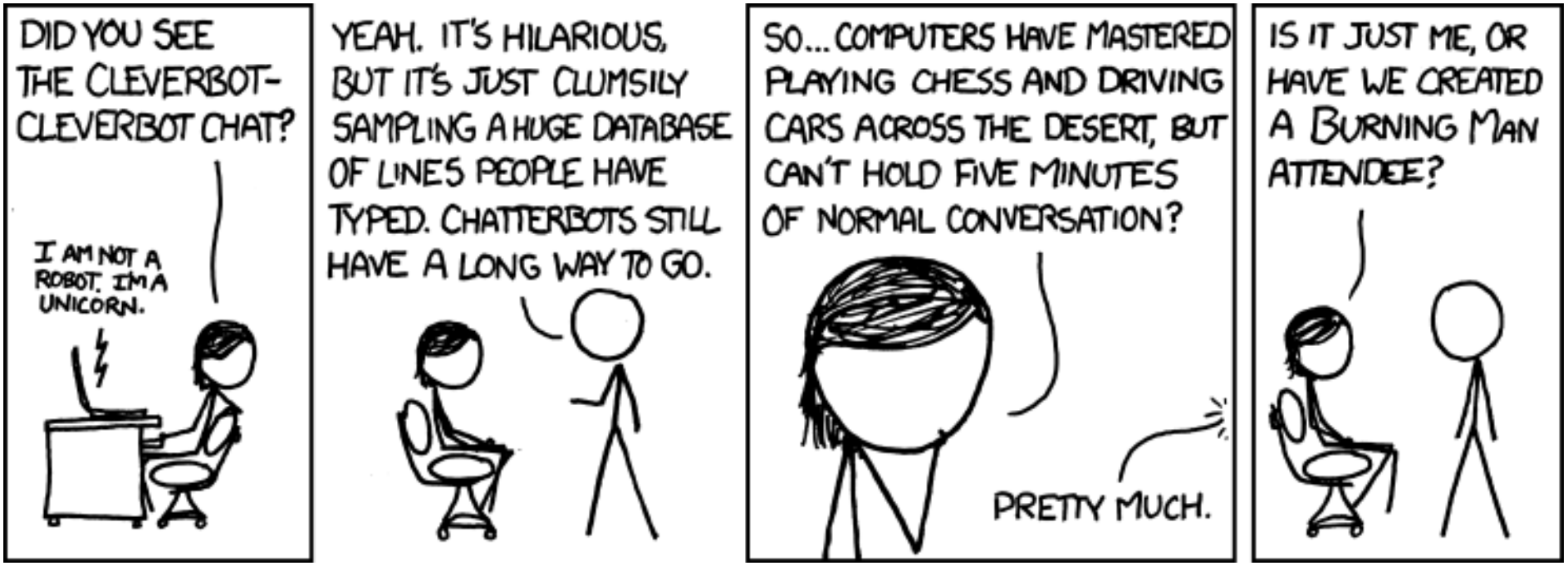
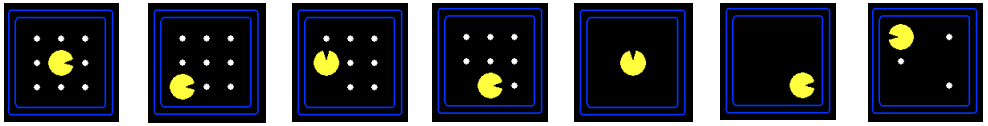
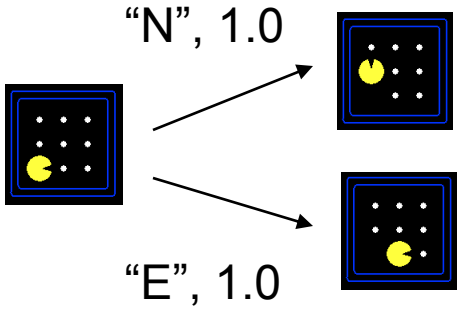

DID YOU SEE THE CLEVERBOT-CLEVERBOT CHAT?
I AM NOT A ROBOT. I'M A UNICORN.
YEAH. IT'S HILARIOUS, BUT IT'S JUST CLUMSILY SAMPLING A HUGE DATABASE OF LINES PEOPLE HAVE TYPED. CHATTERBOTS STILL HAVE A LONG WAY TO GO.
SO... COMPUTERS HAVE MASTERED PLAYING CHESS AND DRIVING CARS ACROSS THE DESERT, BUT CAN'T HOLD FIVE MINUTES OF NORMAL CONVERSATION?
PRETTY MUCH.
IS IT JUST ME, OR HAVE WE CREATED A BURNING MAN ATTENDEE?

Search Problems

- A search problem consists of:
  - A state space

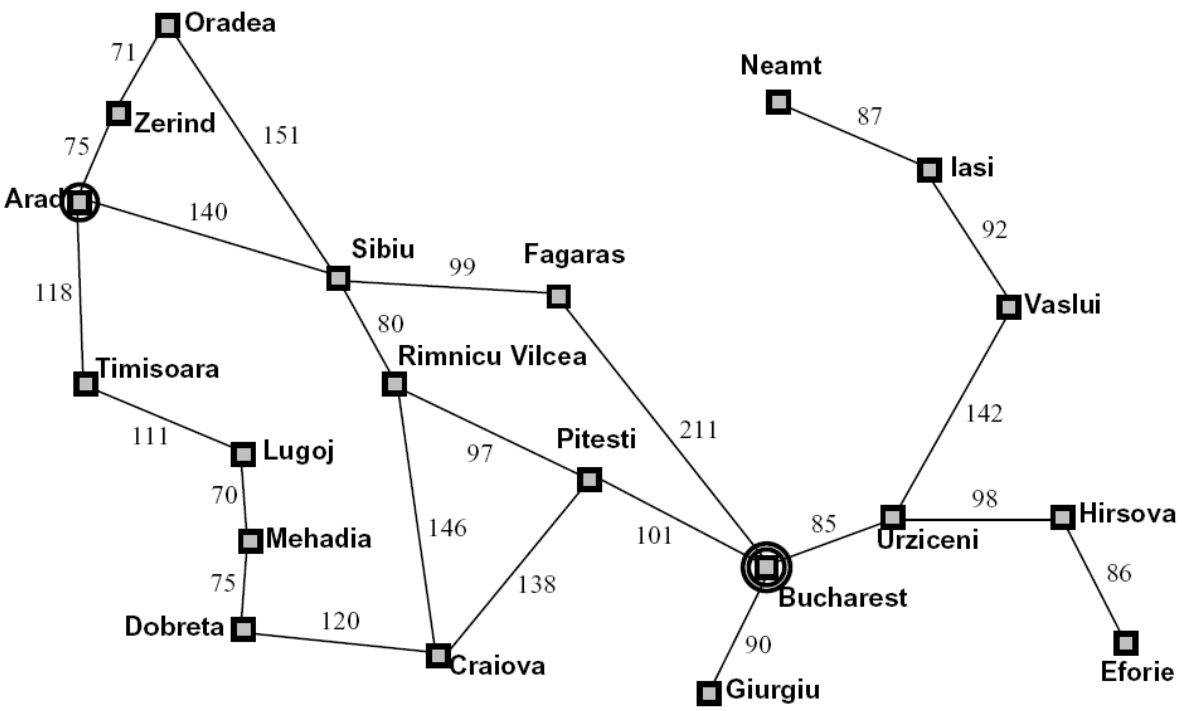


  - A successor function



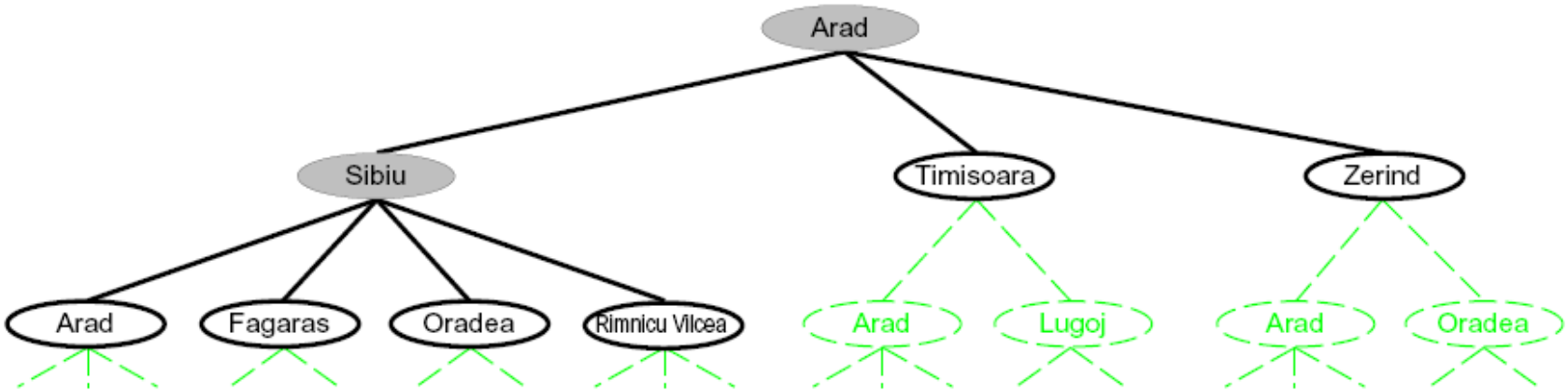
  - A start state and a goal test
- A solution is a sequence of actions (a plan) which transforms the start state to a goal state

Example: Romania



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

Another Search Tree



- Search:
  - Expand out possible plans
  - Maintain a fringe of unexpanded plans
  - 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
```

Detailed pseudocode is in the book!

- Important ideas:
  - Fringe
  - Expansion
  - Exploration strategy

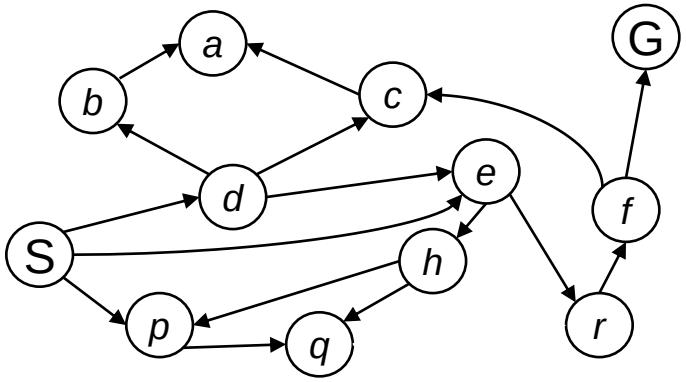
General Tree Search

```
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
```

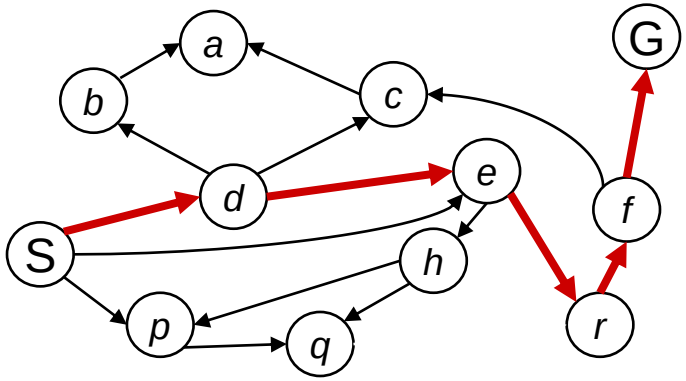
Detailed pseudocode is in the book!

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

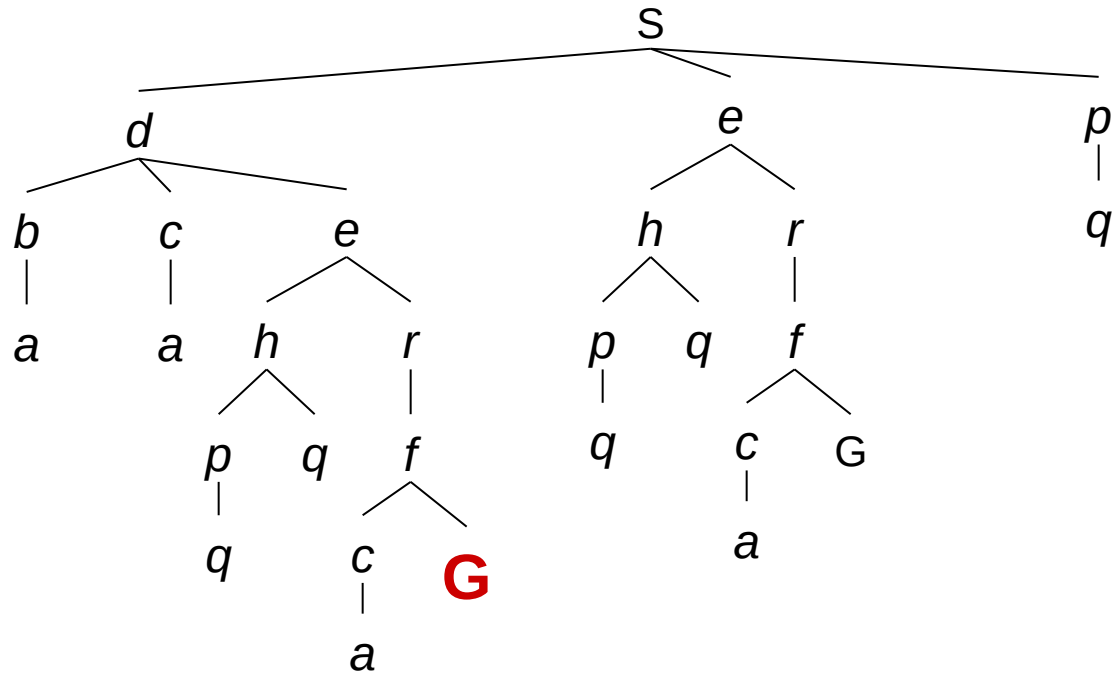
Example: Tree Search



State Graphs vs. Search Trees



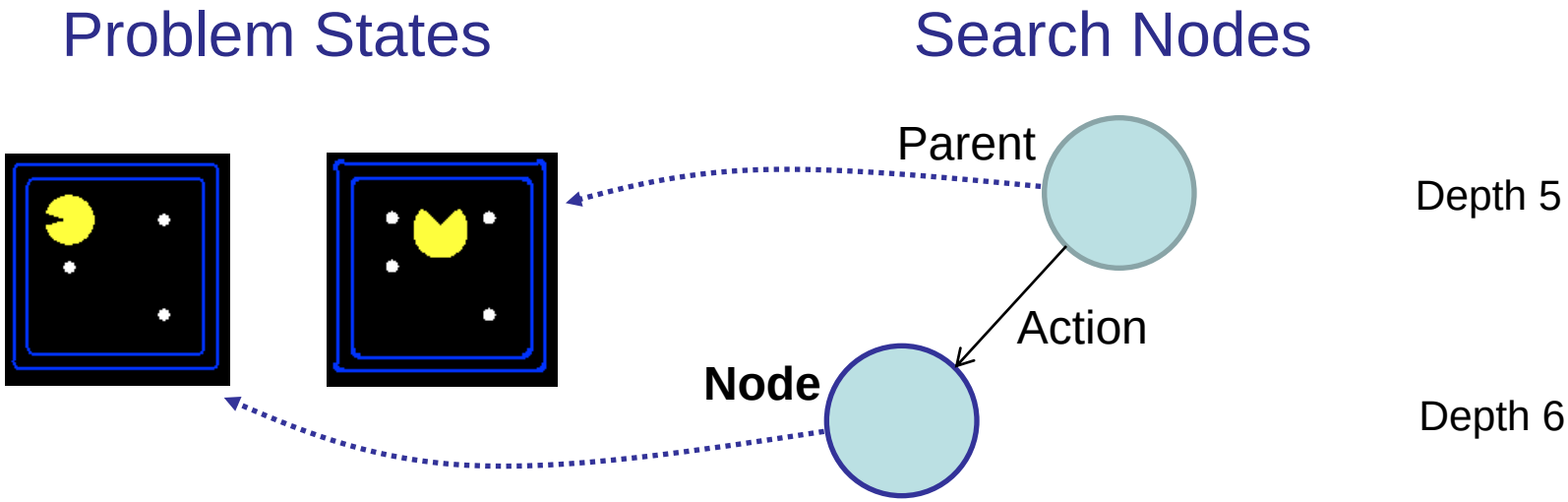
Each NODE in in the search tree is an entire PATH in the problem graph.



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

States vs. Nodes

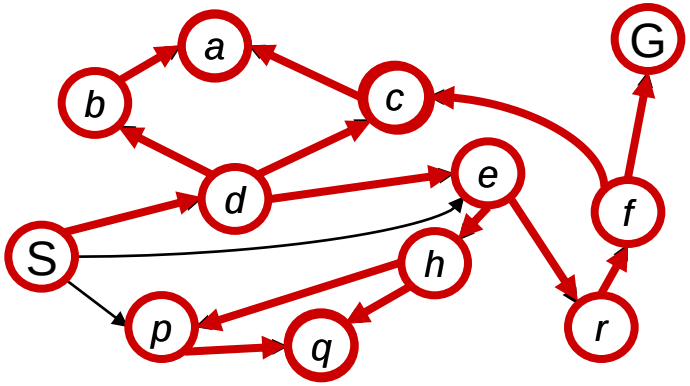
- Nodes in state space graphs are problem states
  - Represent an abstracted state of the world
  - Have successors, can be goal / non-goal, have multiple predecessors
- Nodes in search trees are plans
  - Represent a plan (sequence of actions) which results in the node's state
  - Have a problem state and one parent, a path length, a depth & a cost
  - The same problem state may be achieved by multiple search tree nodes

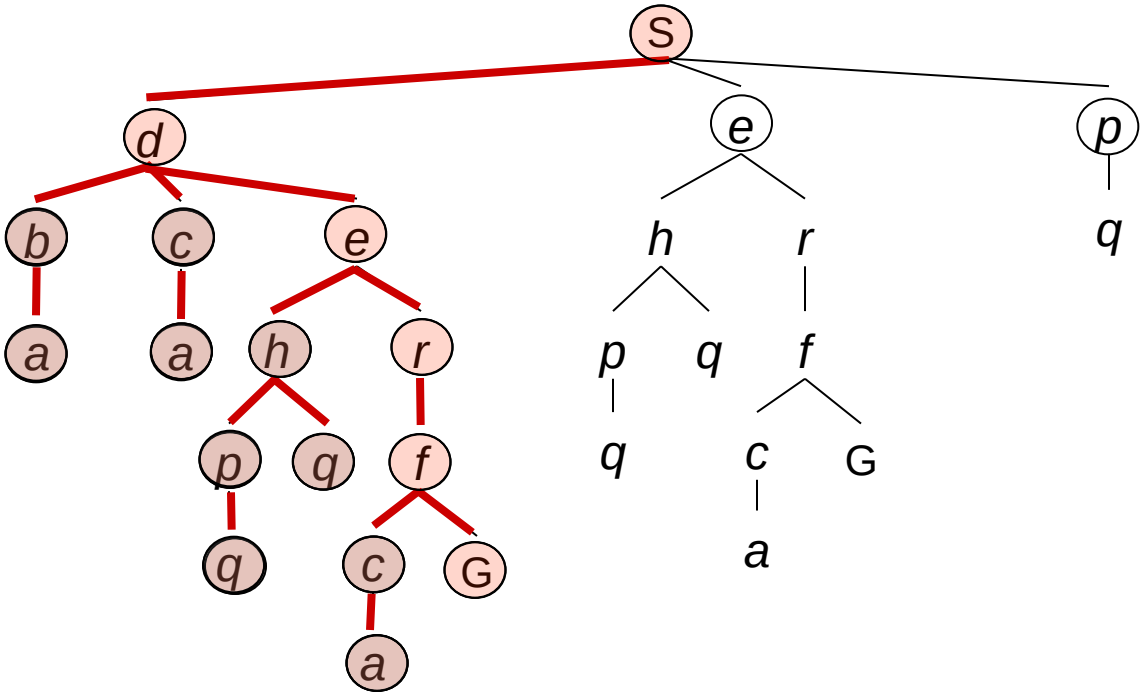


Review: Depth First Search

Strategy: expand deepest node first

Implementation: Fringe is a LIFO stack

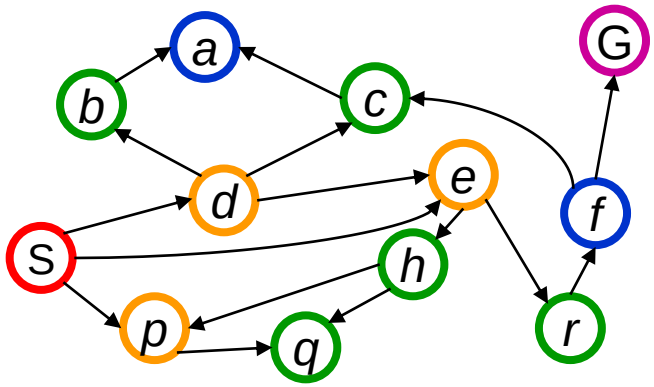


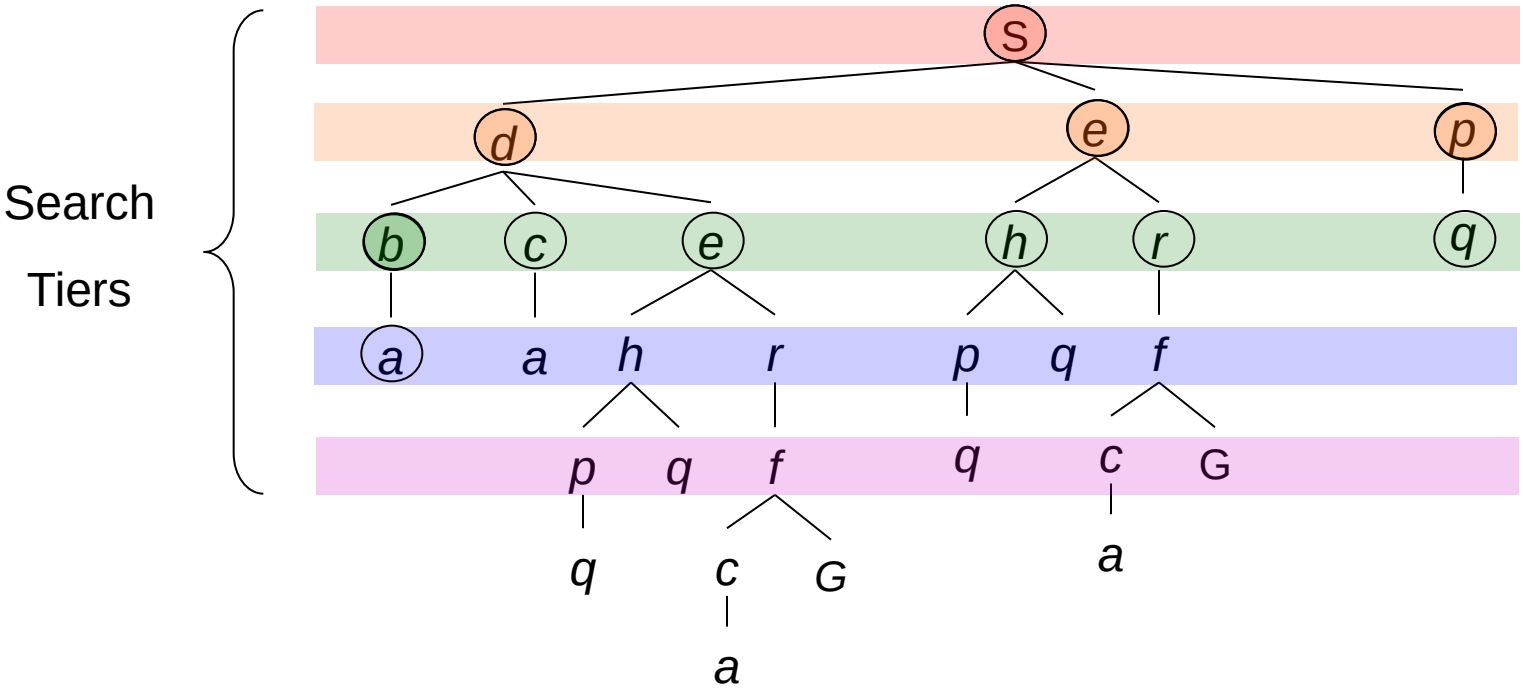


Review: Breadth First Search

Strategy: expand shallowest node first

Implementation: Fringe is a FIFO queue





Search Algorithm Properties

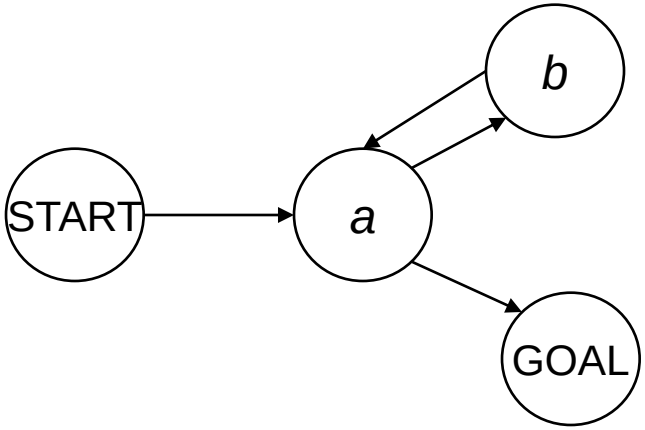
- Complete? Guaranteed to find a solution if one exists?
- Optimal? Guaranteed to find the least cost path?
- Time complexity?
- Space complexity?

Variables:

n	Number of states in the problem
b	The average branching factor *B* (the average number of successors)
C^*	Cost of least cost solution
s	Depth of the shallowest solution
m	Max depth of the search tree

DFS

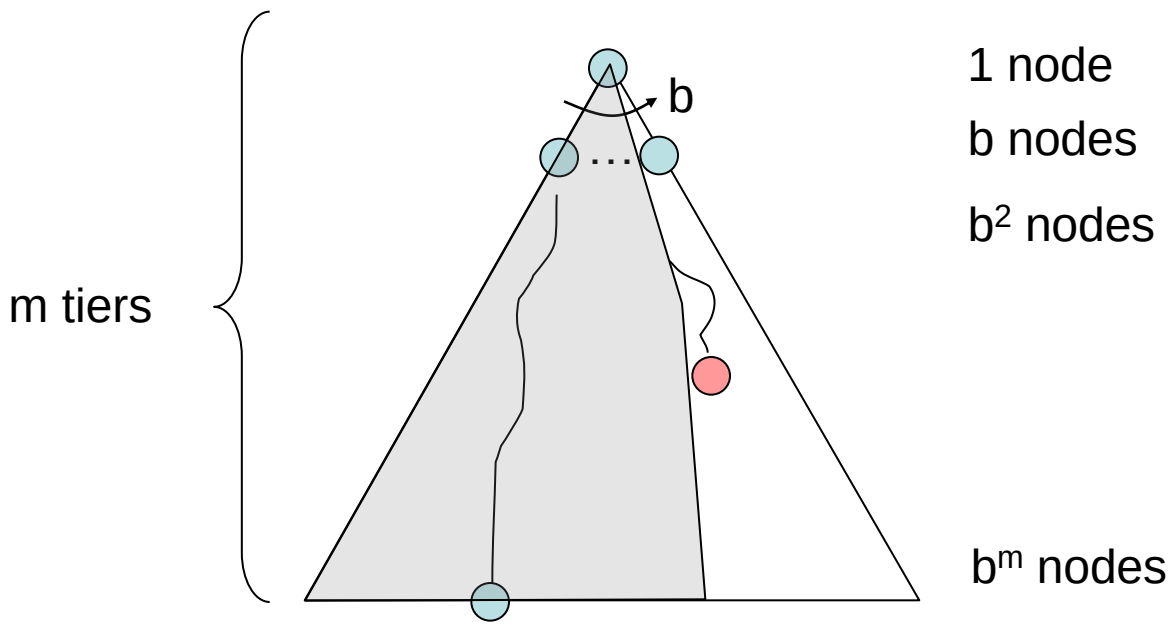
Algorithm		Complete	Optimal	Time	Space
DFS	Depth First Search	N	N	Infinite	Infinite



- Infinite paths make DFS incomplete…
- How can we fix this?

DFS

- With cycle checking, DFS is complete.*

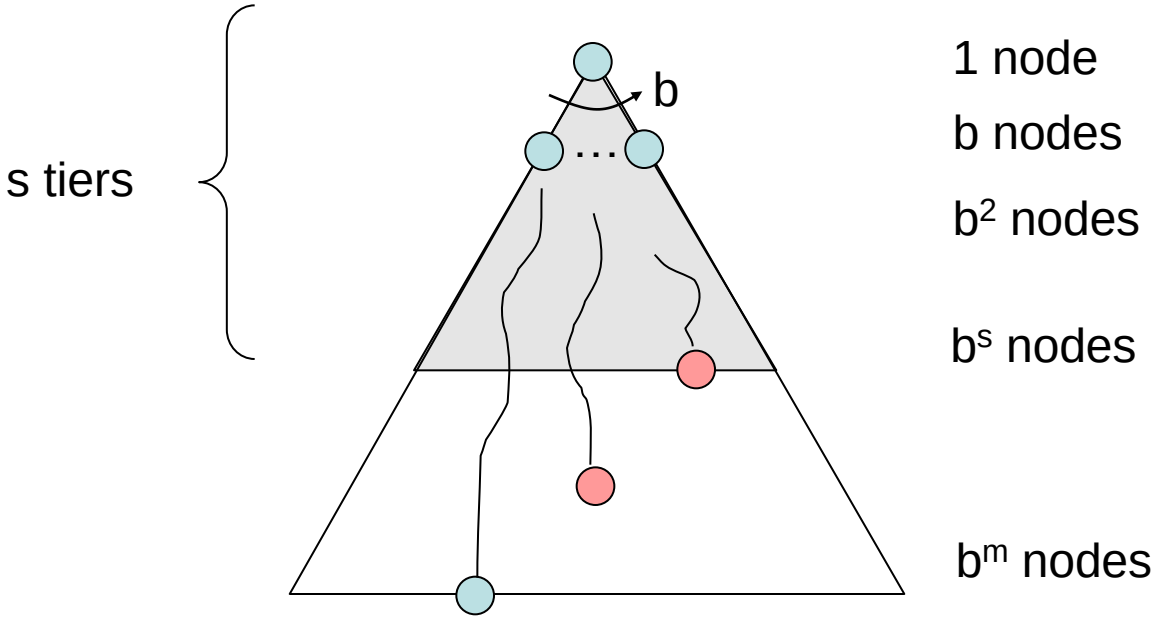


Algorithm		Complete	Optimal	Time	Space
DFS	w/ Path Checking	Y	N	$O(b^{m+1})$	$O(bm)$

* Or graph search – next lecture.

BFS

Algorithm		Complete	Optimal	Time	Space
DFS	w/ Path Checking	Y	N	$O(b^{m+1})$	$O(bm)$
BFS		Y	N*	$O(b^{s+1})$	$O(b^{s})$



- When is BFS optimal?

Comparisons

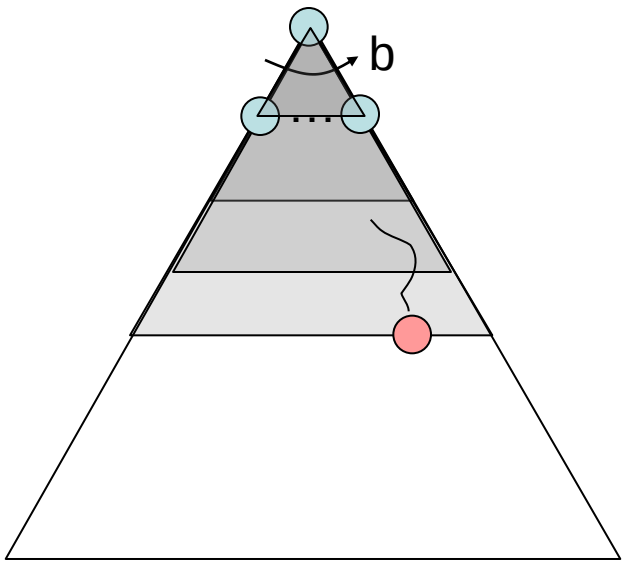
- When will BFS outperform DFS?
- When will DFS outperform BFS?

Iterative Deepening

Iterative deepening uses DFS as a subroutine:

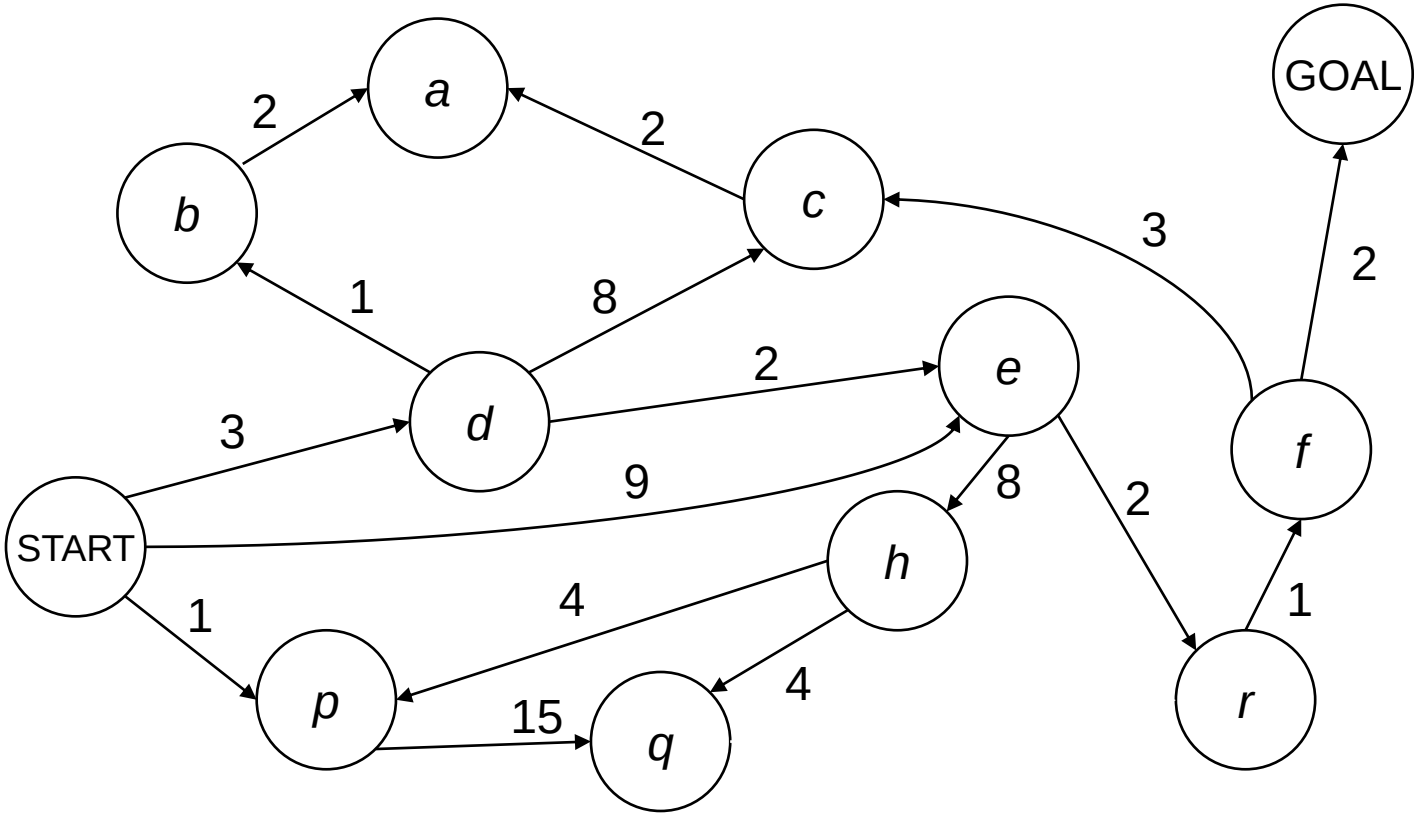
1. Do a DFS which only searches for paths of length 1 or less.
2. If “1” failed, do a DFS which only searches paths of length 2 or less.
3. If “2” failed, do a DFS which only searches paths of length 3 or less.

….and so on.



Algorithm		Complete	Optimal	Time	Space
DFS	w/ Path Checking	Y	N	$O(b^{m+1})$	$O(bm)$
BFS		Y	N*	$O(b^{s+1})$	$O(b^{s})$
ID		Y	N*	$O(b^{s+1})$	$O(bs)$

Costs on Actions



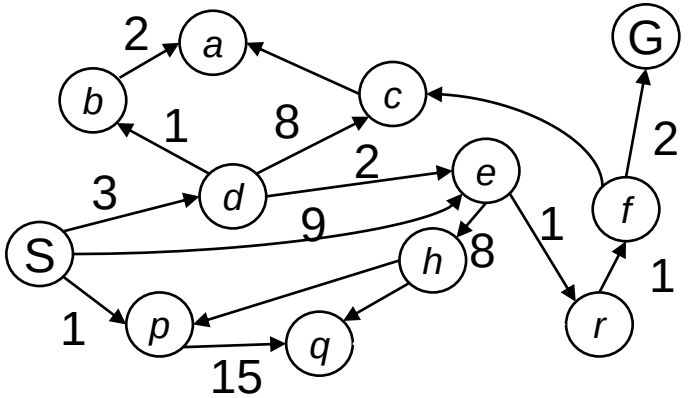
Notice that BFS finds the shortest path in terms of number of transitions. It does not find the least-cost path.

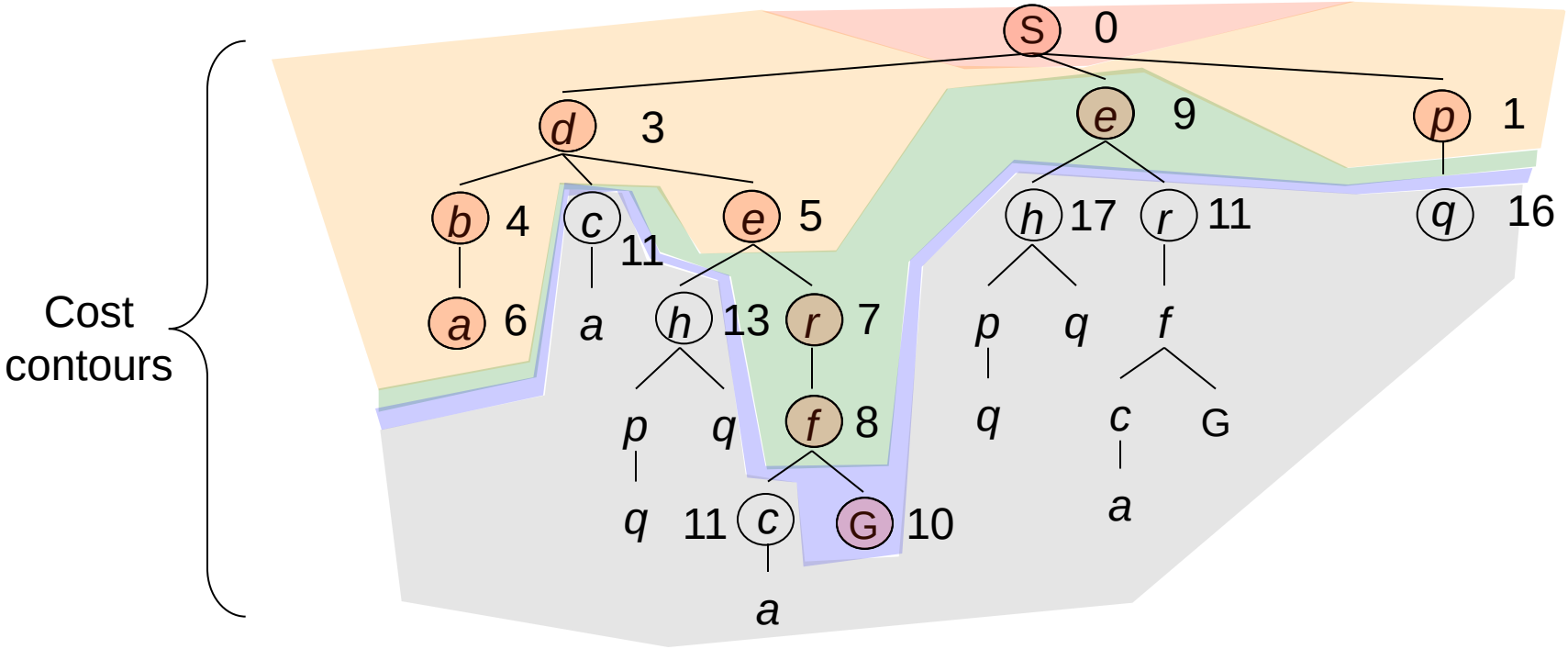
We will quickly cover an algorithm which does find the least-cost path.

Uniform Cost Search

Expand cheapest node first:

Fringe is a priority queue





Priority Queue Refresher

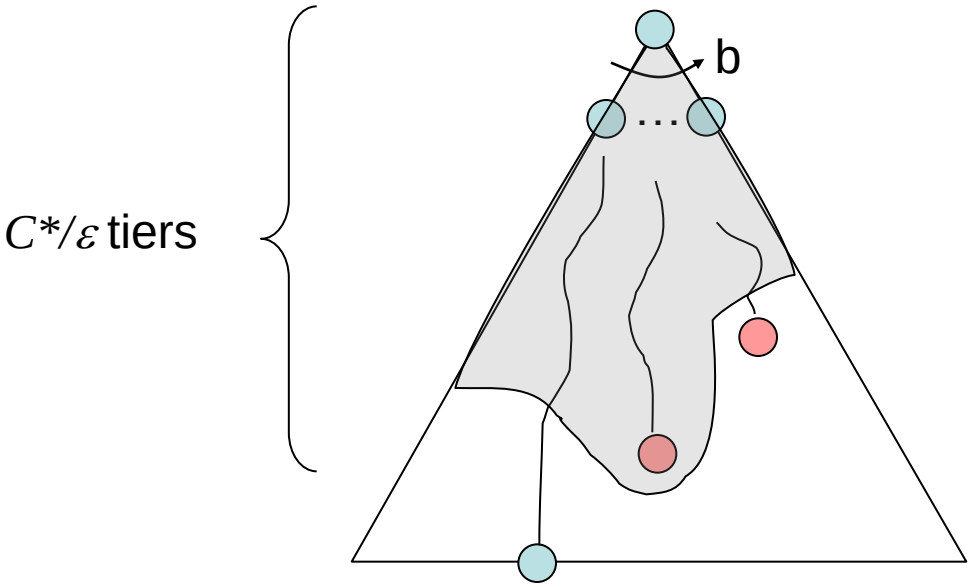
- A priority queue is a data structure in which you can insert and retrieve (key, value) pairs with the following operations:

pq.push(key, value)	inserts *(key, value)* into the queue.
pq.pop()	returns the key with the lowest value, and removes it from the queue.

- You can decrease a key's priority by pushing it again
- Unlike a regular queue, insertions aren't constant time, usually O(log *n*)
- We'll need priority queues for cost-sensitive search methods

Uniform Cost Search

Algorithm		Complete	Optimal	Time	Space
DFS	w/ Path Checking	Y	N	$O(b^{m+1})$	$O(bm)$
BFS		Y	N	$O(b^{s+1})$	$O(b^{s})$
UCS		Y*	Y	$O(b^{C^*/\varepsilon})$	$O(b^{C^*/\varepsilon})$



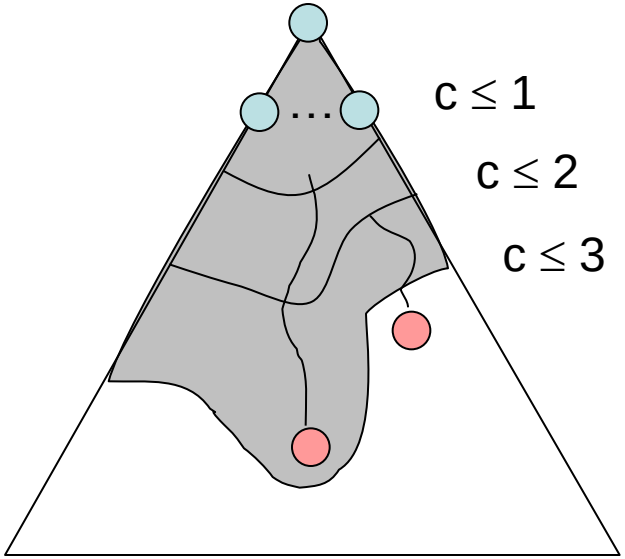
** UCS can fail if actions can get arbitrarily cheap*

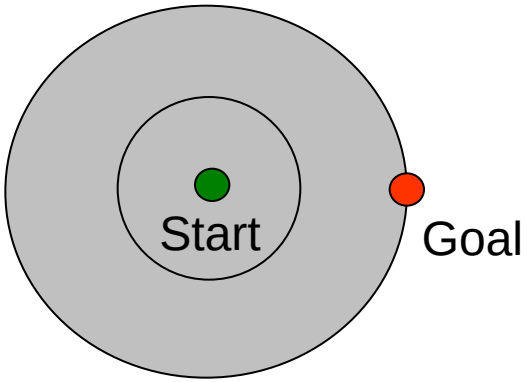
Done with Search?

UCS!

Uniform Cost Issues

- Remember: explores increasing cost contours
- The good: UCS is complete and optimal!
- The bad:
  - Explores options in every "direction"
  - No information about goal location





[demo: search demo empty]