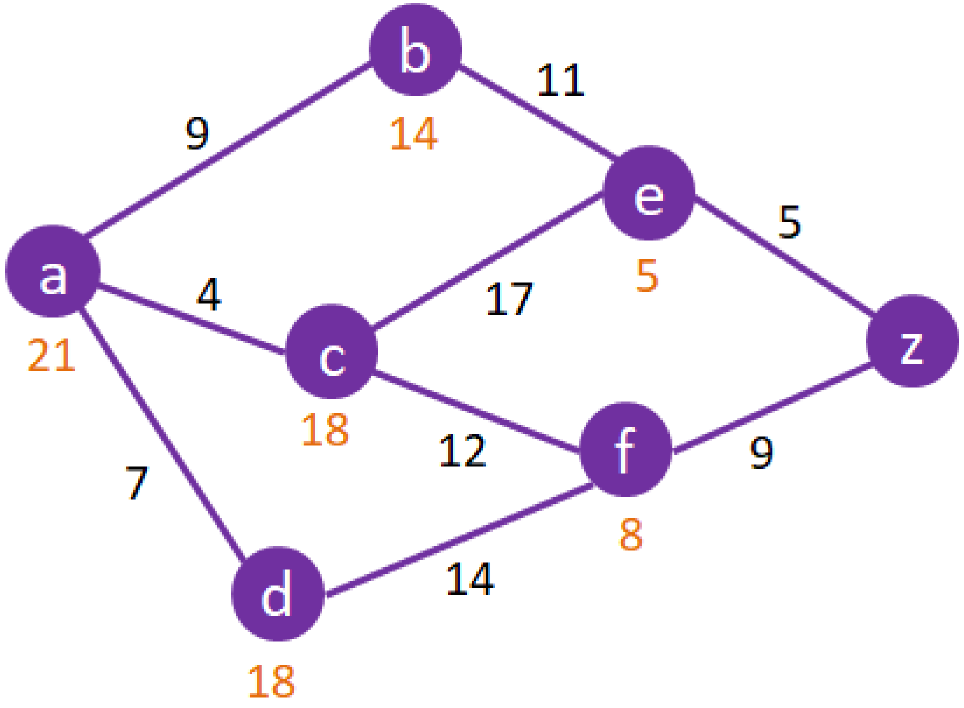
CS 6511: Artificial Intelligence

Search



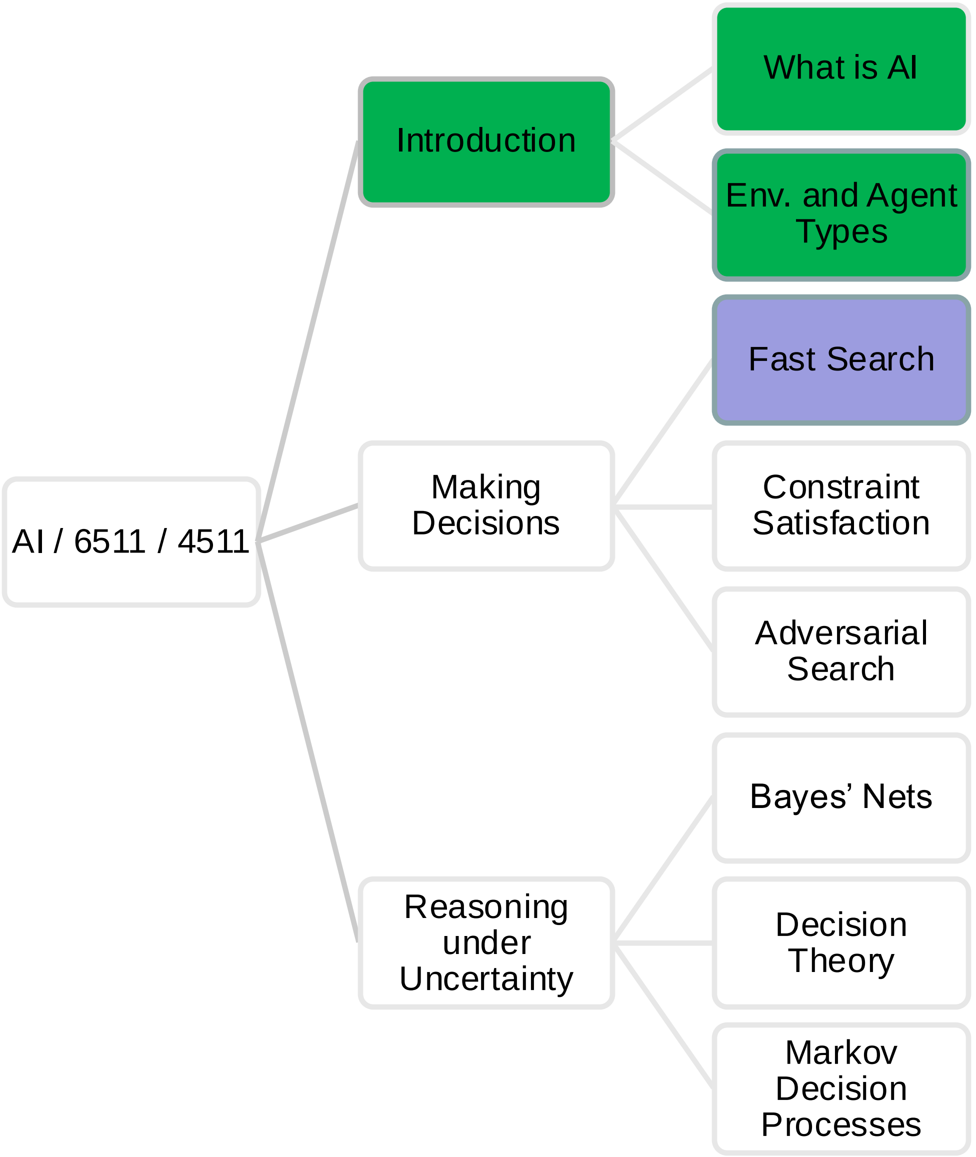
Instructor: Prof. Amrinder Arora

The George Washington University

Course Outline

Other AI Topics (Not included in this class):

- Robotics
- NLP
- Machine Learning
- Big Data
- AR/VR
- Speech Synthesis



Puzzles

Suppose you have two jugs, one capable of holding 5 cups, and one capable of holding 8 cups.

[The jugs are irregularly shaped and without markings, so you can't determine how much water is in either jug unless it is completely full or completely empty.]

You also have a faucet, and as much water as you'd like.

Can you get 3 cups?

Can you obtain 1 cup? 2 cups? 4 cups? 6 cups? 7 cups?

Puzzles (cont.)

Where can we go from here:

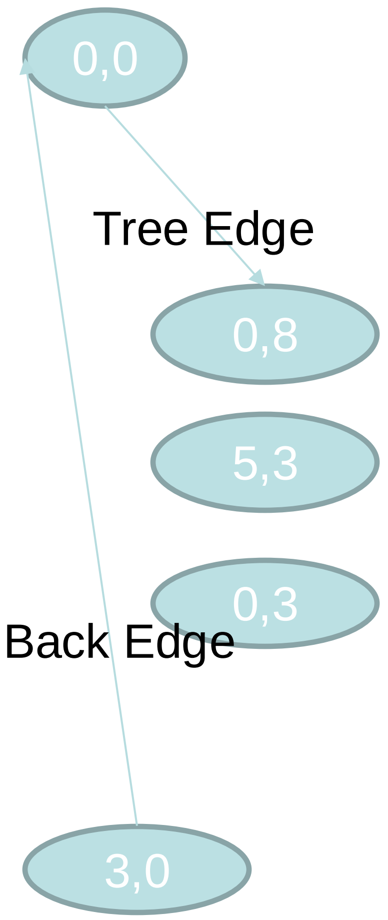
(x,y) →

a. (0,y) / (x,0) ← Empty first/second
b. (5,y) / (x,8) ← Fill first/second
c. (5,x+y-5) ← Second to First, x+y > 5
d. (x+y,0) ← Second to First
e. (x+y-8,8) ← First to Second, x+y > 8
f. (0,x+y) ← First to Second

PUZZLES (cont.)

(0,0)

1. →(0,8) // b
2. →(5,3) // c
3. →(0,3) // a
4. →(3,0) // d
5. →(3,8) // a
6. →(5,6) // c
7. →(0,6) // a
8. →(5,1) // c
9. →(0,1) // a



Traversal problem

Solution

[5,8]

- 0,0
- 0,8
- 5,3
- 0,3
- 3,0
- 3,8
- 5,6
- 0,6
- 5,1
- 0,1

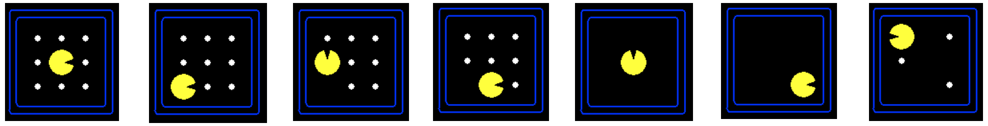
Problem, to Avoid!

[5,8]

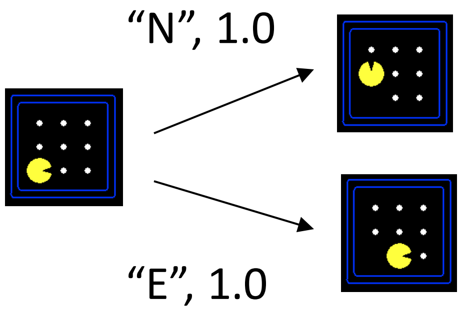
- 0,0
- 5,0
- 5,8
- 0,8
- 0,0
- 5,0
- 5,8
- 0,8
- 0,0
- 5,0
- 5,8
- 0,8

Search Problems

- A search problem consists of:
  - A state space



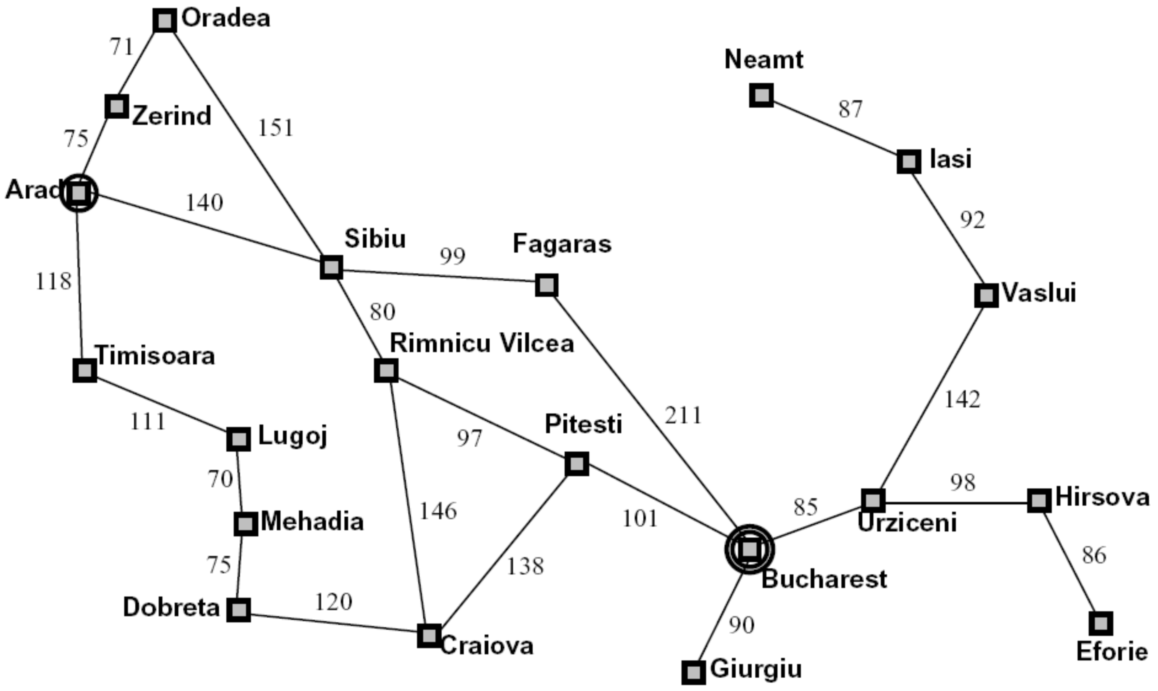
  - A successor function (with actions, costs)



  - 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

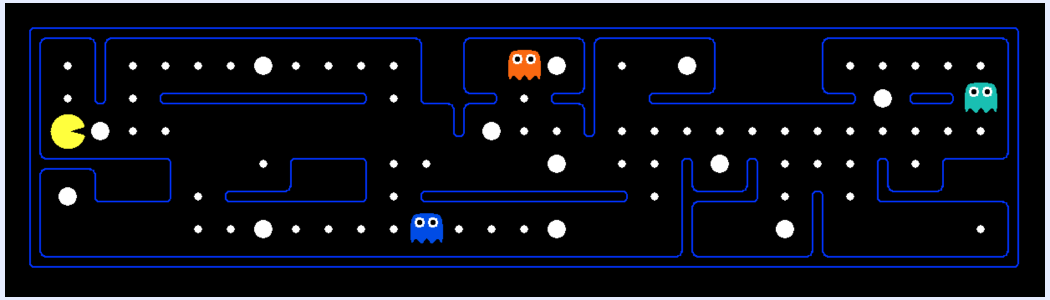
Example: Traveling in Romania



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

What’s 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?

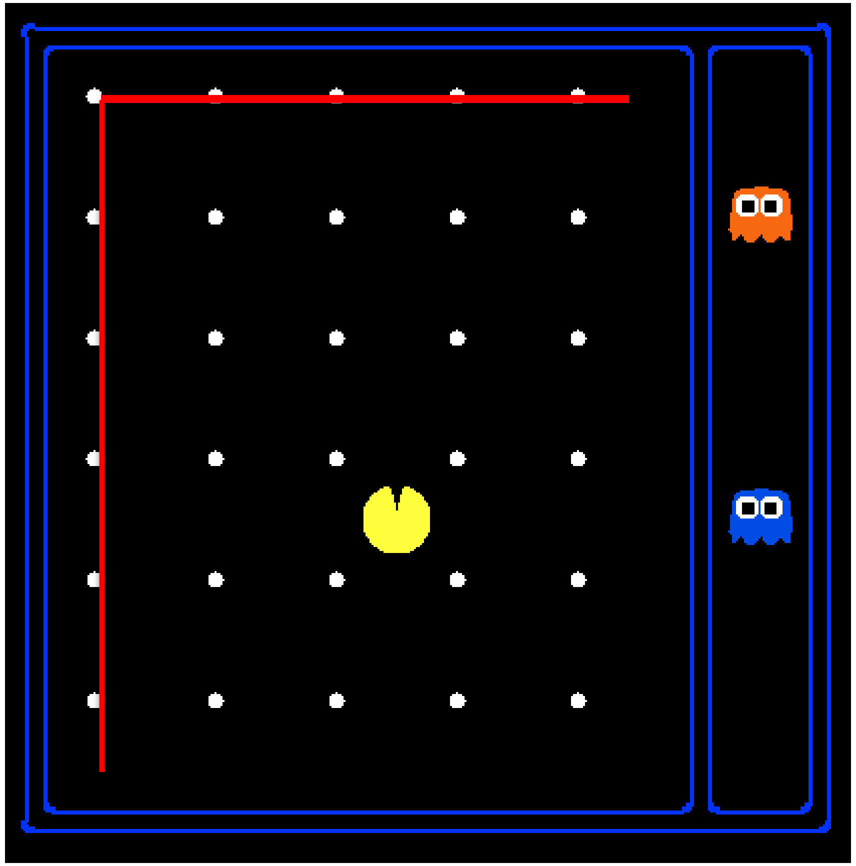
- World state:
  - Agent positions: 120
  - Food count: 30
  - Ghost positions: 12
  - Agent facing: NSEW

- How many
  - World states?

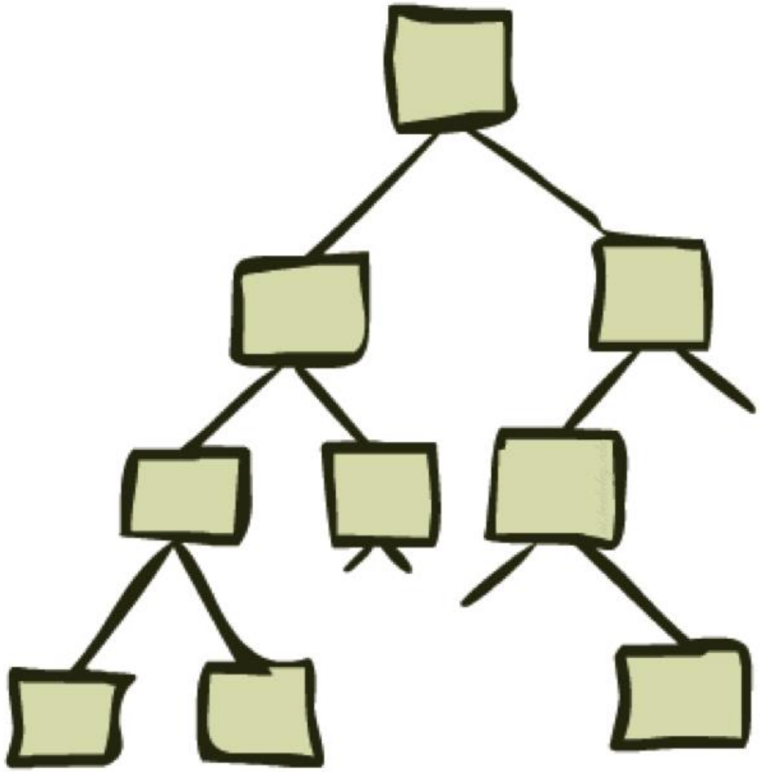
    $120 \times 2^{30} \times 12 \times 12 \times 4$
  - States for pathing?

    120
  - States for eat-all-dots?

    120×2^{30}

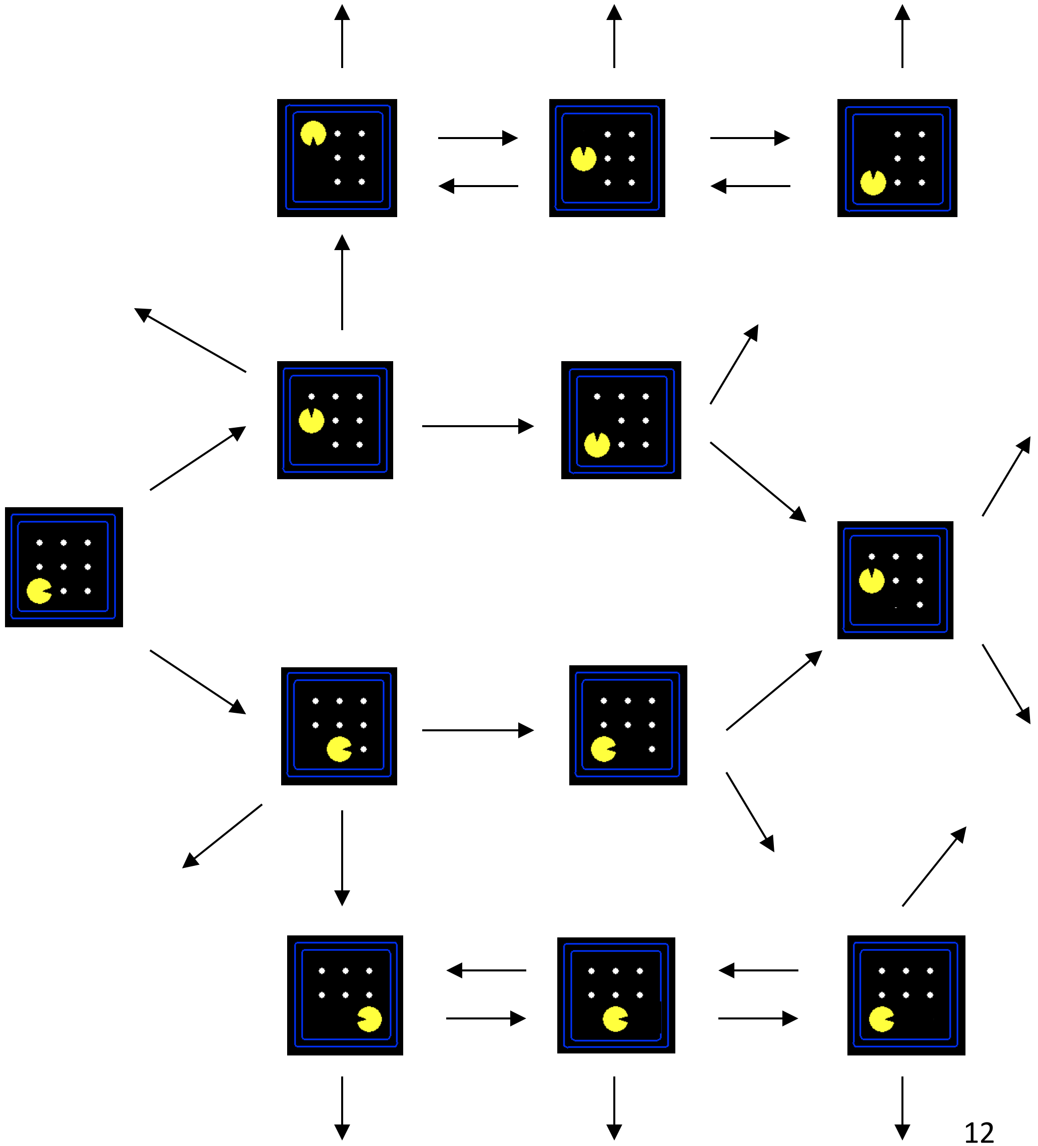


Search Graphs and Search Trees

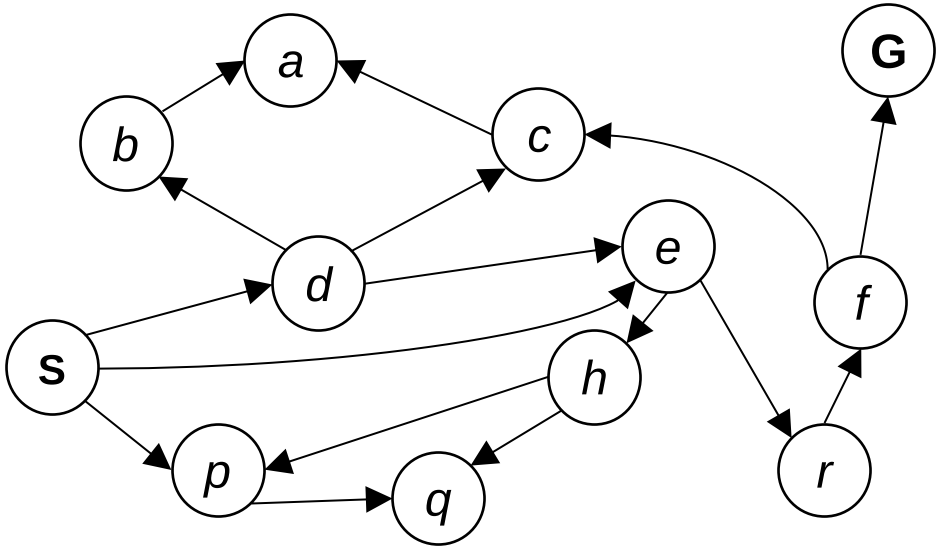


State Space Graphs (Search Graphs)

- State space graph (Search Graph for Short): 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

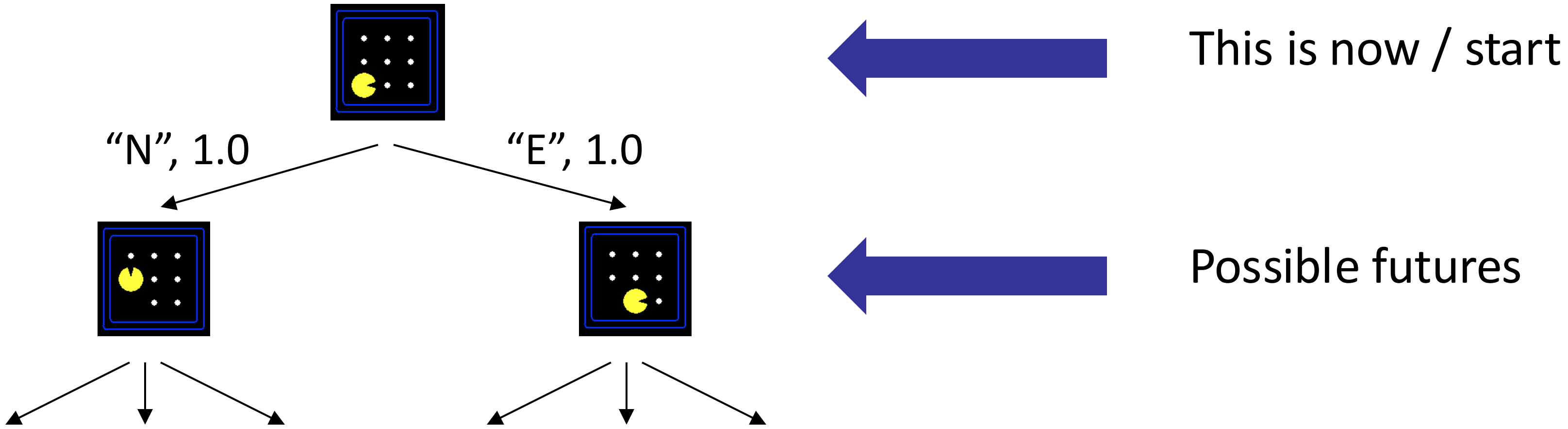


Search Graphs



Tiny search graph for a tiny search problem

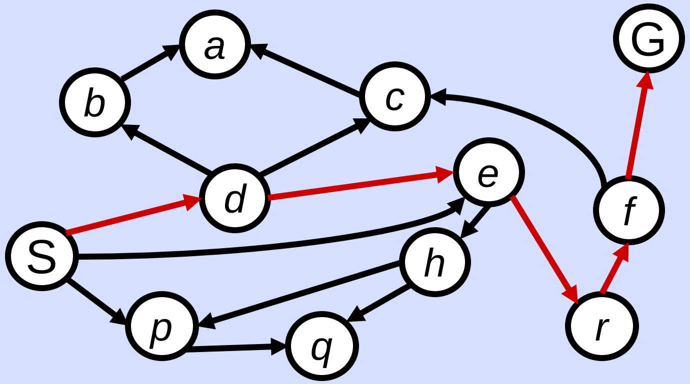
Search Trees



- 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

Search Graphs vs. Search Trees

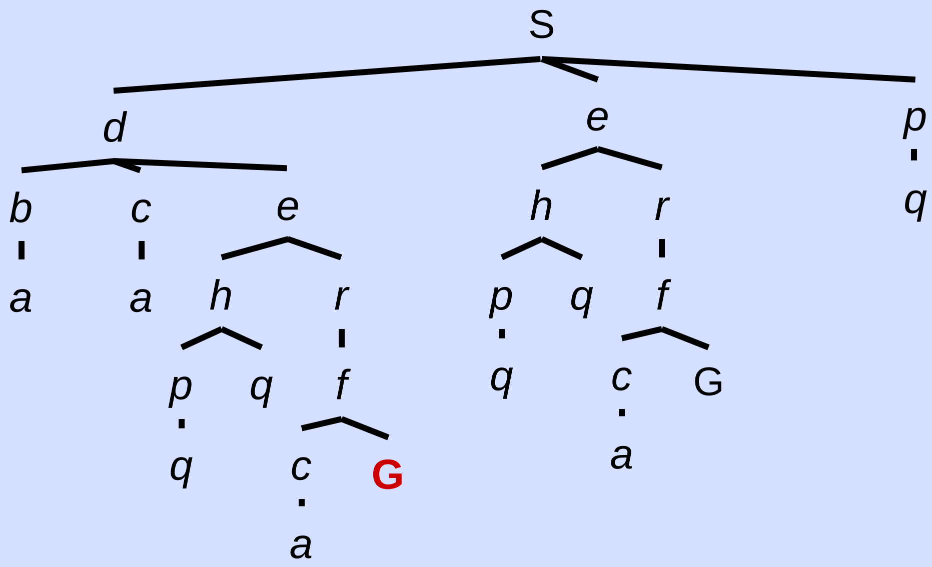
Search Graph



Each NODE 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.

Search Tree



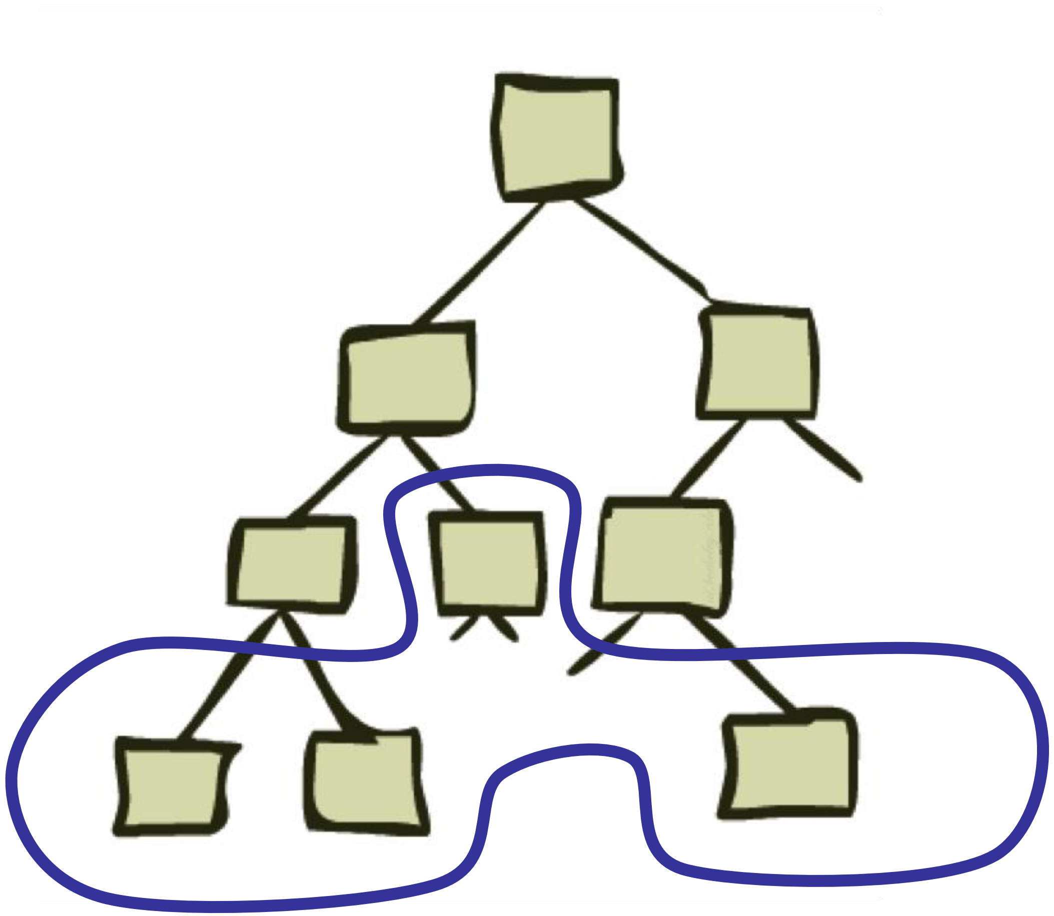
“Algorithms that forget their history are doomed to repeat it.”

Graph Search vs. Tree Search

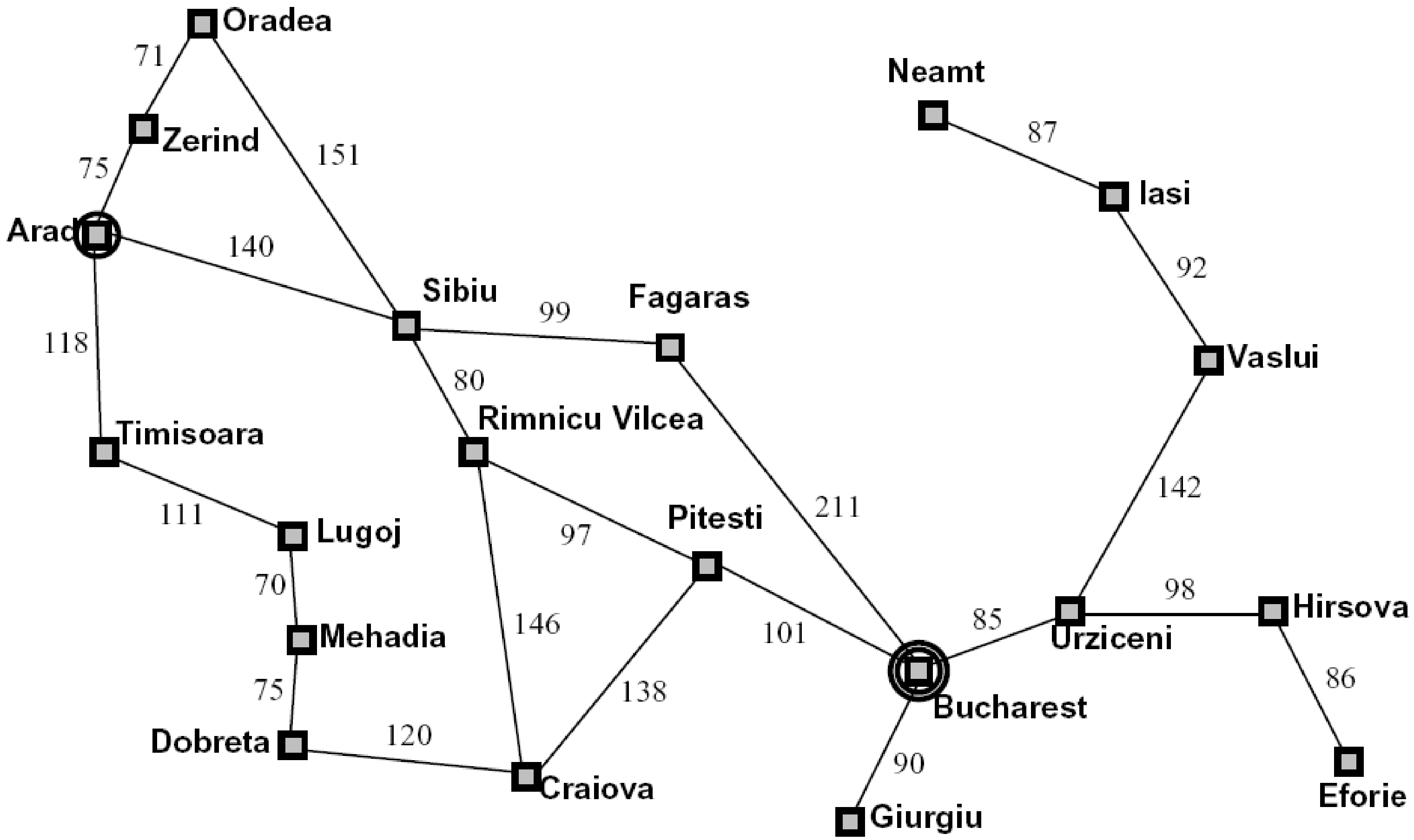
- There is always a lot of confusion about this concept. (And the naming does not help!)
- **The underlying problem is always a graph – So, the difference is not whether the problem is a tree (a special kind of graph), or a general graph!**
- The distinction instead is the structure that we are **maintaining** – tree, or a graph.
  - This is done using a closed list to only add the nodes that are "new".

https://ai.stackexchange.com/questions/6426/what-is-the-difference-between-tree-search-and-graph-search

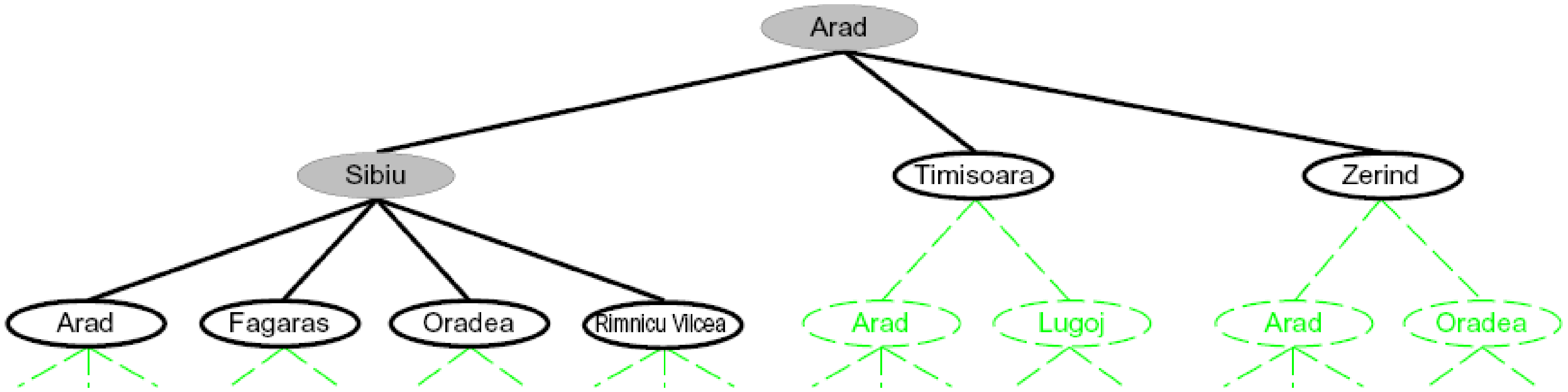
Tree Search



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

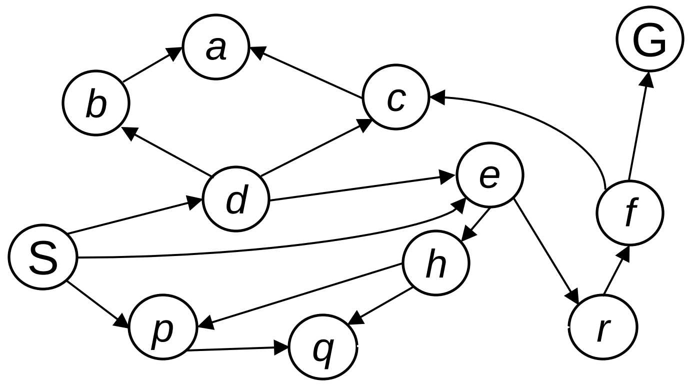
General Tree Search

```
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?

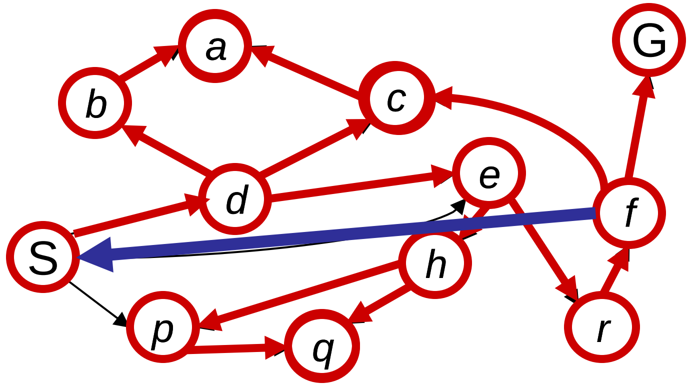
Example: Tree Search

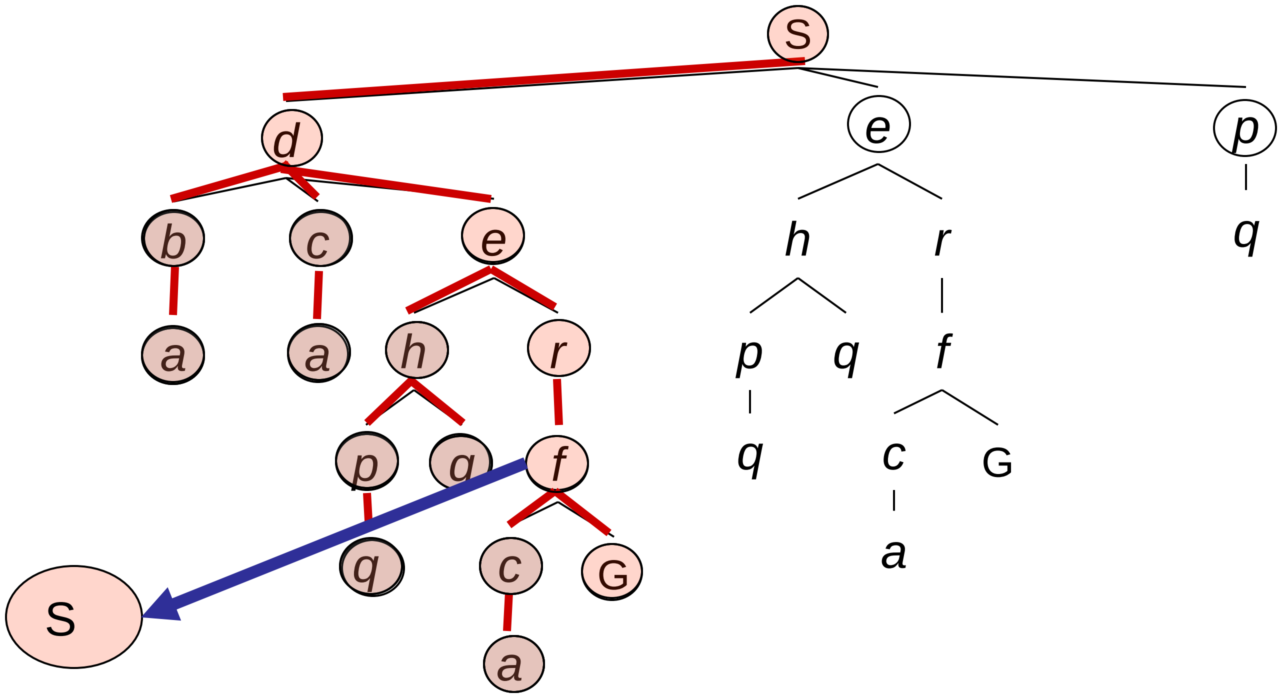


Depth-First Search

Strategy: expand a deepest node first

Implementation: Fringe is a LIFO stack





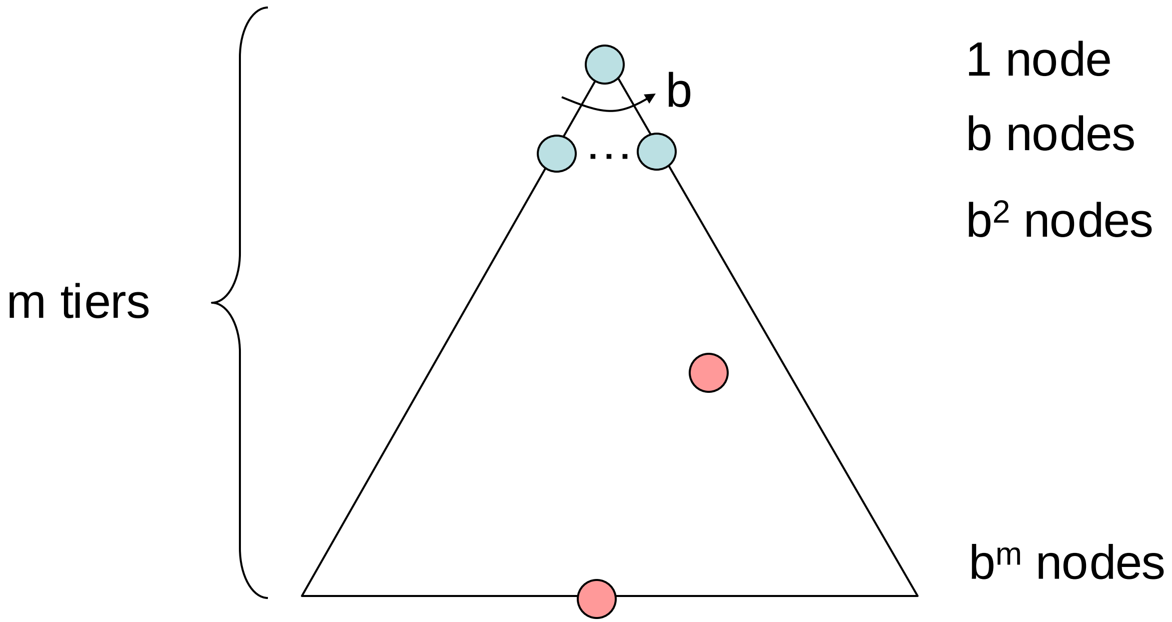
We do NOT recognize that the node "a" has been seen before. In fact, we have no concept of "memory" so, we don't even know that we have not seen the node a before.

We do however not loop in the same path, as we do have the LIFO stack, and we don't add the same node to the LIFO stack over and over again. (Example; We recognize the node S, because it is currently in the LIFO stack, and we don't run into infinite loops, while we do have some repetitions.)

Search Algorithm Properties

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

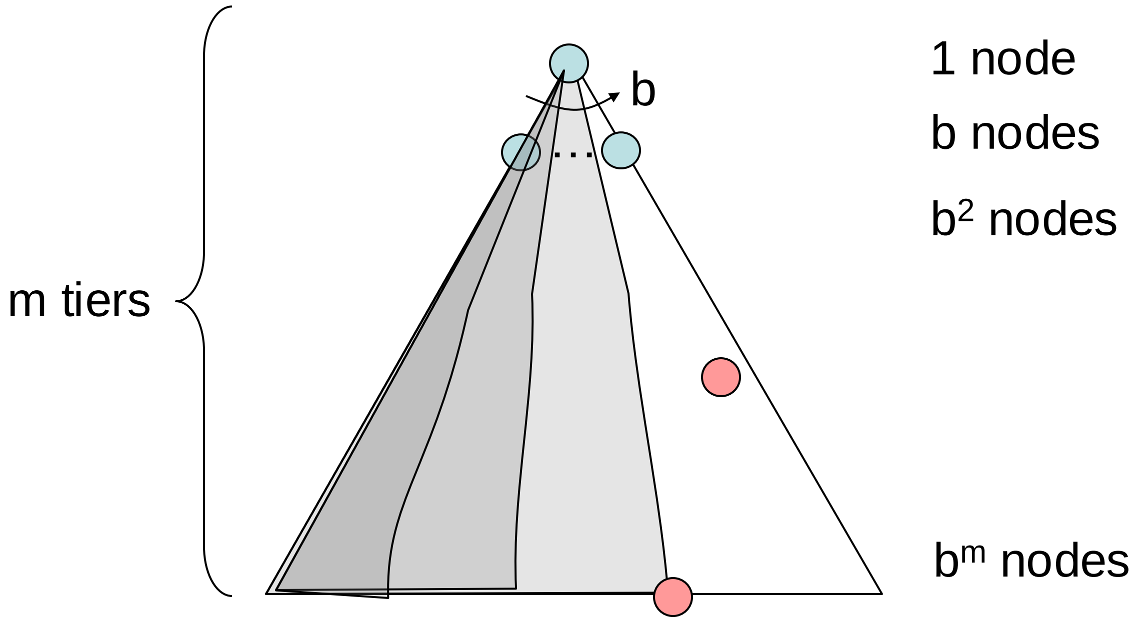
- Search tree:
  - b is the branching factor
  - m is the maximum depth
  - solutions at various depths



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

Depth-First Search (DFS) Properties

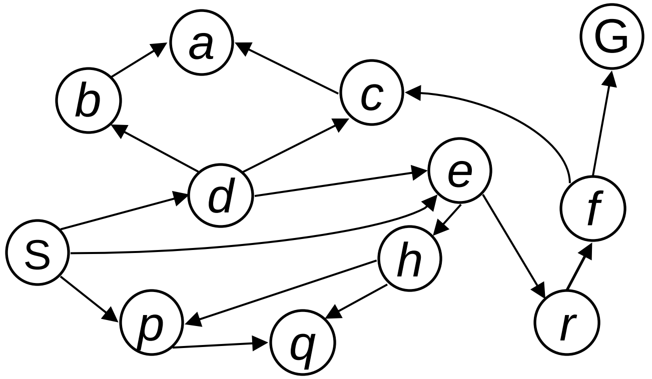
- What nodes DFS expand?
  - Some left prefix of the tree.
  - Could process the whole tree!
  - If m is finite, takes time $O(b^m)$
- How much space does the fringe take?
  - Only has siblings on path to root, so O(bm)
- Is it complete?
  - m could be infinite, so only if we prevent cycles
- Is it optimal?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

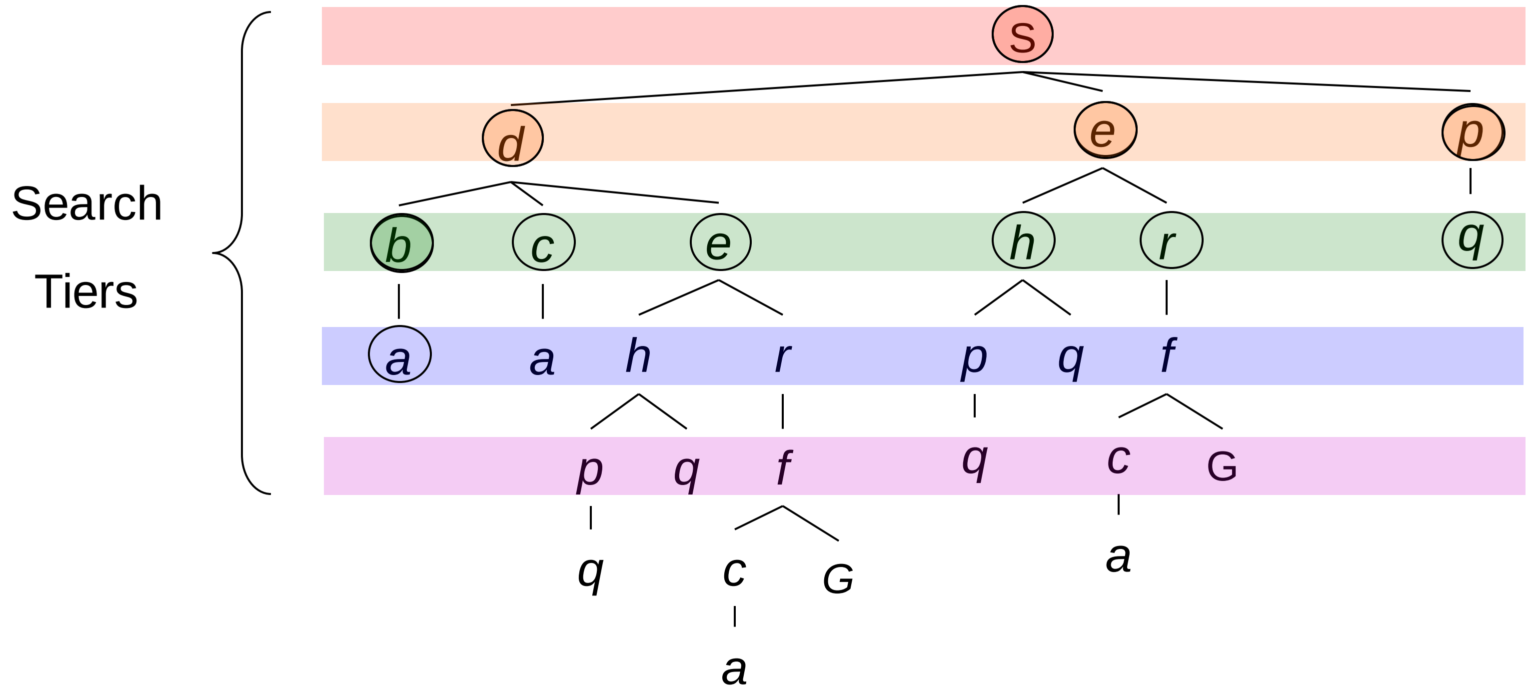


Breadth-First Search

Strategy: expand a shallowest node first

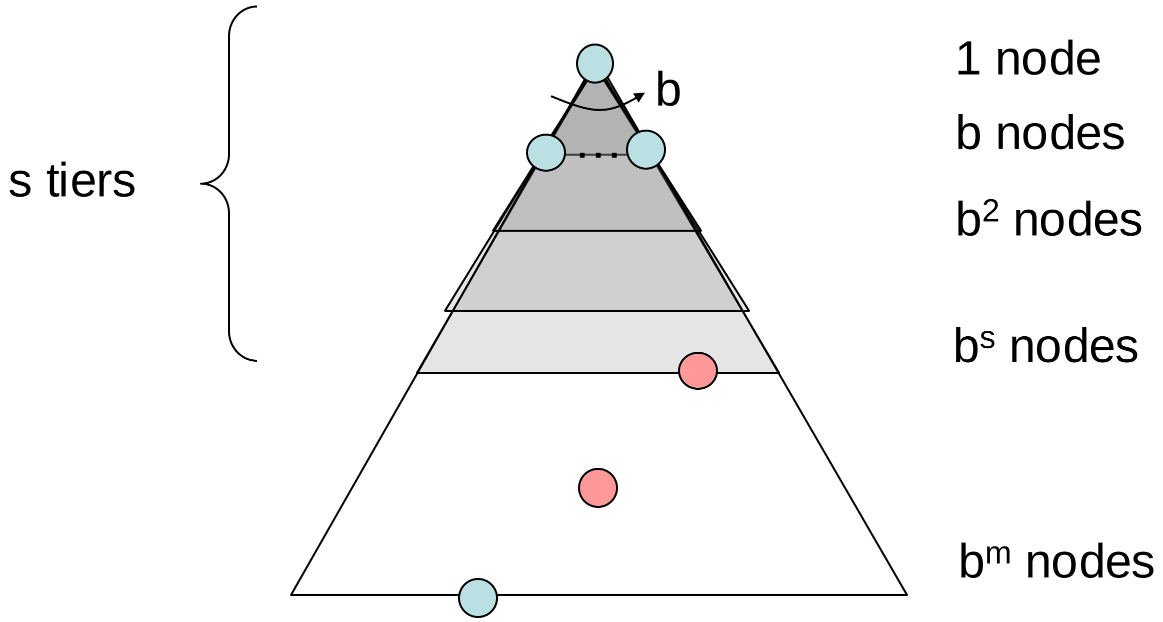
Implementation: Fringe is a FIFO queue





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)

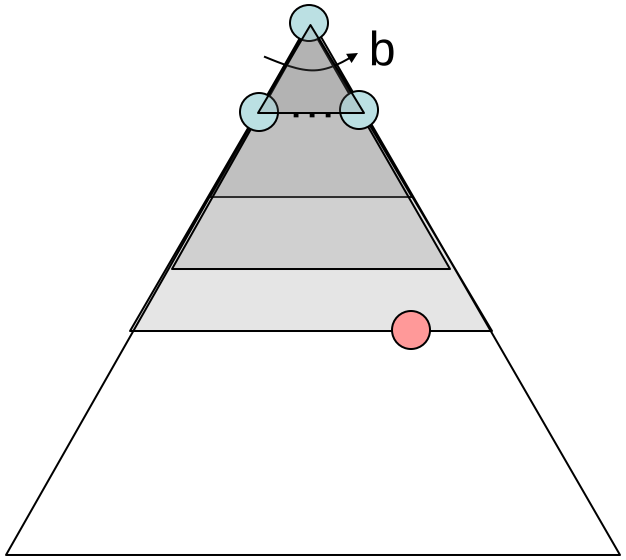


Quiz: DFS vs. BFS

- When will BFS outperform DFS?
- When will DFS outperform BFS?

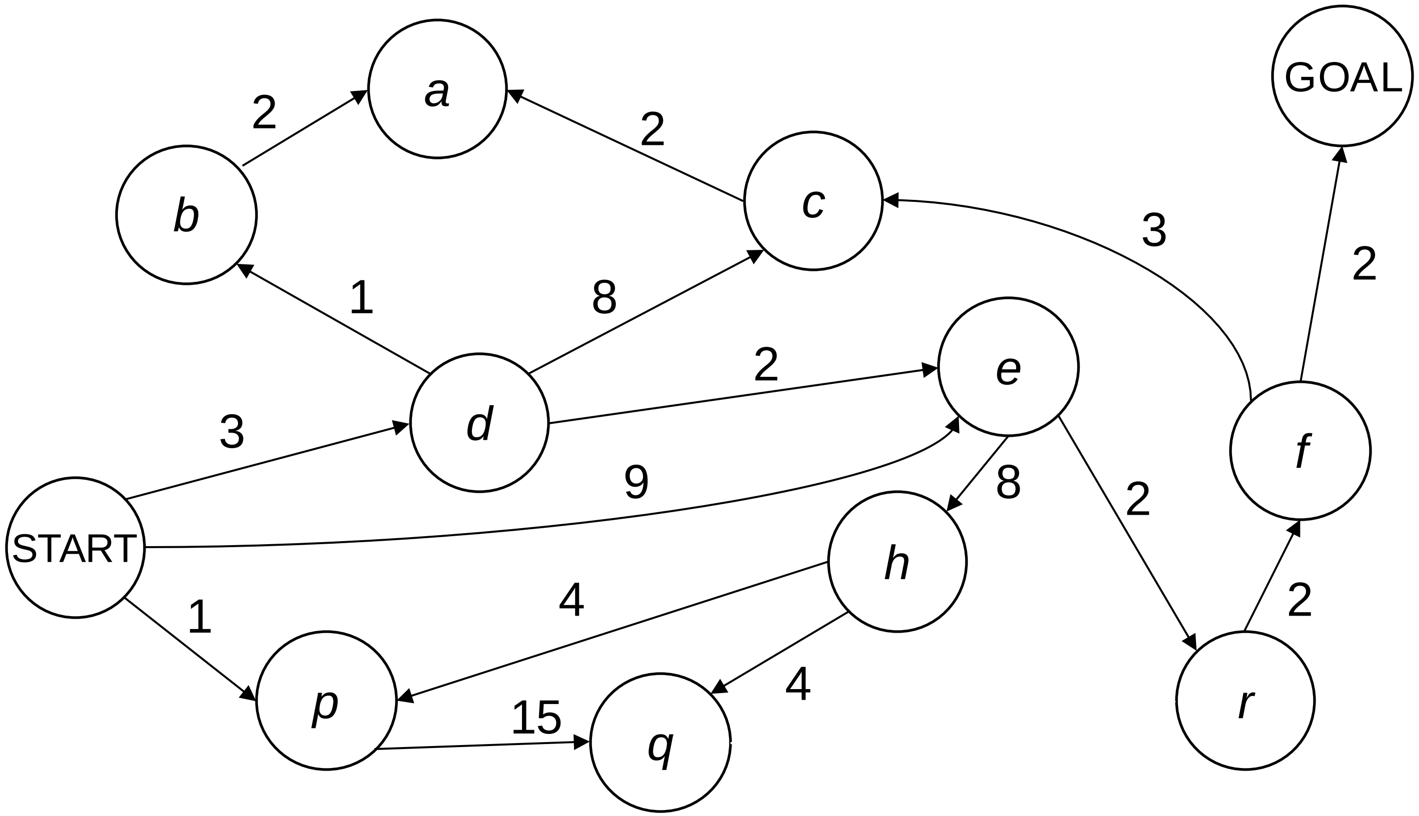
Iterative Deepening

- 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!

Let’s Incorporate Edge Costs into Search

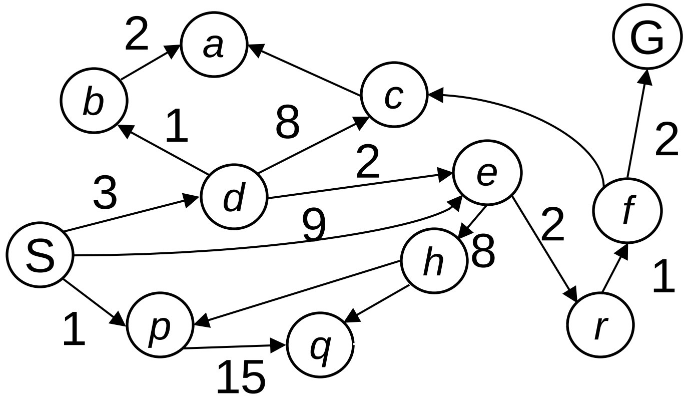


BFS finds the shortest path in terms of number of actions.
It does not find the least-cost path. We will now cover
a similar algorithm which does find the least-cost path.

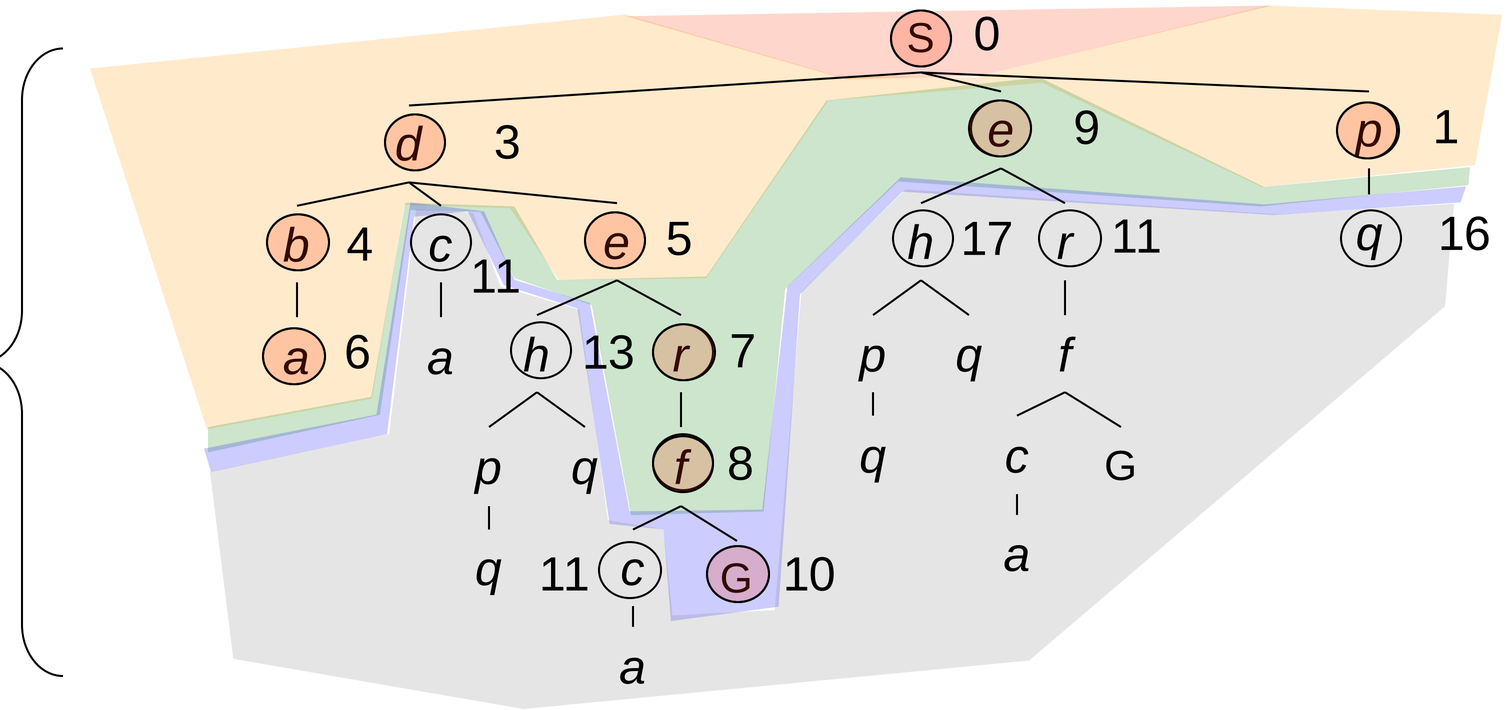
Uniform Cost Search

Strategy: expand a cheapest node first:

Fringe is a priority queue (priority: cumulative cost)

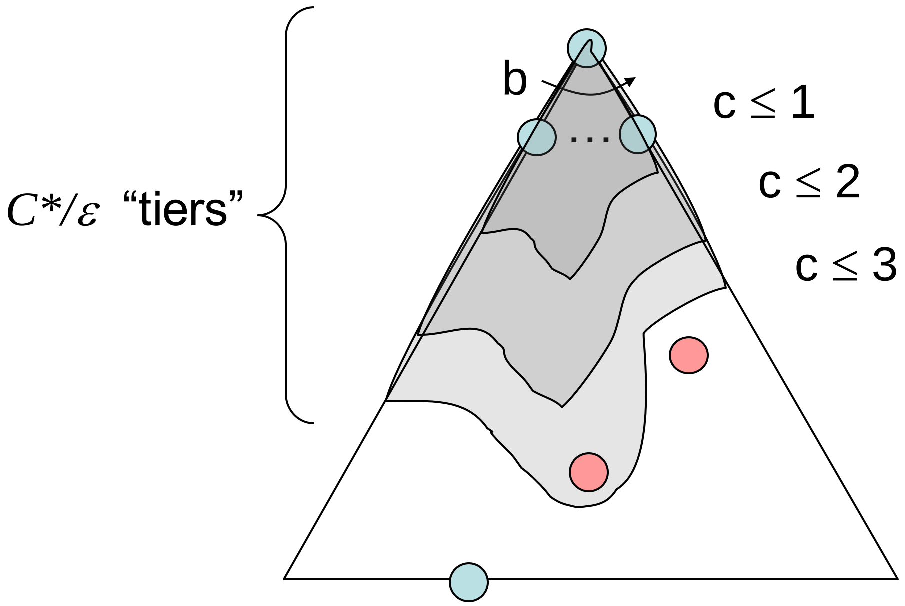


Cost contours



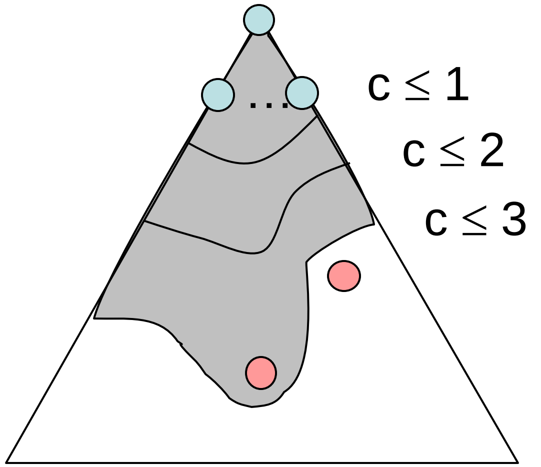
Uniform Cost Search (UCS) Properties

- 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! (Proof next lecture via A*)



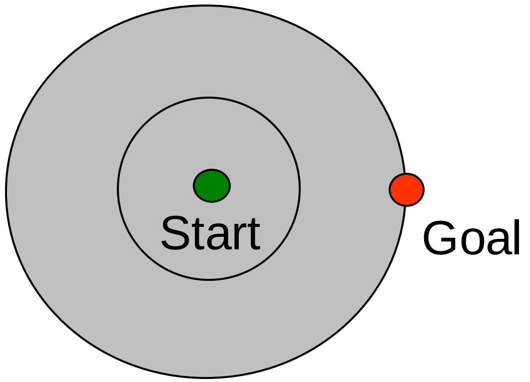
Uniform Cost Issues

- 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



- We'll fix that soon!

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

Some More Toy Problems

- N Puzzle
- Knuth's Factorial, Square Root and Floor
  - Start with 3 and get to 4 by applying these operations.
    - 3! = 6
    - 6! = 720
    - Sqrt(720) = 26.7
    - Floor = 26
    - Sqrt = 5.x
    - Floor = 5
    - Etc..

Course Outline

Other AI Topics (Not included in this class):

- Robotics
- NLP
- Machine Learning
- Big Data
- AR/VR
- Speech Synthesis