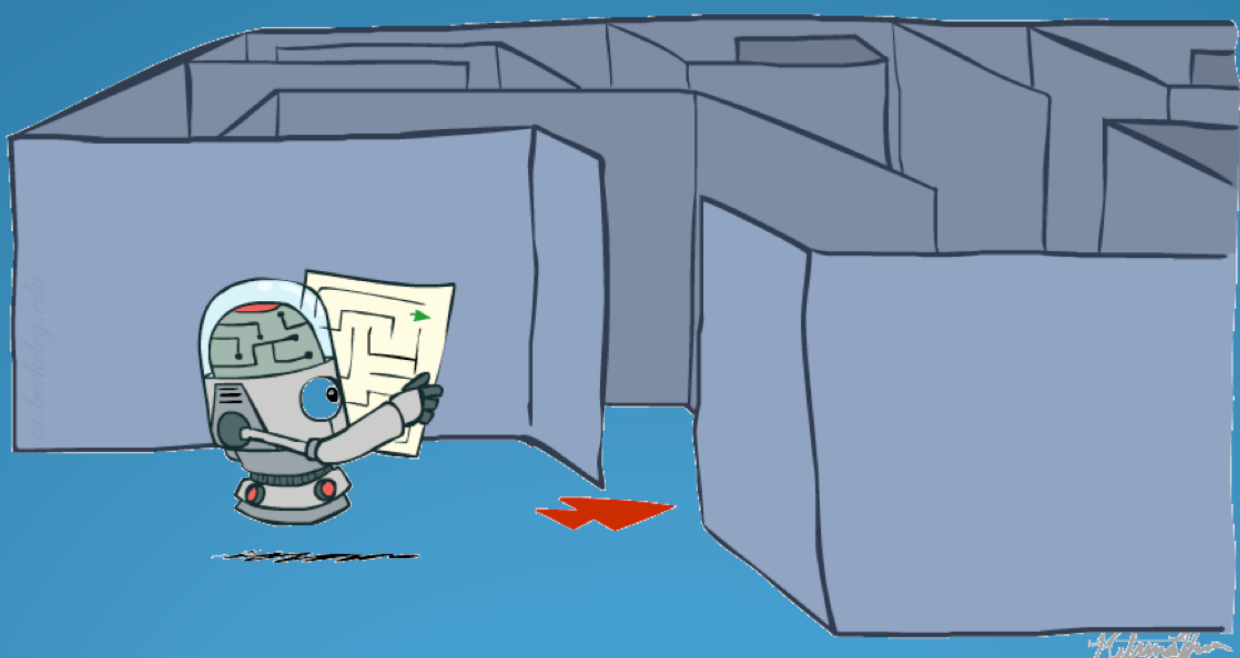
Artificial Intelligence Search

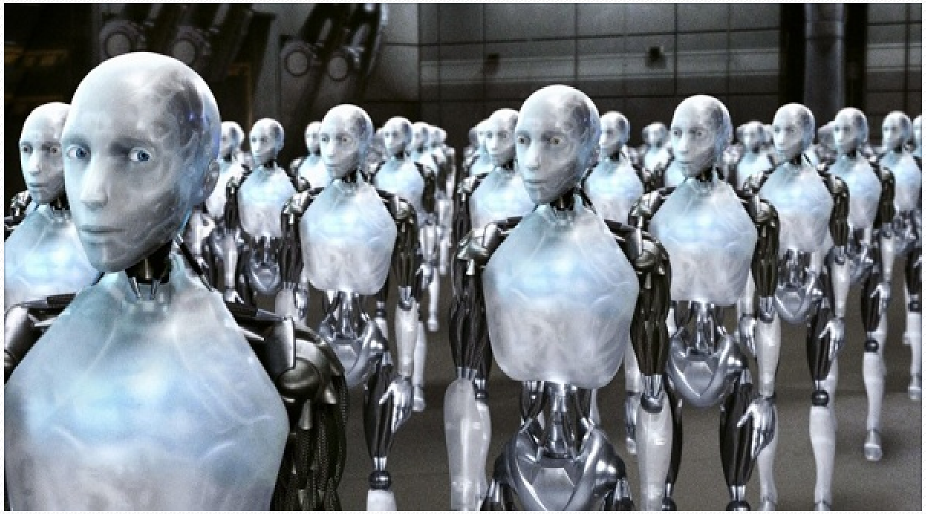
Lecture 2



Search Problem

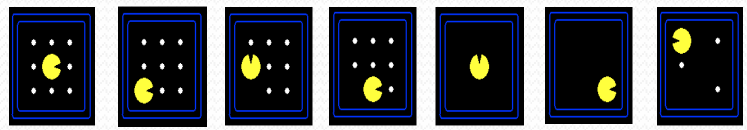
Search is a problem-solving technique to explores

successive stages in problem-solving process.

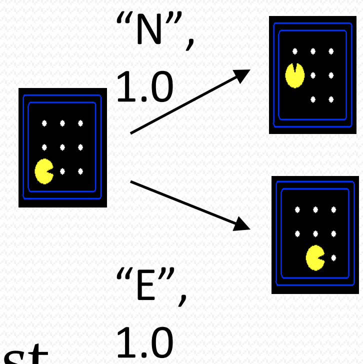


Search Problems

- A search problem consists of:
  - A state space



  - A successor function (with actions, costs)

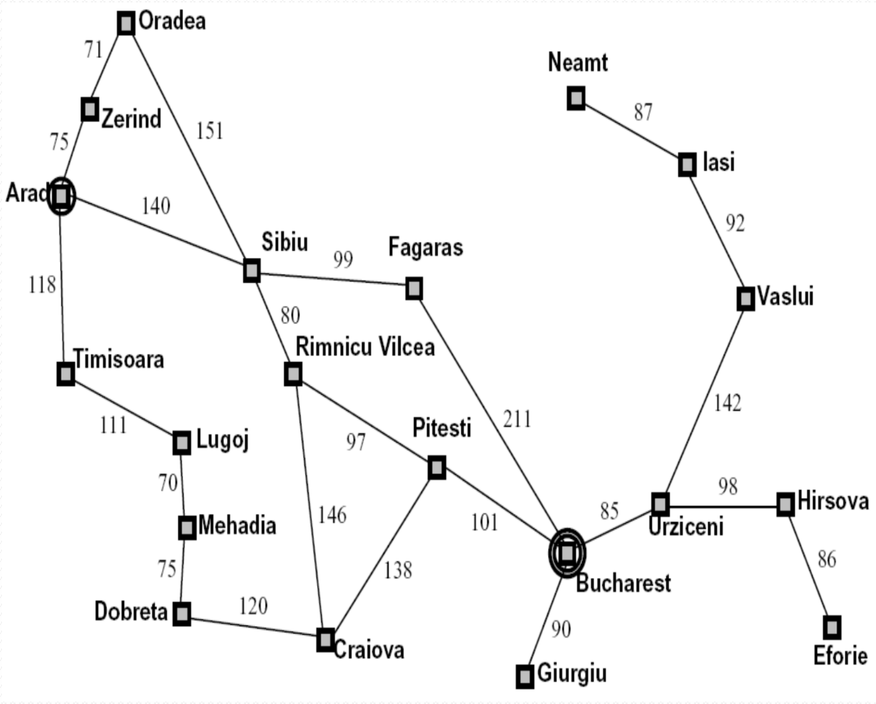


  - A start state and a goal test
- A solution is a sequence of actions (a plan) which transforms the start state to a goal state

State Space Search

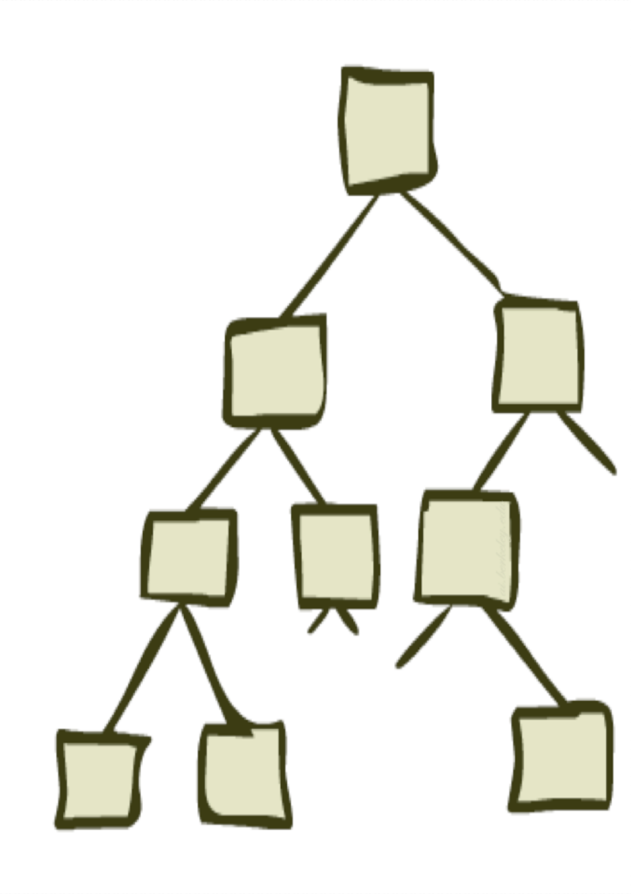
One tool to analyze the search space is to represent it as space graph, so by use graph theory we analyze the problem and solution of it.

Example: Traveling in Romania



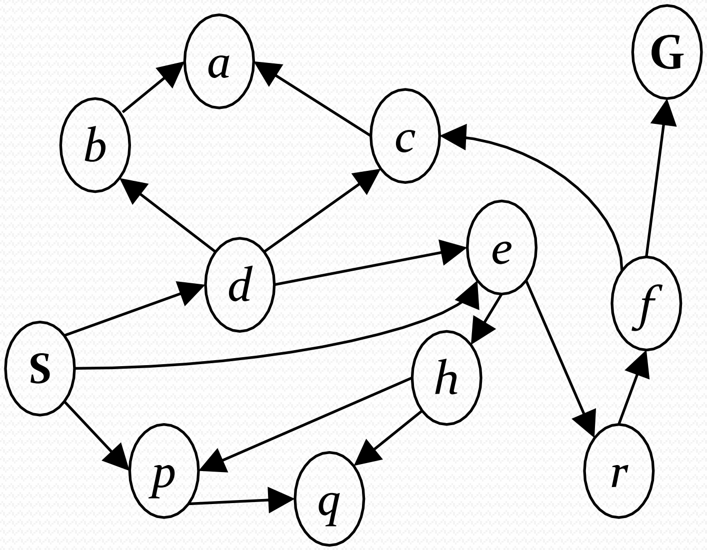
- State space:
  - Cities
- Successor function:
  - Roads: Go to adjacent city with cost = distance
- Start state:
  - Arad
- Goal test:
  - Is state == Bucharest?
- Solution?

State Space Graphs and Search Trees



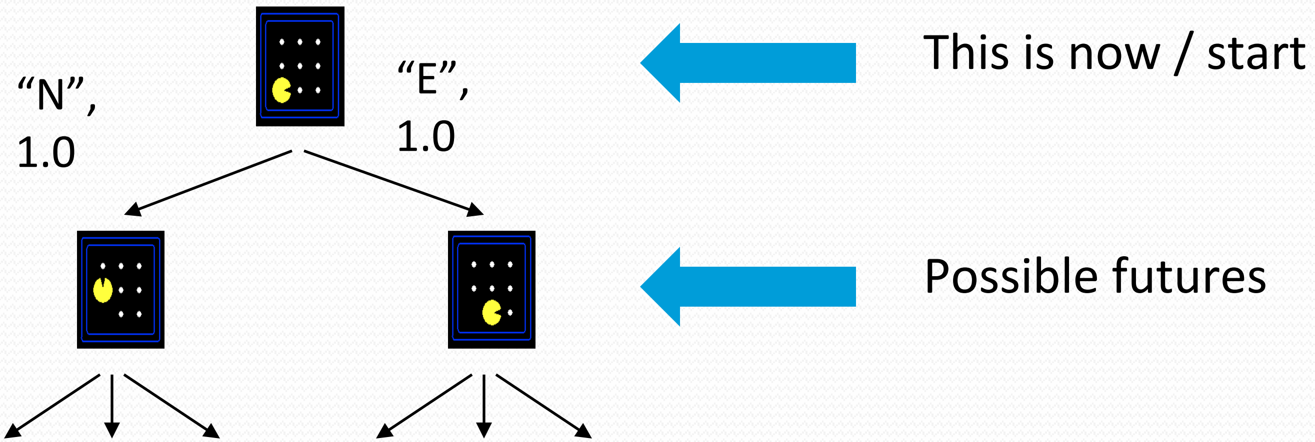
State Space Graphs

- 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 search graph, each state occurs only once!



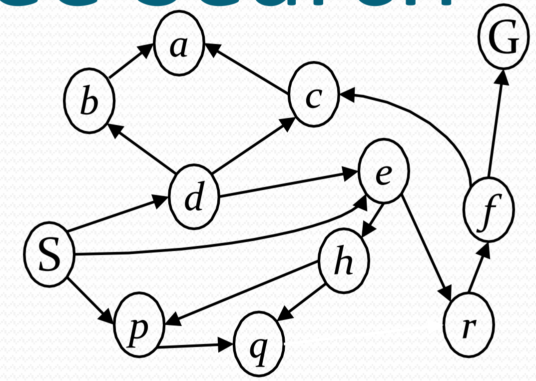
Tiny search graph for a tiny search problem

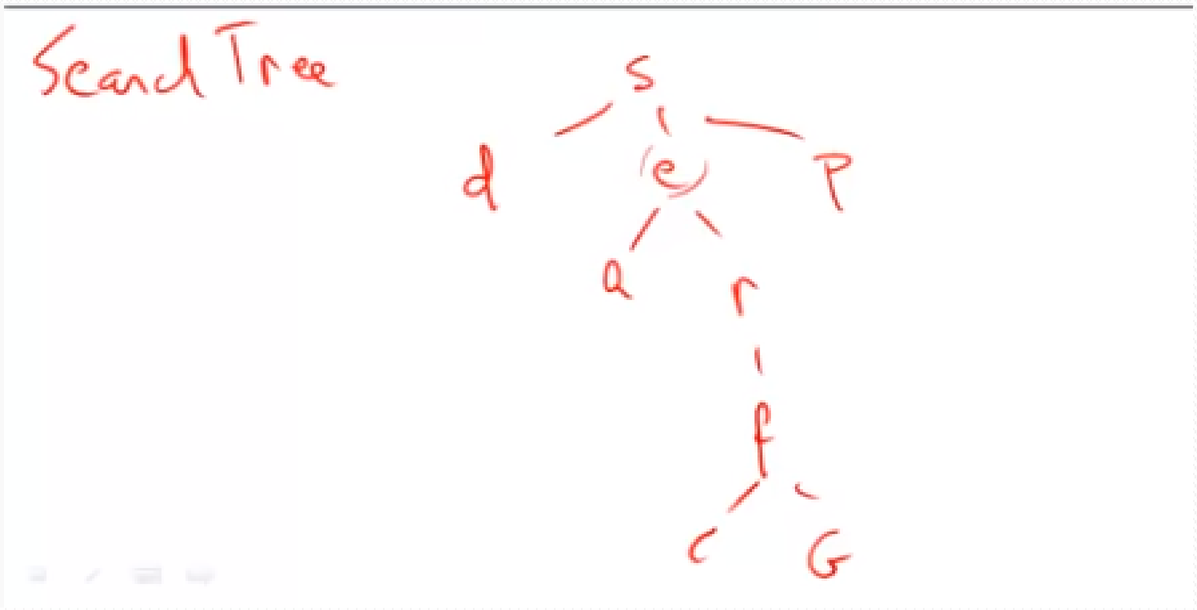
Search Trees



- 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

Example: Tree Search





Algorithm types

- There are two kinds of search algorithm
  - ***Complete***
    - guaranteed to find solution or prove there is none
  - ***Incomplete***
    - may not find a solution even when it exists
    - often more efficient (or there would be no point)

Comparing Searching Algorithms: Will it find a solution? the best one?

Def. : A search algorithm is ***complete*** if whenever there is at least one solution, the algorithm **is guaranteed to find it** within a finite amount of time.

Def.: A search algorithm is **optimal** if when it finds a solution, it is **the best one**

Comparing Searching Algorithms: Complexity

Branching factor b of a node is the number of arcs going out of the node

Def.: The ***time complexity*** of a search algorithm is

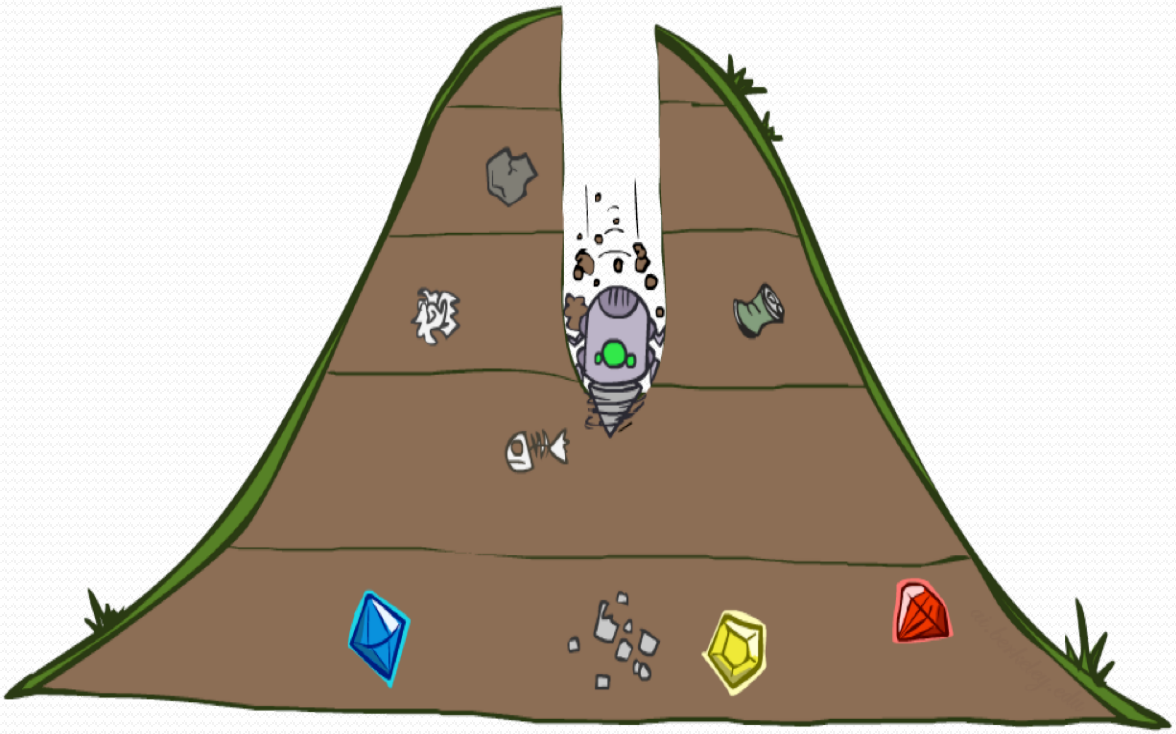
the **worst- case** amount of time it will take to run, expressed in terms of

- *maximum path length* ***m***
- *maximum branching factor* ***b***.

Def.: The **space complexity** of a search algorithm is

the **worst-case** amount of memory that the algorithm will use (i.e., the maximum number of nodes on the frontier), also expressed in terms of *m* and *b*.

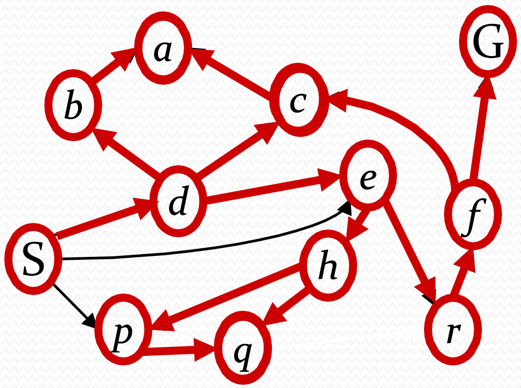
Depth-First Search

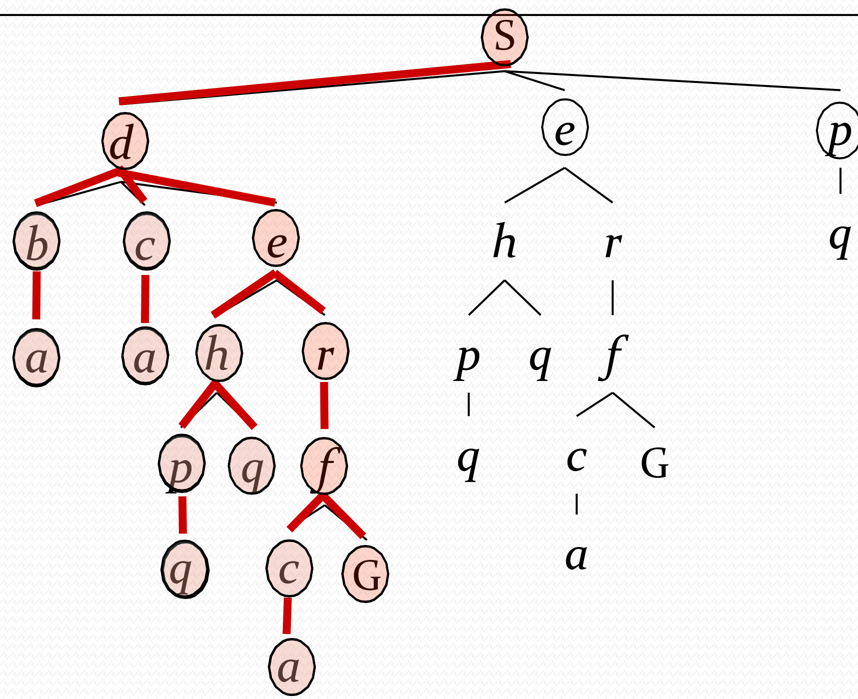


Depth-First Search

Strategy: expand a deepest node first

Implementation: Fringe is a LIFO stack





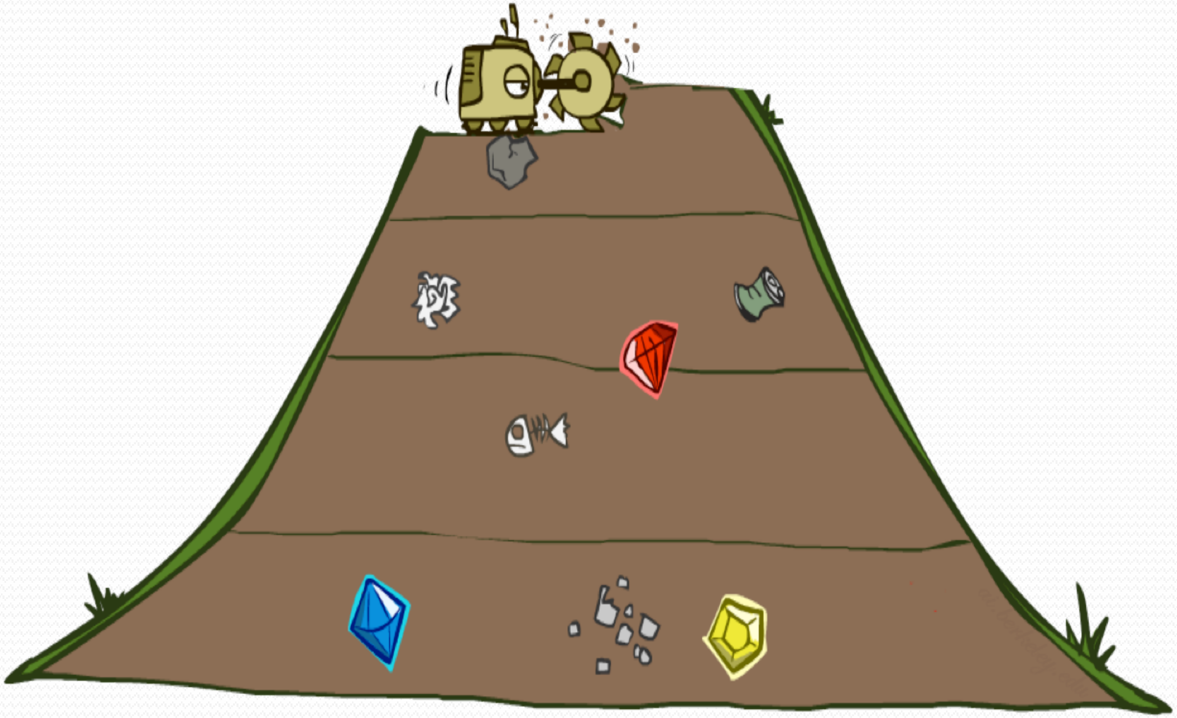
Search Algorithm Properties

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

Depth-First Search (DFS) Properties

- What nodes DFS expand?
  - Some left prefix of the tree.
  - Could process the whole tree!
- 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

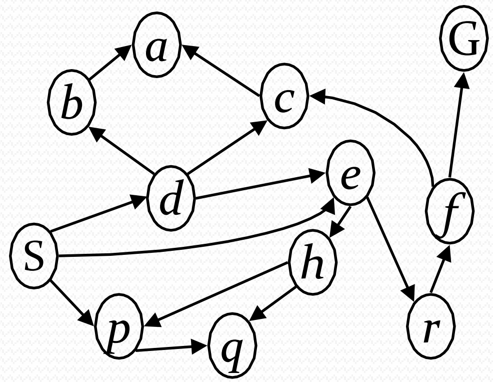
Breadth-First Search

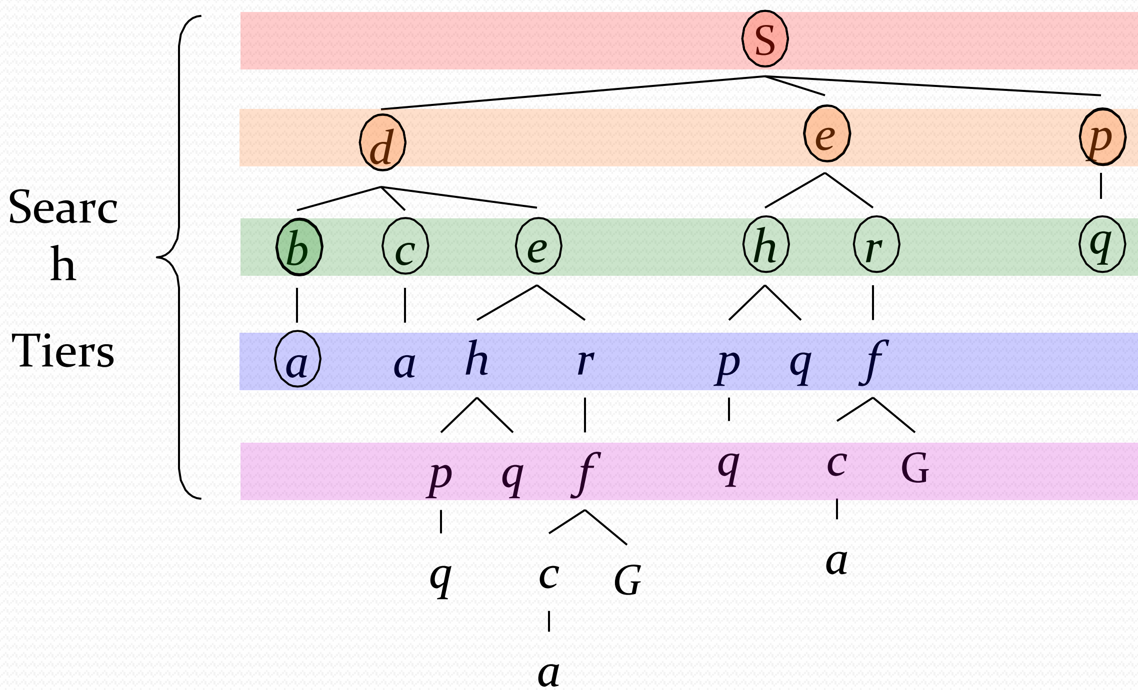


Breadth-First Search

Strategy: expand a shallowest node first

Implementation: Fringe is a FIFO queue

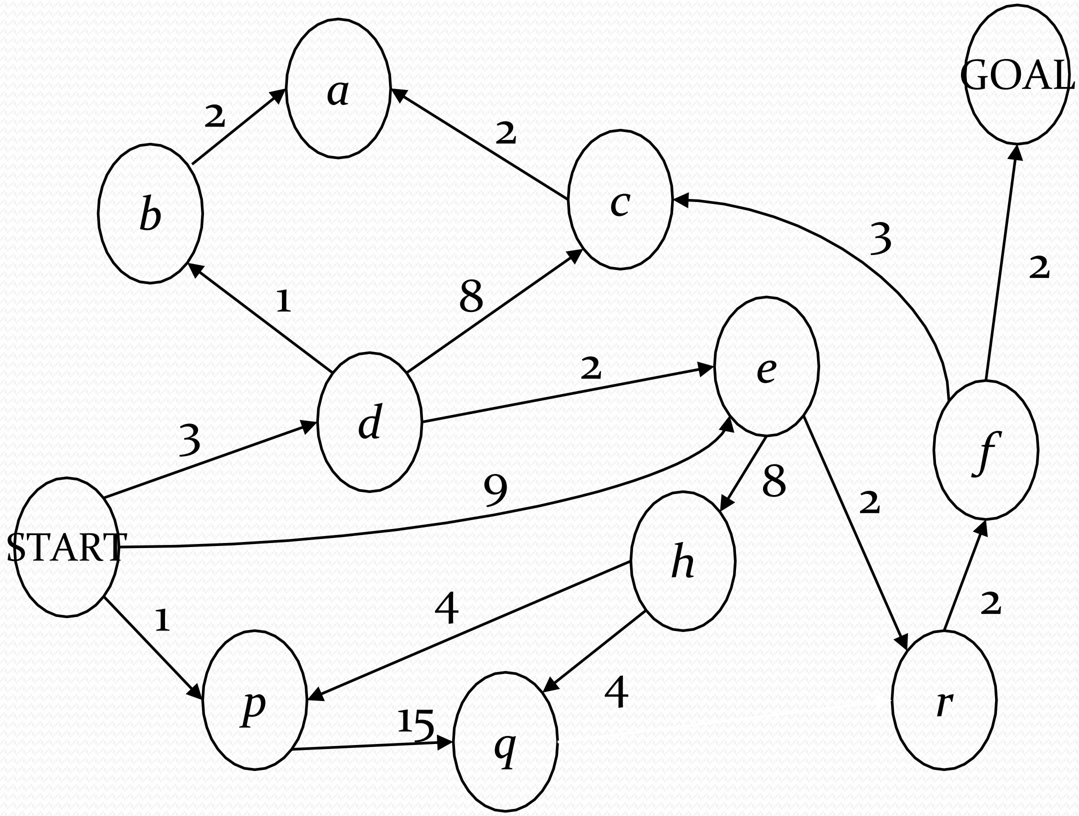




Breadth-First Search (BFS) Properties

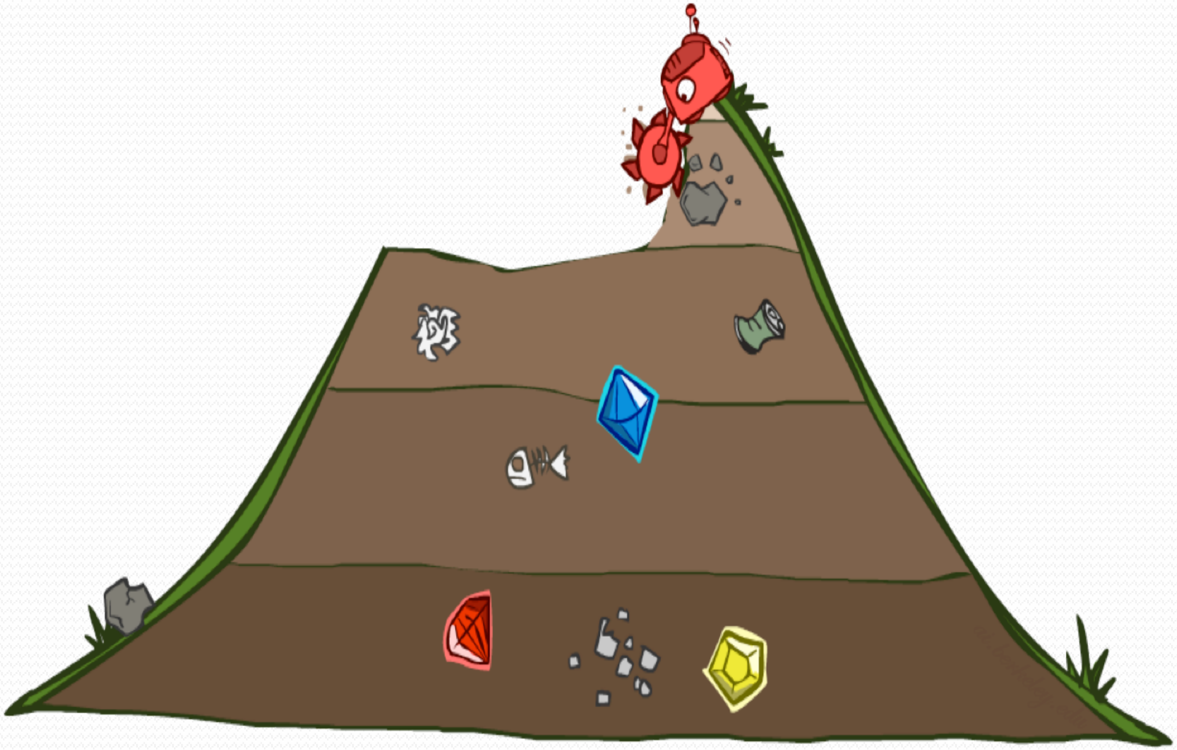
- What nodes does BFS expand?
  - Processes all nodes above shallowest solution
- Is it complete?
  - s must be finite if a solution exists, so yes!
- Is it optimal?
  - Only if costs are all 1 (more on costs later)

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.

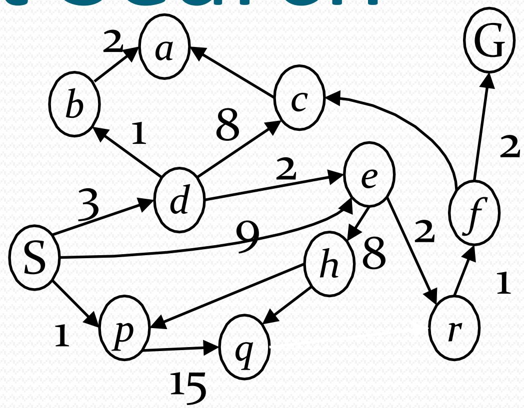
Uniform Cost Search



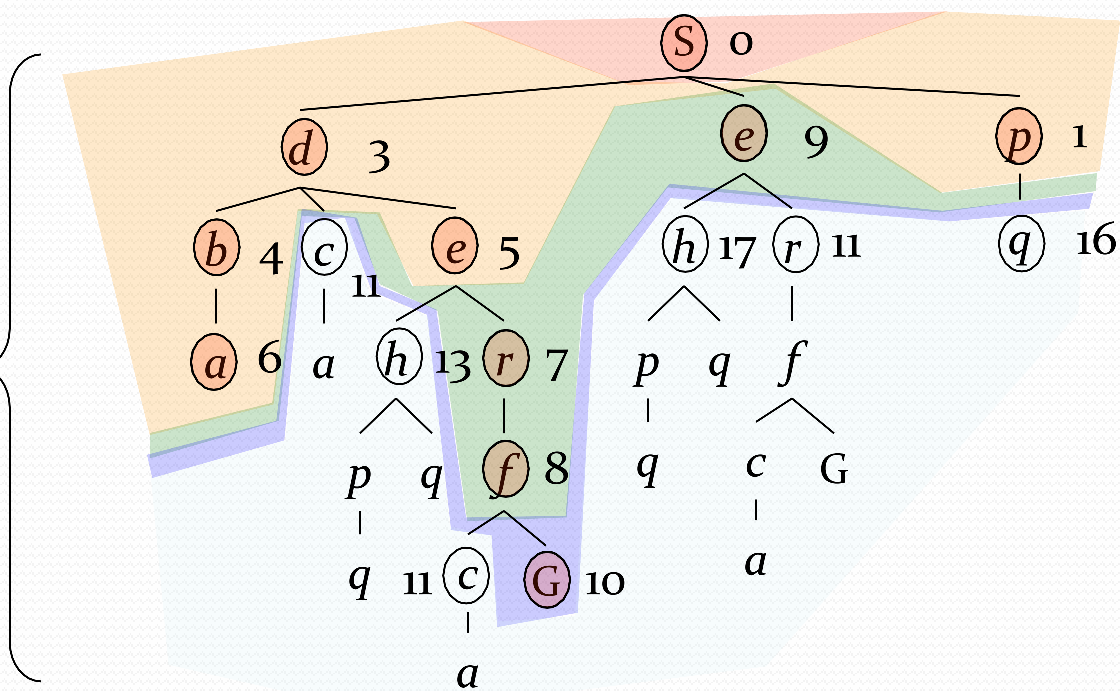
Uniform Cost Search

Strategy: expand a cheapest node first:

Fringe is a priority queue (priority: cumulative cost)



Cost contours

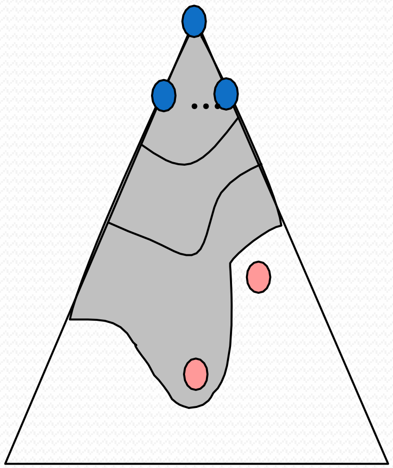


Uniform Cost Search (UCS) Properties

- What nodes does UCS expand?
  - Processes all nodes with cost less than cheapest solution!
- 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*)

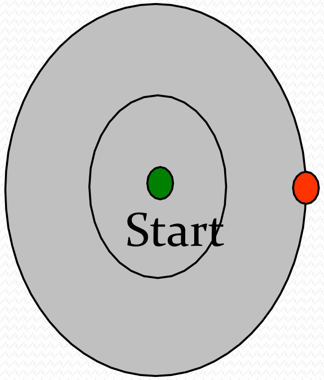
Uniform Cost Issues

- 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



Goal

- We'll fix that soon!