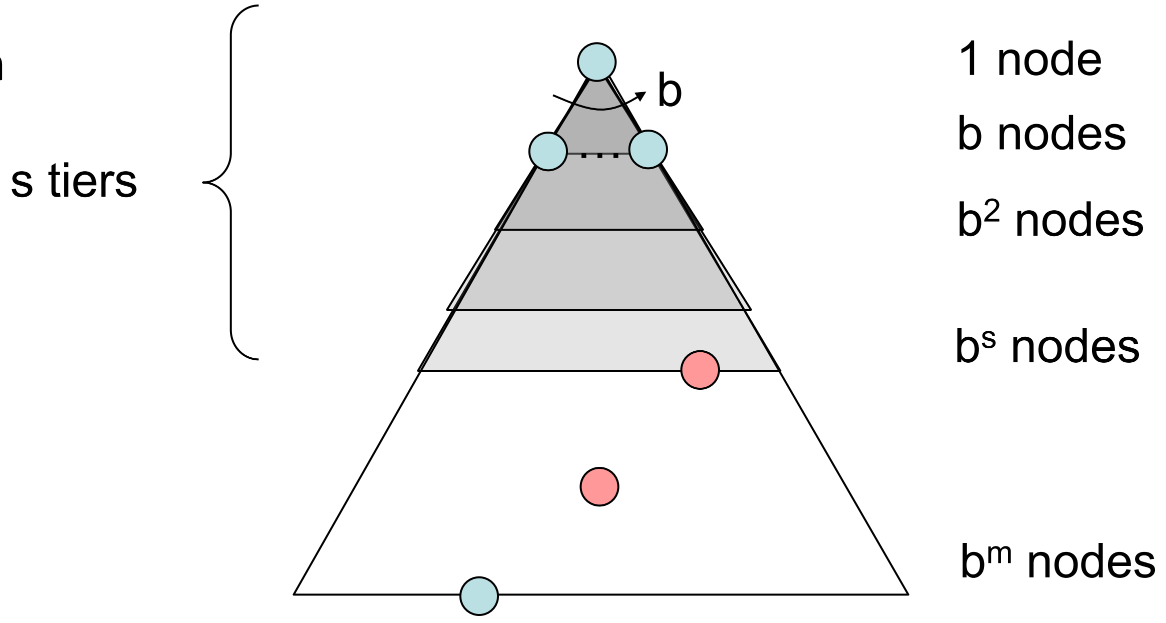
Implementation: Fringe is a FIFO queue





Breadth-First Search (BFS) Properties

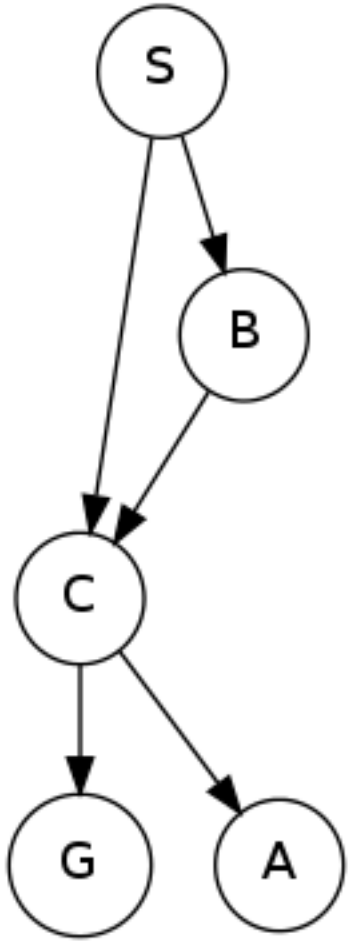
- What nodes does BFS expand?
  - Processes all nodes above shallowest solution
  - Let depth of shallowest solution be s
  - Search takes time $O(b^s)$
- How much space does the fringe take?
  - Has roughly the last tier, so $O(b^s)$
- Is it complete?
  - s must be finite if a solution exists, so yes!
- Is it optimal?
  - Only if costs are all 1 (more on costs later)



Exercise

Q: Consider a breadth-first graph search on the graph below, where S is the start and G is the goal state. Assume that ties are broken alphabetically (so a partial plan S->X->A would be expanded before S->X->B and S->A->Z would be expanded before S->B->A). You may find it helpful to execute the search on scratch paper.

Please show the final path returned by breadth-first graph search. Your answer should be a string with S as your first character and G as your last character.



Solution to Exercise

Step 1: Expand S
Fringe: S-B, S-C
Closed Set: S

Step 2: Expand S-B
Fringe: S-C, S-B-C
Closed Set: S, B

Step 3: Expand S-C
Fringe: S-B-C, S-C-A, S-C-G
Closed Set: S, B, C

Step 4: Pop S-B-C from our fringe, but do not expand it, because C is in our closed set
Fringe: S-C-A, S-C-G
Closed Set: S, B, C

Step 5: Expand S-C-A
Fringe: S-C-G
Closed Set: S, B, C, A

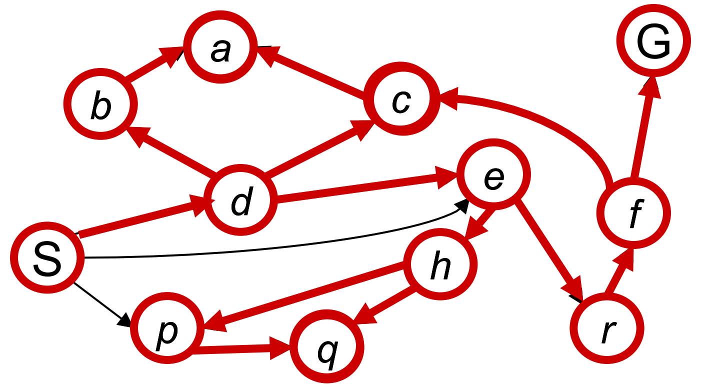
Step 6: Expand S-C-G, finding the goal

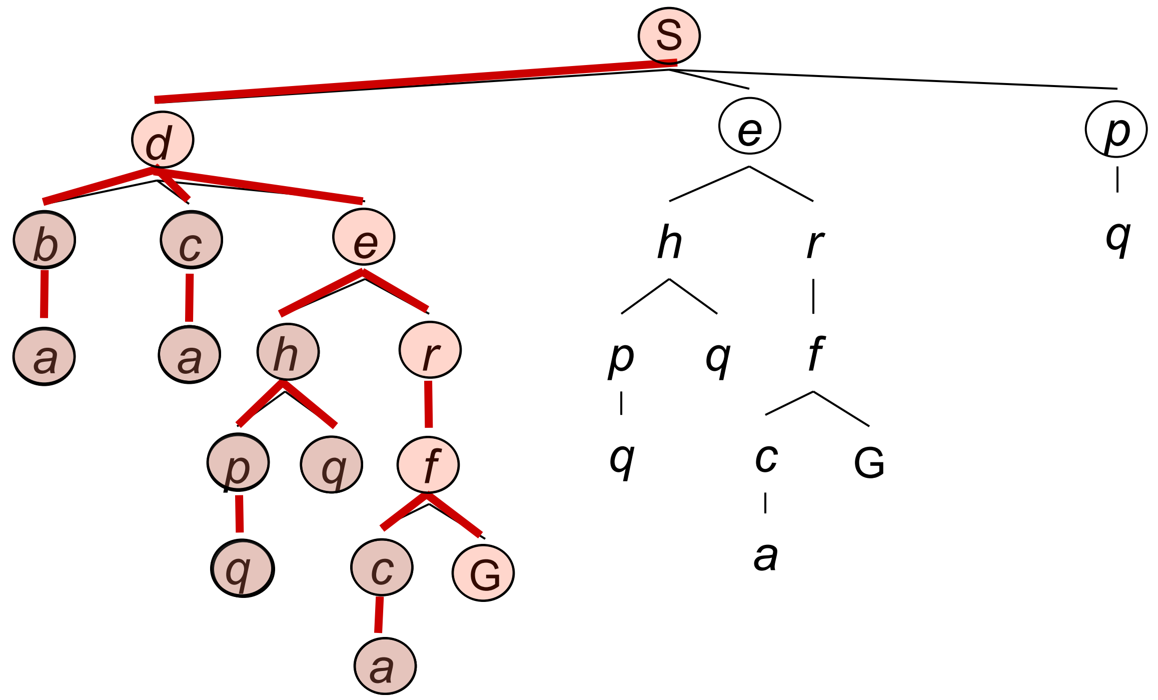
Depth-First Search

Depth-First Search

Strategy: expand a deepest node first

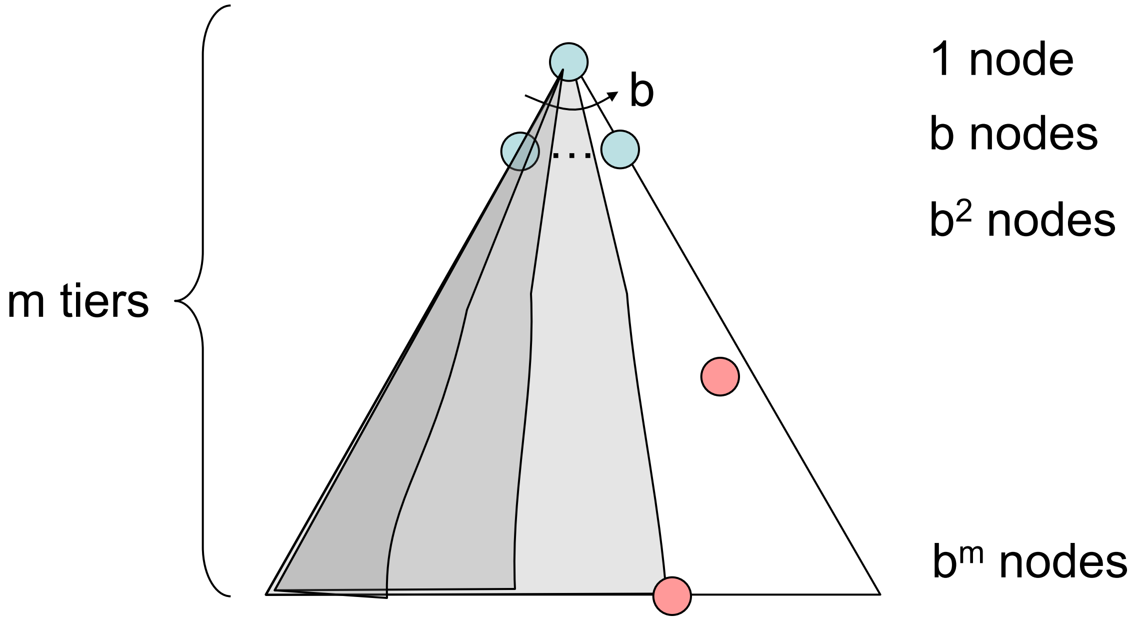
Implementation: Fringe is a LIFO stack





Depth-First Search (DFS) Properties

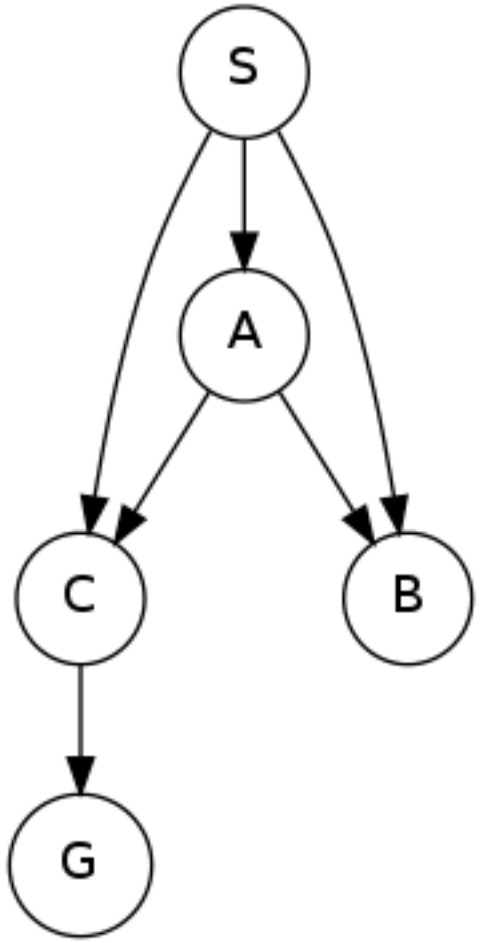
- What nodes DFS expand?
  - Some left prefix of the tree.
  - Could process the whole tree!
  - If m is finite, takes time $O(b^m)$
- How much space does the fringe take?
  - Only has siblings on path to root, so $O(bm)$
- Is it complete?
  - m could be infinite, so only if we prevent cycles (more later)
- Is it optimal?
  - No, it finds the “leftmost” solution, regardless of depth or cost



Exercise

Q: Consider a depth-first graph search on the graph below, where S is the start and G is the goal state. Assume that ties are broken alphabetically (so a partial plan S->X->A would be expanded before S->X->B and S->A->Z would be expanded before S->B->A). You may find it helpful to execute the search on scratch paper.

Please show the final path returned by depth-first graph search. Your answer should be a string with S as your first character and G as your last character.



Solution to Exercise

Step 1: Expand S
Fringe: S-A, S-B, S-C
Closed Set: S

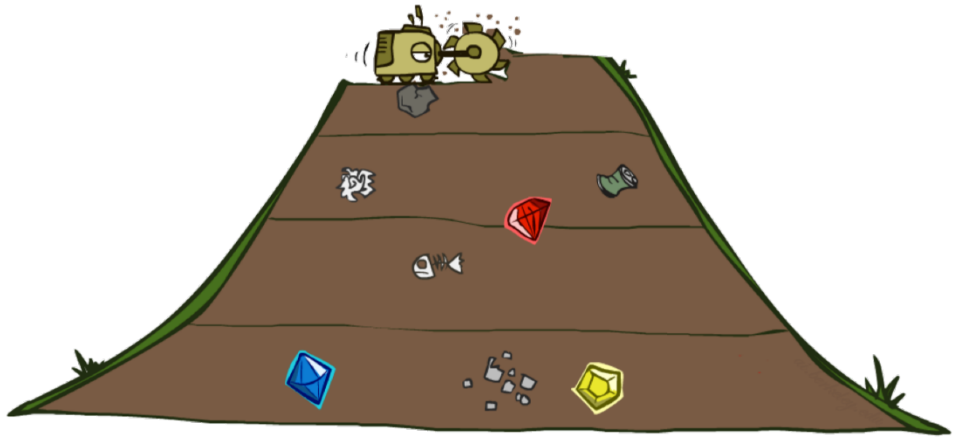
Step 2: Expand S-A
Fringe: S-A-B, S-A-C, S-B, S-C
Closed Set: S, A

Step 3: Expand S-A-B
Fringe: S-A-C, S-B, S-C
Closed Set: S, A, B

Step 4: Expand S-A-C
Fringe: S-A-C-G, S-B, S-C
Closed Set: S, A, B, C

Step 5: Expand S-A-C-G, finding the goal

Quiz: DFS vs BFS

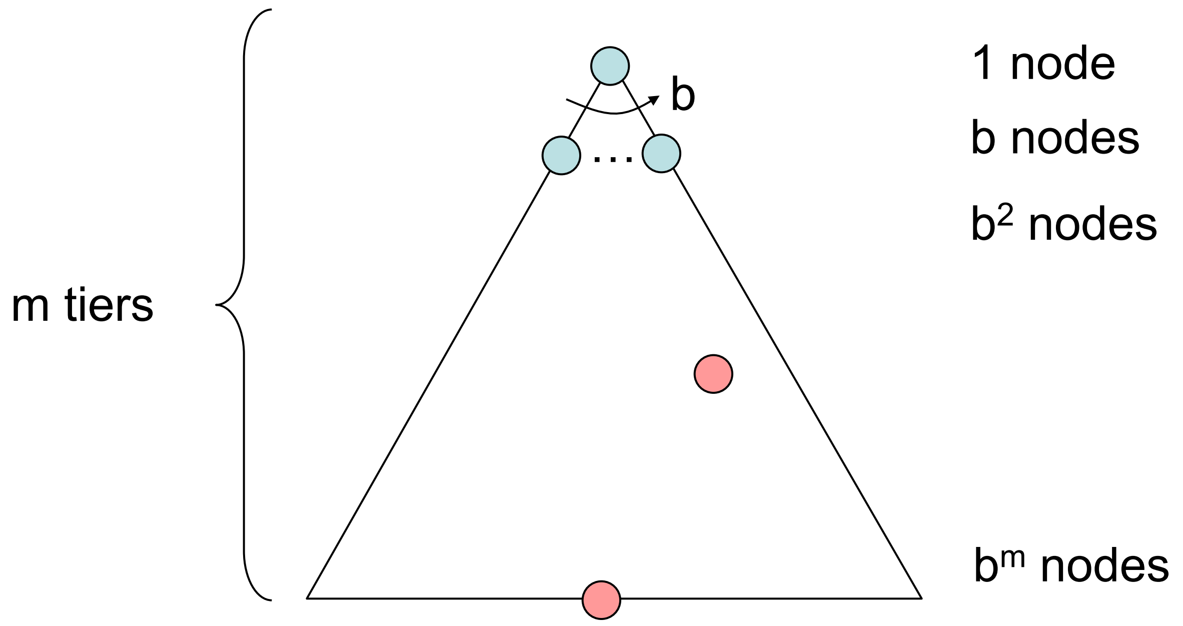


Search Algorithm Properties

- 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
  - solutions at various depths

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

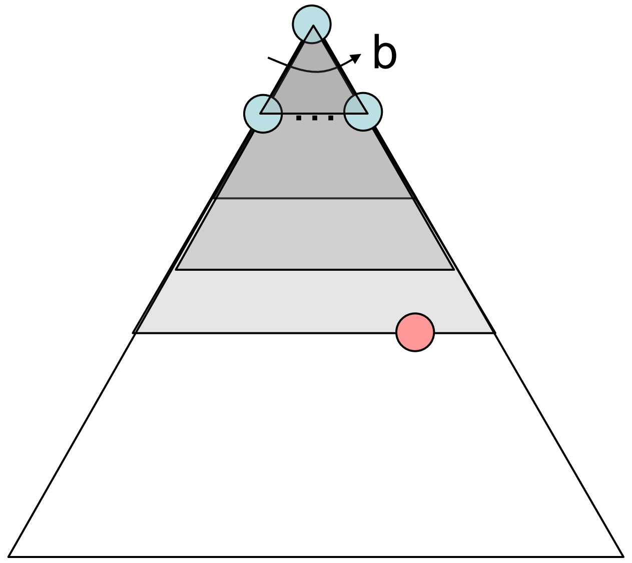


Quiz: DFS vs BFS

- 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 lowest level searched, so not so bad!

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)
  - Practically, for DFS and BFS, you can avoid the log(n) overhead from an actual priority queue, by using stacks and queues
  - Can even code one implementation that takes a variable queuing object