

CS 188: Artificial Intelligence

Fall 2007

Lecture 2: Queue-Based Search

8/31/2007

Dan Klein – UC Berkeley

Many slides from either Stuart Russell or Andrew Moore

Announcements

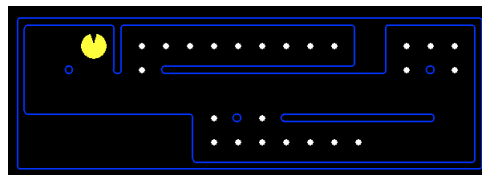
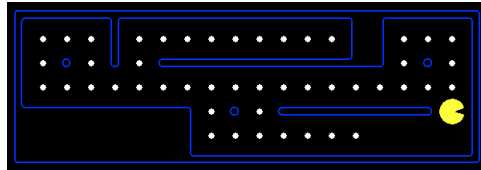
- Next week
 - New room is 105 North Gate, starts Tuesday
 - Check web page for sections (new coming)
- Lab Friday 10am to 5pm in Soda 275
 - Learn Python
 - Come for whatever times you like
- Project 1.1 posted by weekend due 9/12

Today

- Agents that Plan Ahead
- Search Problems
- Uniformed Search Methods
 - Depth-First Search
 - Breadth-First Search
 - Uniform-Cost Search
- Heuristic Search Methods
 - Greedy Search
 - A* Search

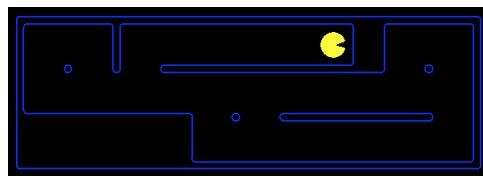
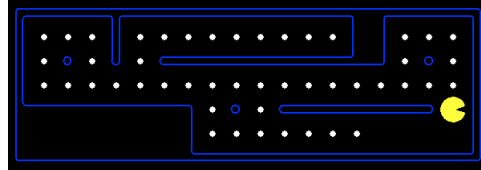
Reflex Agents

- Reflex agents:
 - Choose action based on current percept and memory
 - May have memory or a model of the world's current state
 - Do not consider the future consequences of their actions
- Can a reflex agent be rational?


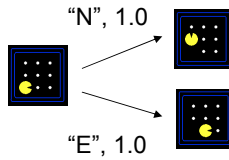


Goal Based Agents

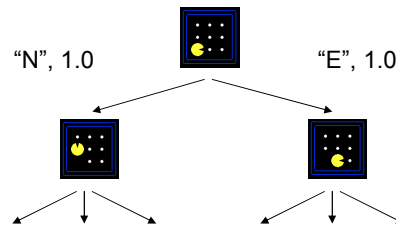
- Goal-based agents:
 - Plan ahead
 - Decisions based on (hypothesized) consequences of actions
 - Must have a model of how the world evolves in response to actions



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 which transform the start state to a goal state

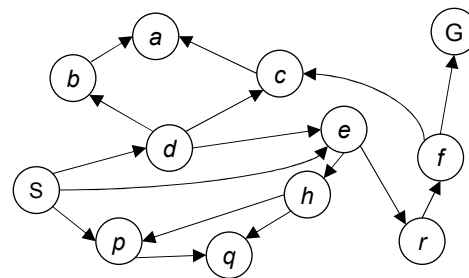
Search Trees



- A search tree:
 - This is a “what if” tree
 - Start state at the root node
 - Children correspond to successors
 - Nodes labeled with states, correspond to PATHS to those states
 - For most problems, can never actually build the whole tree
 - So, have to find ways of using only the important parts of the tree!

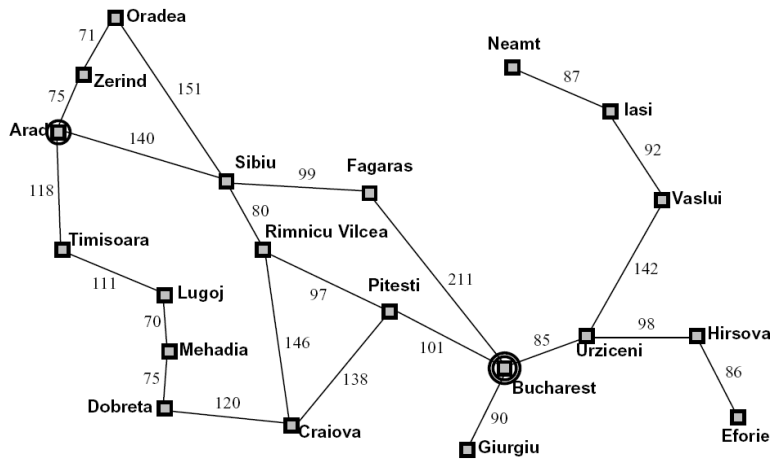
State Space Graphs

- There's some big graph in which
 - Each state is a node
 - Each successor is an outgoing arc
- Important: For most problems we could never actually build this graph
- How many states in Pacman?

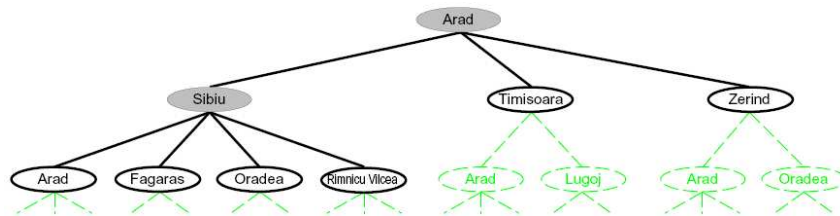


Laughably tiny search graph for a tiny search problem

Example: Romania



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

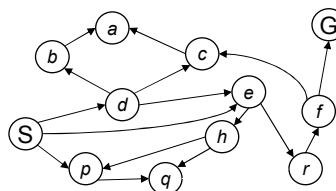
```
function TREE-SEARCH(problem, strategy) returns a solution, or failure
  initialize the search tree using the initial state of problem
  loop do
    if there are no candidates for expansion then return failure
    choose a leaf node for expansion according to strategy
    if the node contains a goal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

- Important ideas:
 - Fringe
 - Expansion
 - Exploration strategy

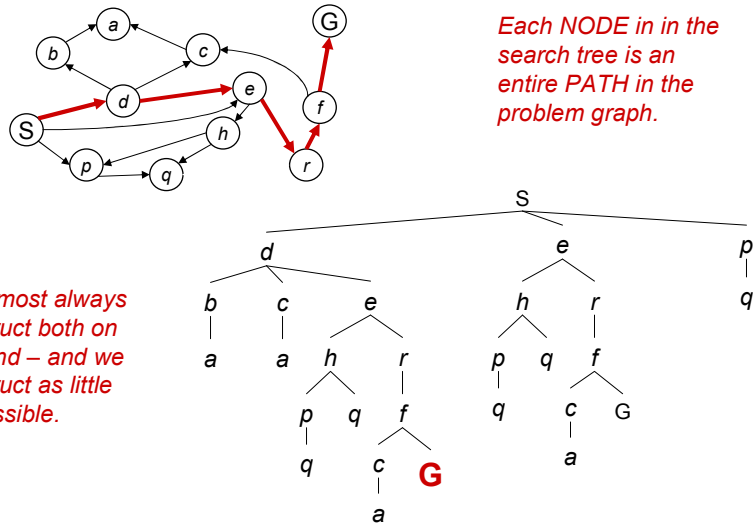
*Detailed pseudocode
is in the book!*

- Main question: which fringe nodes to explore?

Example: Tree Search



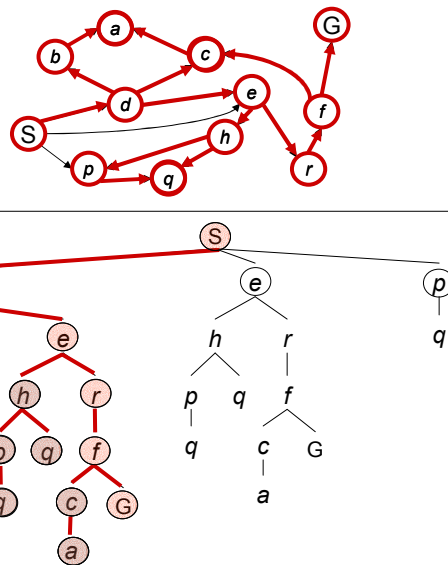
State Graphs vs Search Trees



Review: Depth First Search

Strategy: expand deepest node first

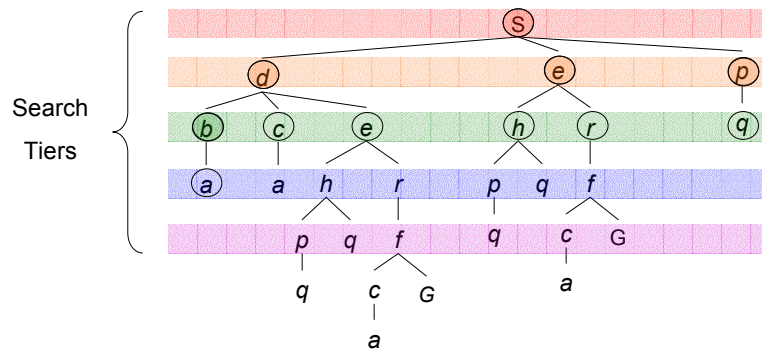
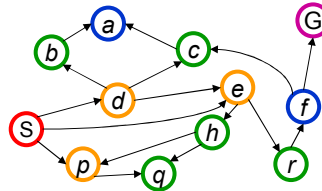
Implementation: Fringe is a LIFO stack



Review: Breadth First Search

Strategy: expand shallowest node first

Implementation: Fringe is a FIFO queue



Search Algorithm Properties

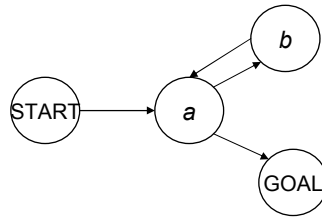
- **Complete?** Guaranteed to find a solution if one exists?
- **Optimal?** Guaranteed to find the least cost path?
- **Time complexity?**
- **Space complexity?**

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

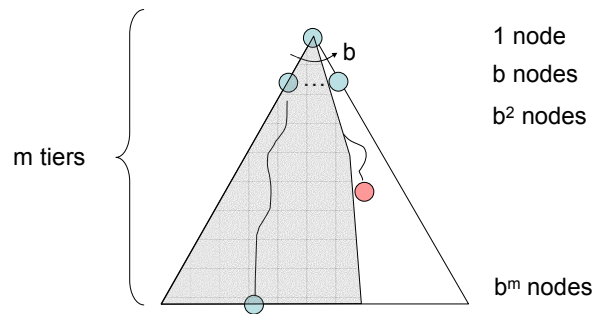
Algorithm		Complete	Optimal	Time	Space
DFS	Depth First Search	N	N	Infinite	Infinite



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

DFS

- With cycle checking, DFS is complete.

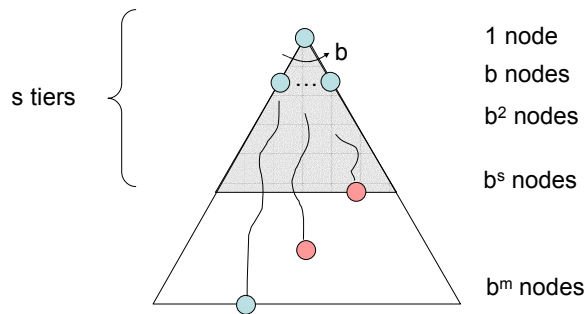


Algorithm		Complete	Optimal	Time	Space
DFS	w/ Path Checking	Y	N	$O(b^{m+1})$	$O(bm)$

- When is DFS optimal?

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?

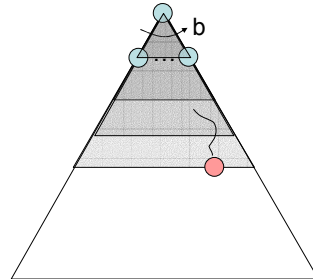
Comparisons

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

Iterative Deepening

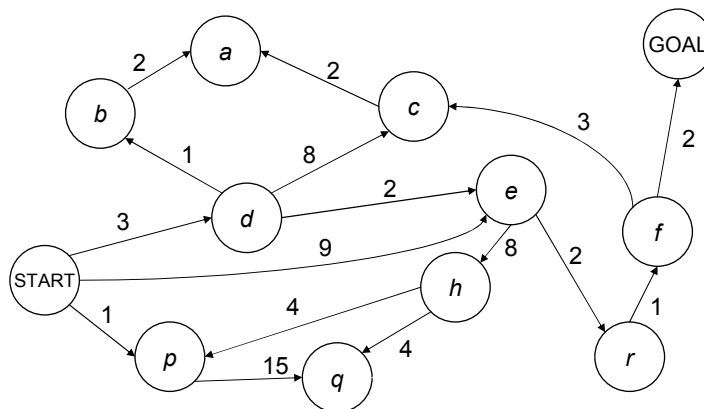
Iterative deepening uses DFS as a subroutine:

1. Do a DFS which only searches for paths of length 1 or less. (DFS gives up on any path of length 2)
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

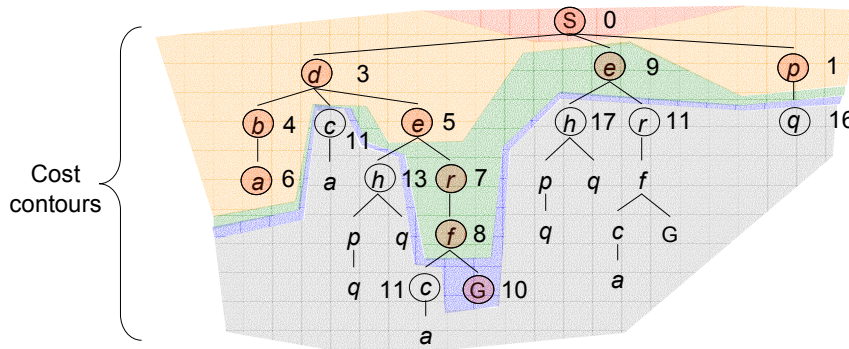
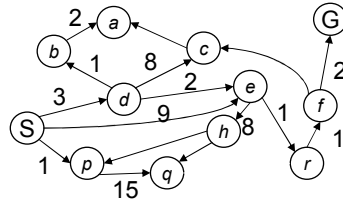


Notice that BFS finds the shortest path in terms of number of transitions. It does not find the least-cost path.

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



Cost contours



Priority Queue Refresher

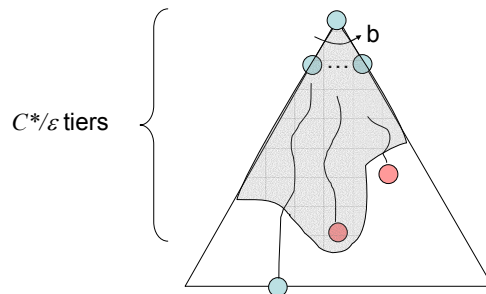
- A priority queue is a data structure in which you can insert and retrieve (key, value) pairs with the following operations:

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 promote or demote keys by resetting their priorities
- Unlike a regular queue, insertions into a priority queue are not constant time, usually $O(\log n)$
- We'll need priority queues for most cost-sensitive search methods.

Uniform Cost Search

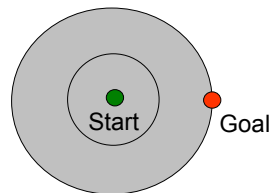
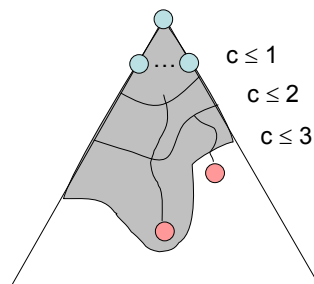
Algorithm		Complete	Optimal	Time	Space
DFS	w/ Path Checking	Y	N	$O(b^{m+1})$	$O(bm)$
BFS		Y	N	$O(b^{s+1})$	$O(b^s)$
UCS		Y*	Y	$O(C^* b^{C^*/\epsilon})$	$O(b^{C^*/\epsilon})$



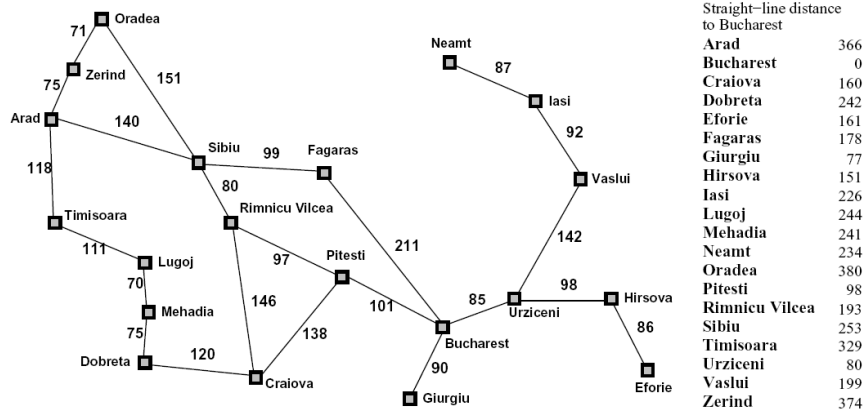
We'll talk more about uniform cost search's failure cases later...

Uniform Cost Problems

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

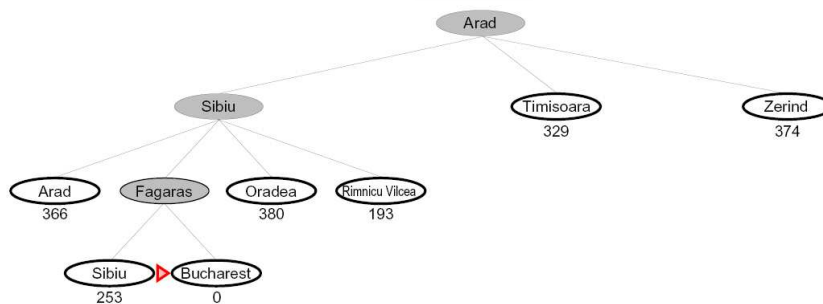


Best First / Greedy Search



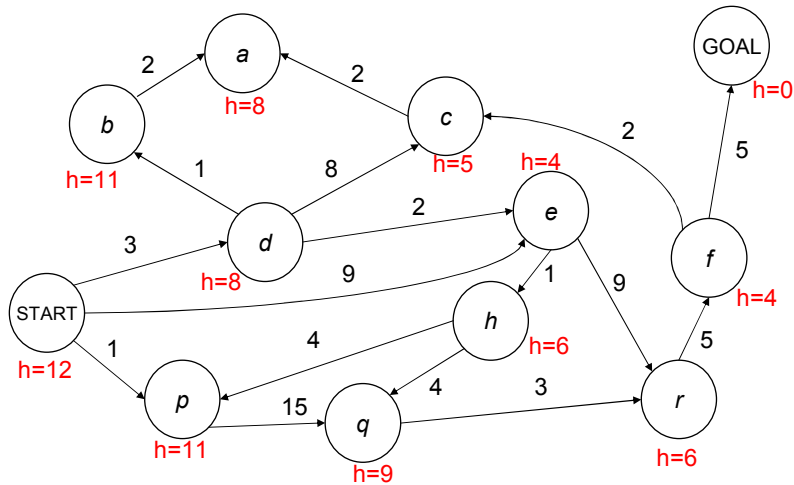
Best First / Greedy Search

- Expand the node that seems closest...



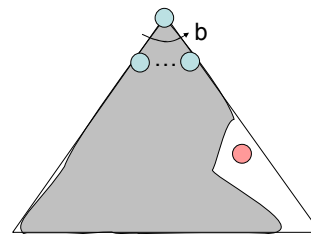
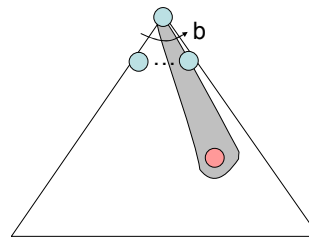
- What can go wrong?

Best First / Greedy Search

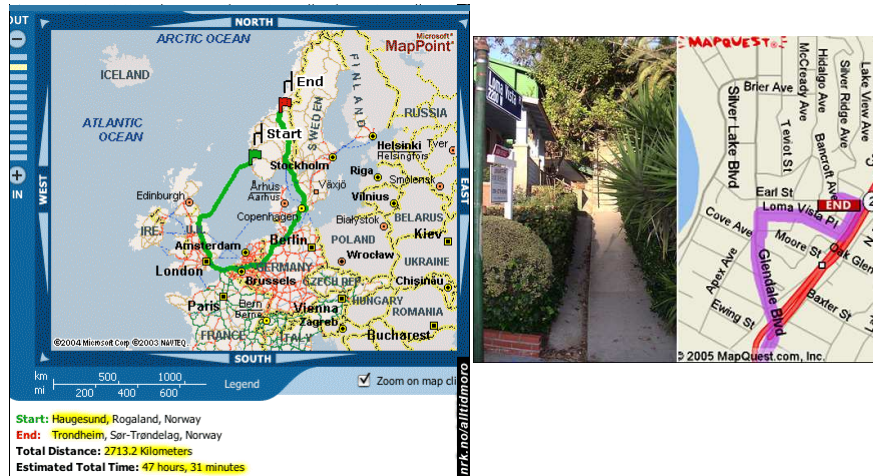


Best First / Greedy Search

- A common case:
 - Best-first takes you straight to the (wrong) goal
- Worst-case: like a badly-guided DFS in the worst case
 - Can explore everything
 - Can get stuck in loops if no cycle checking
- Like DFS in completeness (finite states w/ cycle checking)

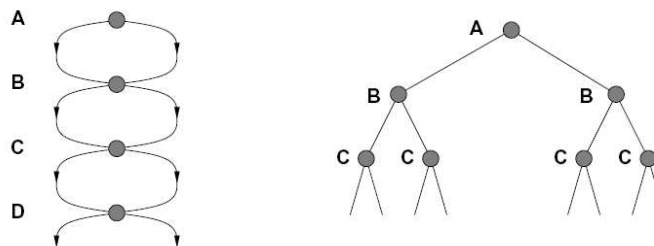


Search Gone Wrong?



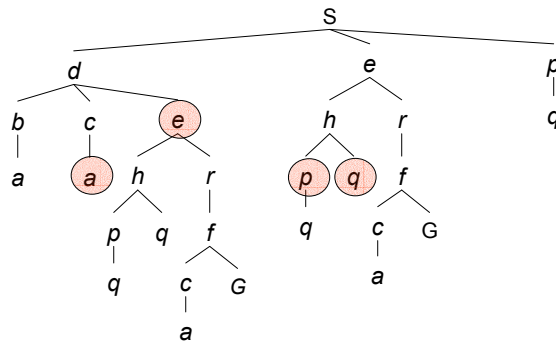
Extra Work?

- Failure to detect repeated states can cause exponentially more work. Why?



Graph Search

- In BFS, for example, we shouldn't bother expanding the circled nodes (why?)



Graph Search

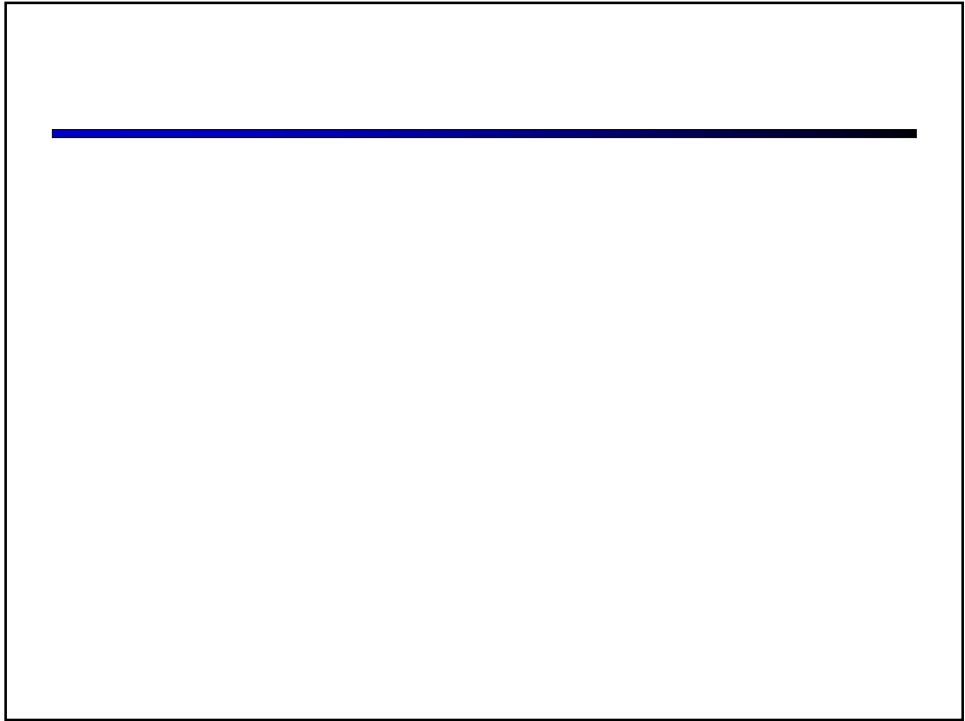
- Very simple fix: never expand a node twice

```

function GRAPH-SEARCH(problem, fringe) returns a solution, or failure
  closed ← an empty set
  fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
  loop do
    if fringe is empty then return failure
    node ← REMOVE-FRONT(fringe)
    if GOAL-TEST(problem, STATE[node]) then return node
    if STATE[node] is not in closed then
      add STATE[node] to closed
      fringe ← INSERTALL(EXPAND(node, problem), fringe)
  end
  
```

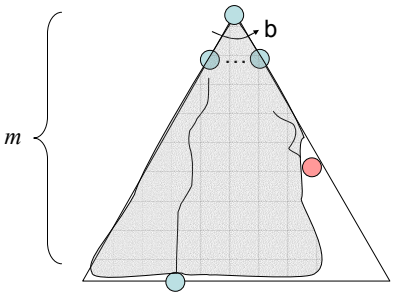


- Can this wreck completeness? Why or why not?



Best First Greedy Search

Algorithm	Complete	Optimal	Time	Space
Greedy Best-First Search	Y*	N	$O(b^m)$	$O(b^m)$



- What do we need to do to make it complete?
- Can we make it optimal? Next class!

