Part 1:
Solving Problems with Search

[Some slides adapted from Dan Klein and Pieter Abbeel]

Lecture plan

- Project 0: questions, comments?
- General tree and graph search algorithm
- Un-informed search algorithms
  - DFS (review)
  - BFS, IDFS
  - If time: USC

Project 0: Python 101. Due Friday 1/20

- Goal: gentle introduction to Python language, common data structures
- Algorithm:
    1. Download files
    2. Follow tutorial (step-by-step)
    3. Read provided starting code
    4. Fill in function bodies
    5. Run autograder
    6. If all tests passed – submit. Else: goto 3.
- Questions, stuck? Piazza!
- **Python clinic:** Wednesday 1/18, Lab. **2-6pm**.

Review + Python Reflex agent

- Example reflex agent Python code:

```
def ReflexVacuumAgent():
    "A reflex agent for the two-state vacuum environment. [Figure 2.8]"
    def program(percept):
        location, status = percept
        if status == 'Dirty':
            return 'Suck'
        elif location == loc_A:
            return 'Right'
        elif location == loc_B:
            return 'Left'
    return Agent(program)
```

https://github.com/aimacode/aima-python/blob/master/agents.py

Path finding: Amazon PrimeAir™

- First successful delivery in Cambridge, U.K in December
- Fill in the blank:

  Perf: ______

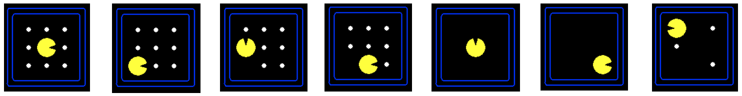
  Env: ______

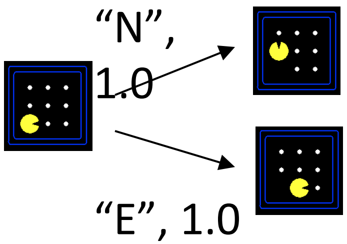
  Act: ______

  Sens: ______

https://www.youtube.com/watch?v=vNySOrI2Ny8

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

What’s in a State Space?

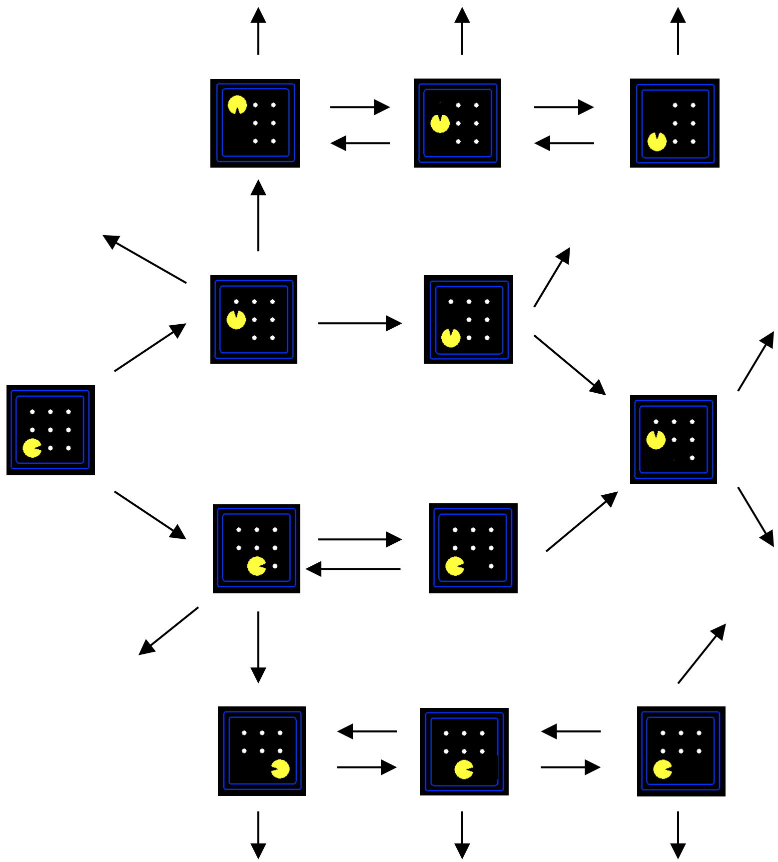
The world state includes every last detail of the environment

A search state keeps **only** the details needed for planning (abstraction).
Note: *Static info (e.g., road coordinates) do not need to be stored for each state*

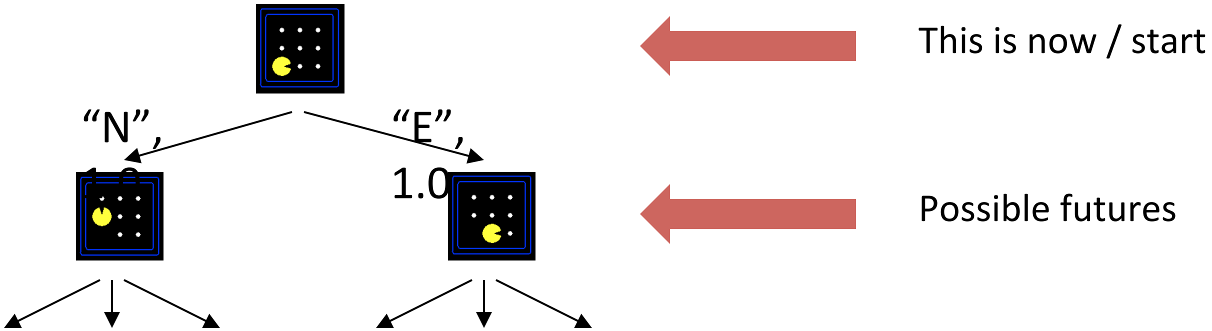
- Problem: Drone Path-Finding
  - States: (x,y,z) location, fuel, time, ….
  - Actions: Rotate **R**, **L**; Increase/decrease thrust: **I**, **D**; Drop package (**P**)
  - Successor: update position (X,Y,Z); fuel; time; …
  - Goal test: ?

State Space Graphs: 1

- **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!
  - Careful: if there are loops in this graph, keep track of visited states
- We can rarely build this full graph in memory (it's too big), but it's a useful idea



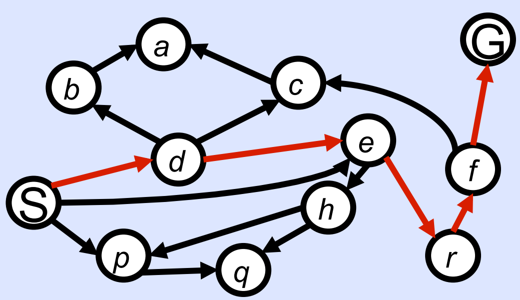
Search Trees



- 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 (paths from root to the state)
  - For most problems, we can never actually build the whole tree (too large)

State Space Graphs vs. Search Trees

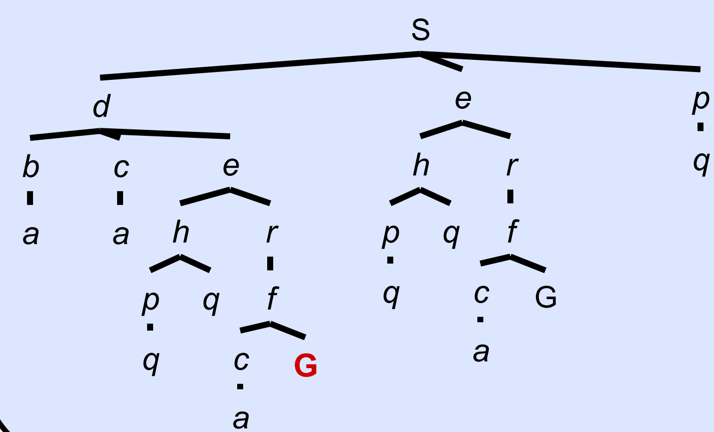
State Space Graph



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.

Search Tree



General Tree Search

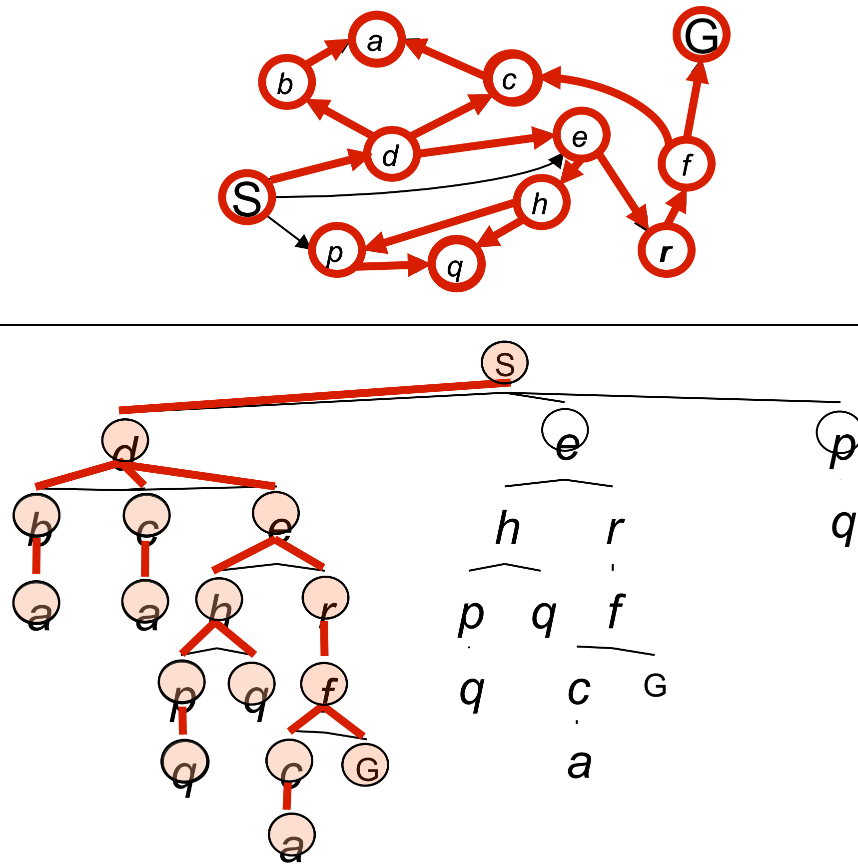
```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the 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 (add child nodes to fringe)
  - **Exploration strategy (which fringe nodes to explore?)**

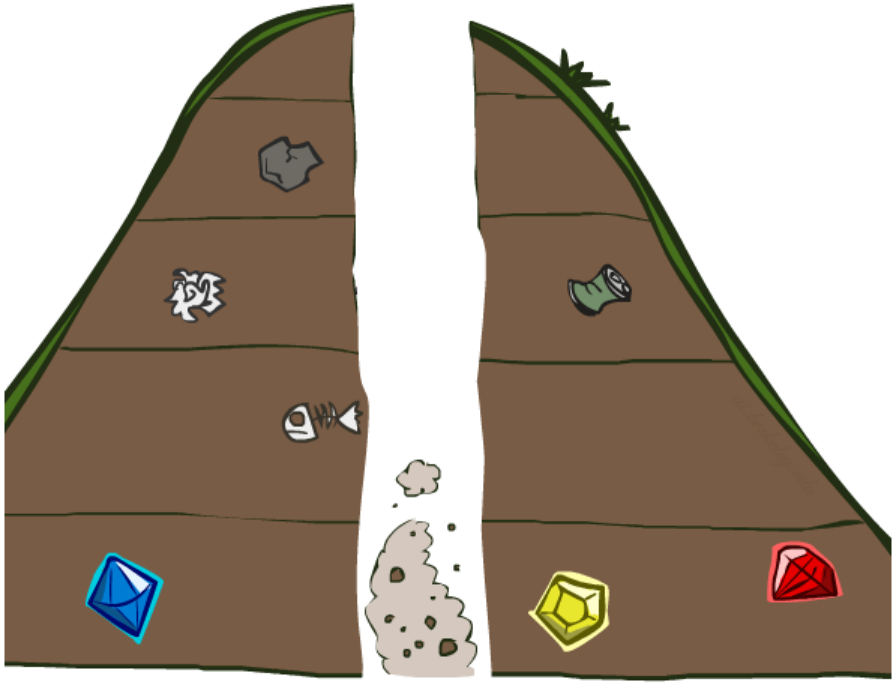
Depth-First Search

Strategy: expand a deepest node first

Implementation: Fringe is a LIFO stack



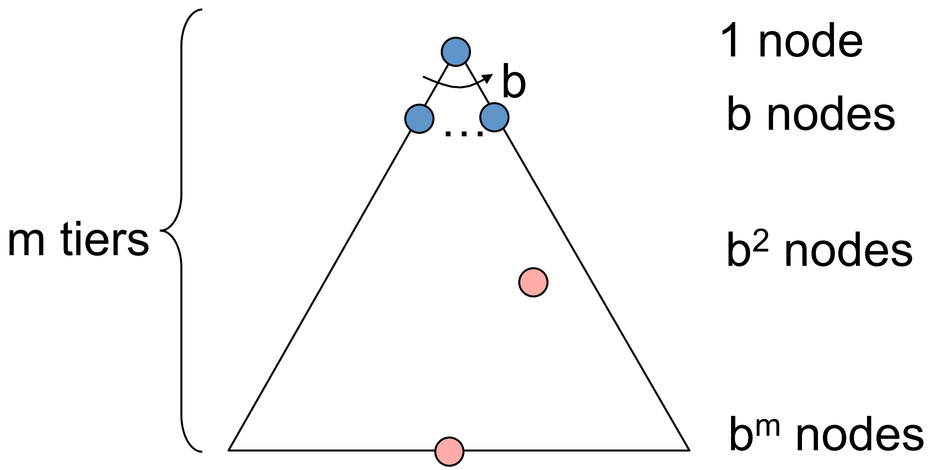
Search Algorithm Properties



Search Algorithm Properties

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

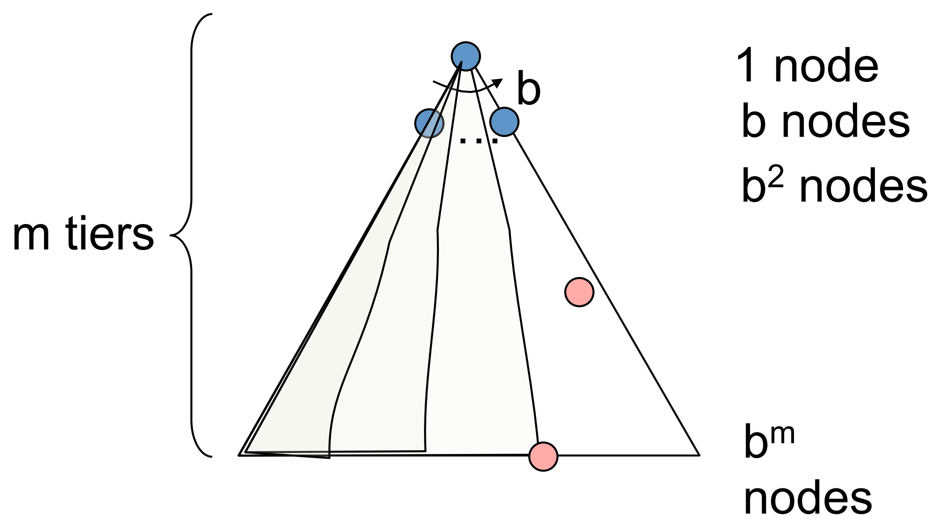
- Cartoon of search tree:
  - b is the branching factor
  - m is the maximum depth
  - solutions at various depths



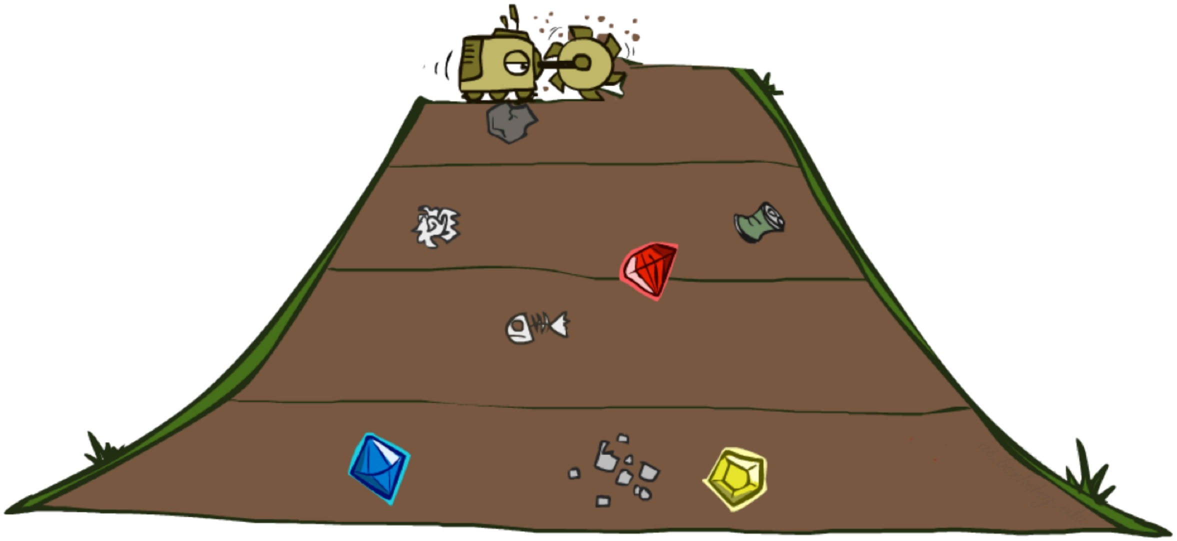
- Number of nodes in entire tree?
  - $1 + b + b^2 + ... b^m = O(b^m)$

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



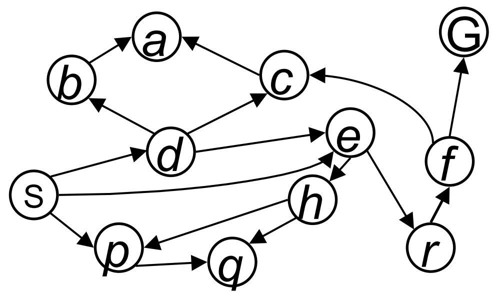
Breadth-First Search



Breadth-First Search

Strategy: expand a shallowest node first

Implementation: Fringe is a FIFO queue

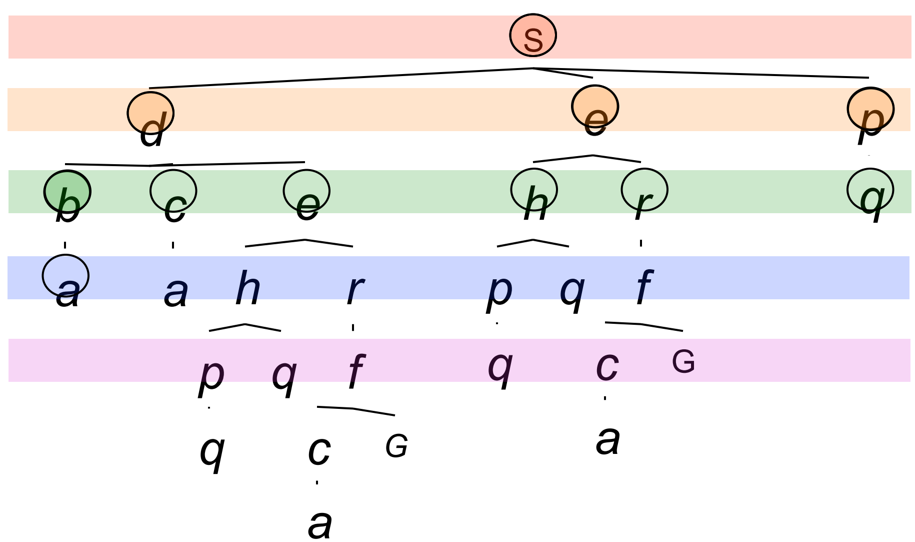


Fringe:

First in – first out

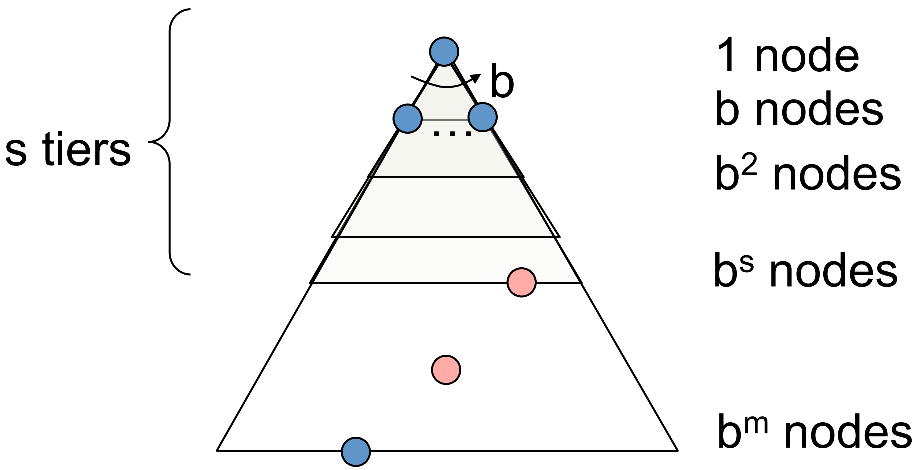
Search

Tiers



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)



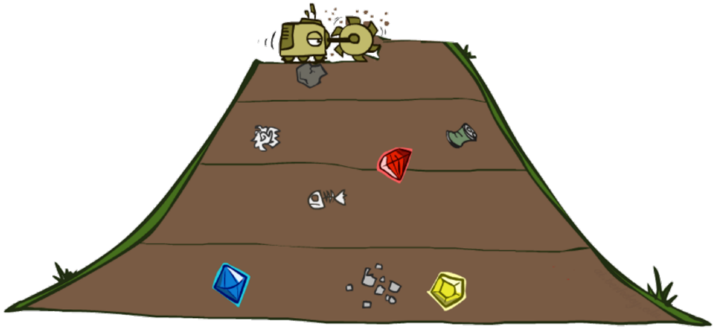
Memory a Limitation?

- Suppose:
  - 4 GHz CPU
  - 6 GB main memory
  - 100 instructions / expansion
  - 5 bytes / node

- 400,000 expansions / sec
  - Memory filled in 300 sec … 5 min

Remember: BFS needs to keep $O(b^d)$ states (fringe) in memory

Quiz: DFS vs BFS



Quiz: DFS vs BFS

- When will BFS outperform DFS?

- When will DFS outperform BFS?

Algorithm		Complete	Optimal	Time	Space
DFS	w/ Path Checking	Y	N	$O(b^m)$	$O(bm)$
BFS		Y	Y*	$O(b^d)$	$O(b^d)$

Comparisons

- When will BFS outperform DFS?

- When will DFS outperform BFS?

Algorithm		Complete	Optimal	Time	Space
DFS	w/ Path Checking	Y	N	$O(b^m)$	$O(bm)$
BFS		Y	Y*	$O(b^d)$	$O(b^d)$

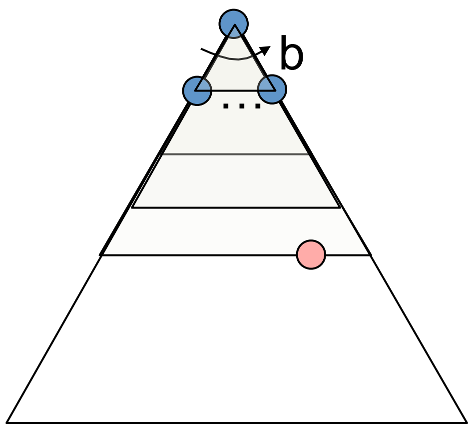
Video of Demo Maze Water DFS/BFS (part 1)

Video of Demo Maze Water DFS/BFS (part 2)

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!

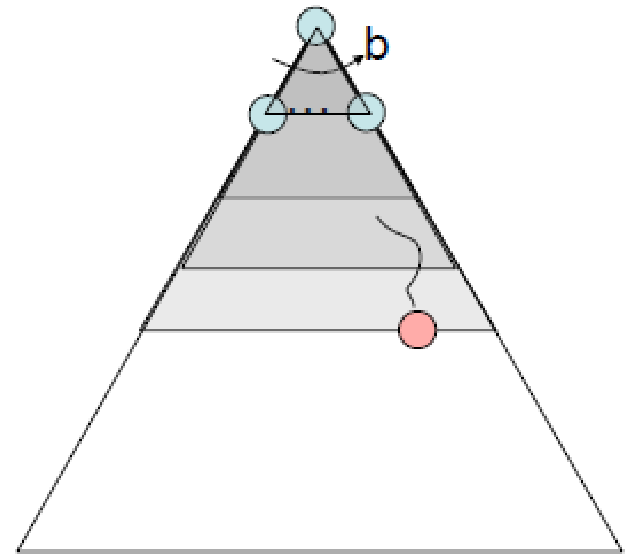


Iterative Deepening

Iterative deepening uses DFS as a subroutine:

1. Do a DFS which only searches for paths of length 1 or less.
2. If "1" failed, do a DFS which only searches paths of length 2 or less.
3. If "2" failed, do a DFS which only searches paths of length 3 or less.

….and so on.



Algorithm		Complete	Optimal	Time	Space
DFS	w/ Path Checking	Y	N	$O(b^m)$	$O(bm)$
BFS		Y	Y*	$O(b^d)$	$O(b^d)$
ID		Y	Y*	$O(b^d)$	$O(bd)$

Speed

Assuming 10M nodes/sec & sufficient memory

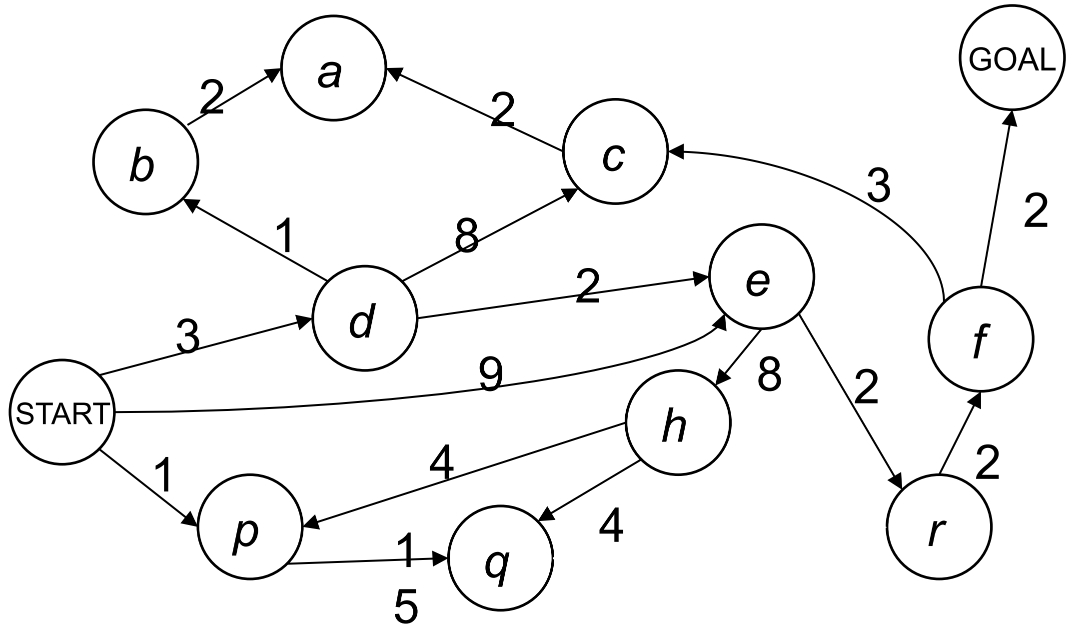
	BFS Nodes	BFS Time		Iter. Deep. Nodes	Iter. Deep. Time
8 Puzzle	10^5	.01 sec		10^5	.01 sec
2x2x2 Rubik's	10^6	.2 sec		10^6	.2 sec
15 Puzzle	10^{13}	6 days	1Mx	10^{17}	20k yrs
3x3x3 Rubik's	10^{19}	68k yrs	8x	10^{20}	574k yrs
24 Puzzle	10^{25}	12B yrs		10^{37}	10^{23} yrs

Why the difference?
- Rubik has higher branch factor
- 15 puzzle has greater depth

# of duplicates

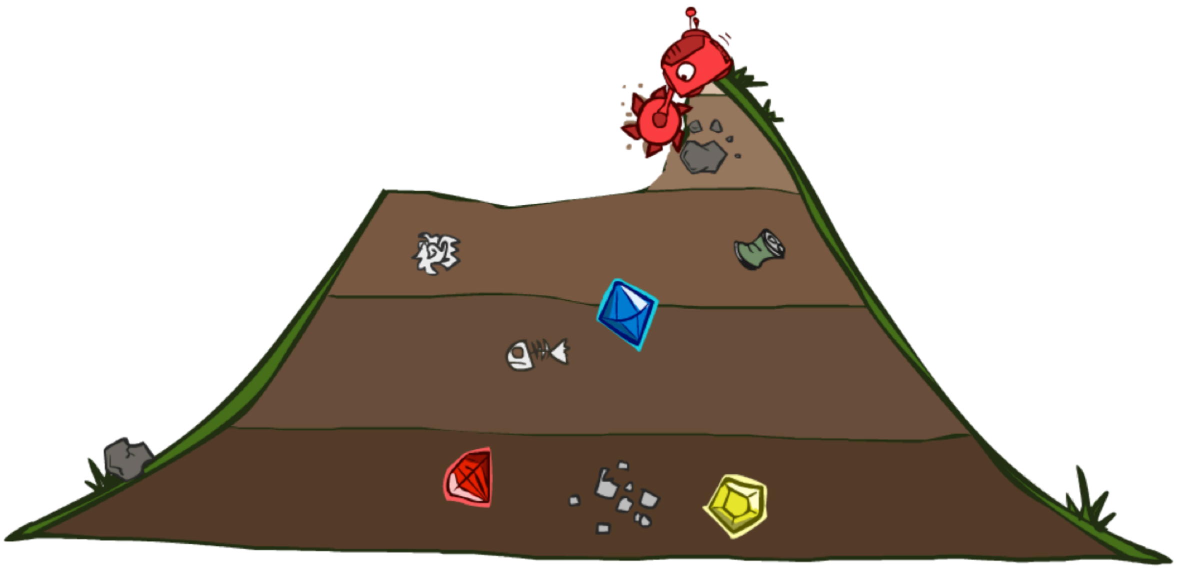
Slide adapted from Richard Korf presentation

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.

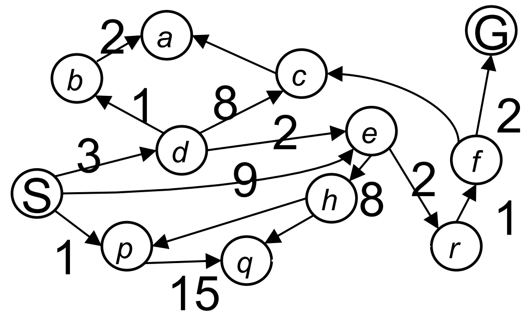
Uniform Cost Search

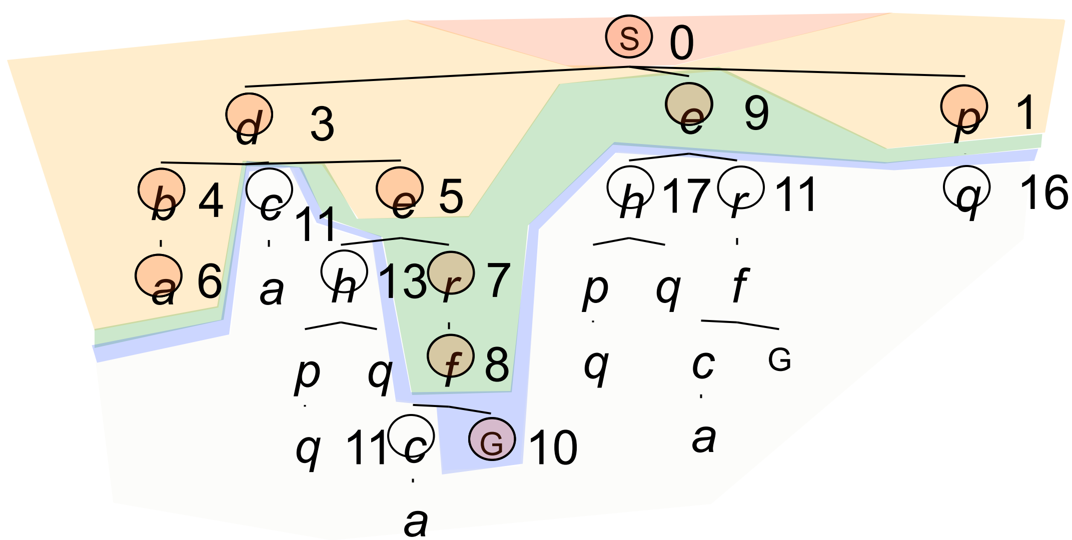


Uniform Cost Search

Strategy: expand a cheapest node first:

Fringe is ***a priority queue*** *(priority: cumulative cost)*

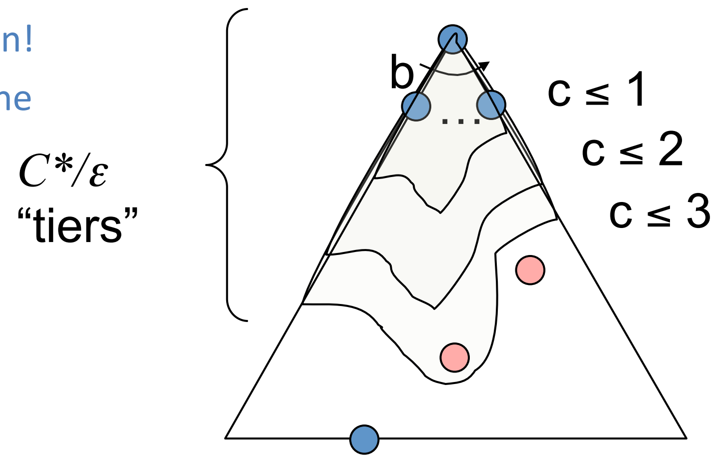




Cost contours

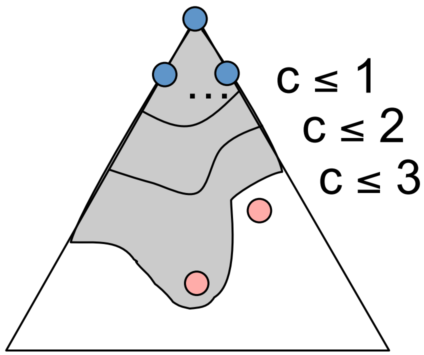
Uniform Cost Search (UCS) Properties

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



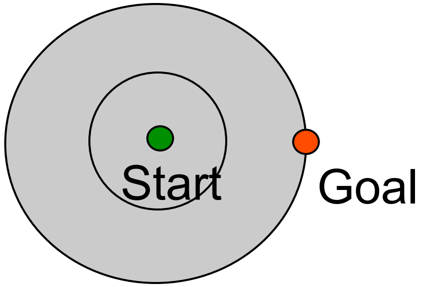
Uniform Cost Issues

- 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!

[Demo: empty grid UCS
[Demo: maze with deep/shallow water DFS/BFS/UCS]

Video of Demo Empty UCS

Video of Demo Maze with Deep/Shallow Water --- DFS, BFS, or UCS? (part 1)

Video of Demo Maze with Deep/Shallow Water --- DFS, BFS, or UCS? (part 2)

Video of Demo Maze with Deep/Shallow Water --- DFS, BFS, or UCS? (part 3)

To Do:

- Enroll in Piazza, check for updates!
- Do the readings (Chapter 3 in R&N)
- Finish Project 0 (posted on website and Piazza)
- Thursday: Project 1 will be assigned. hardness: 7
  - **Implication**: if you have not done Python tutorial and finished project 0 – do it now.