Decoding

Wednesday, March 25, 2015

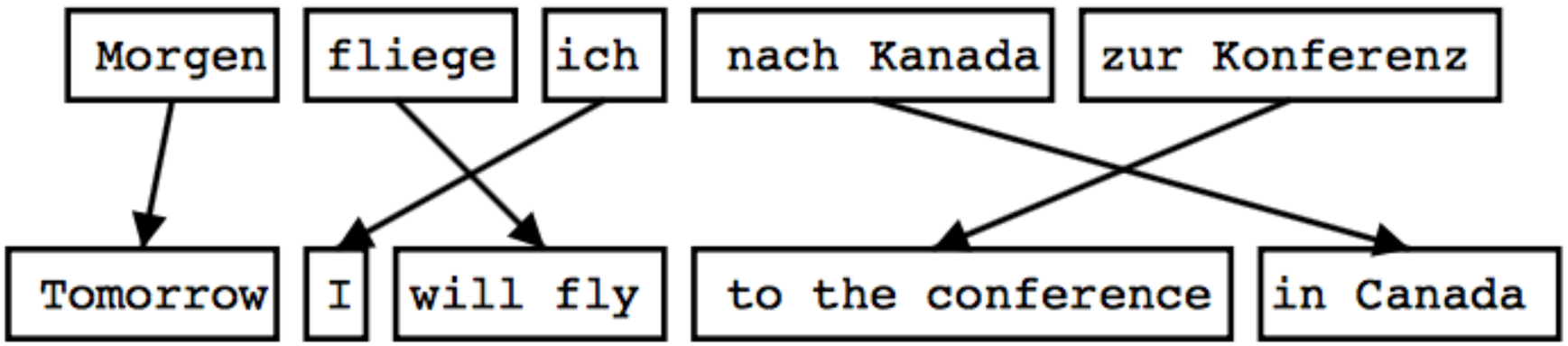
Plan for Today:

- Stack Decoding
- Prepare you for Project 2

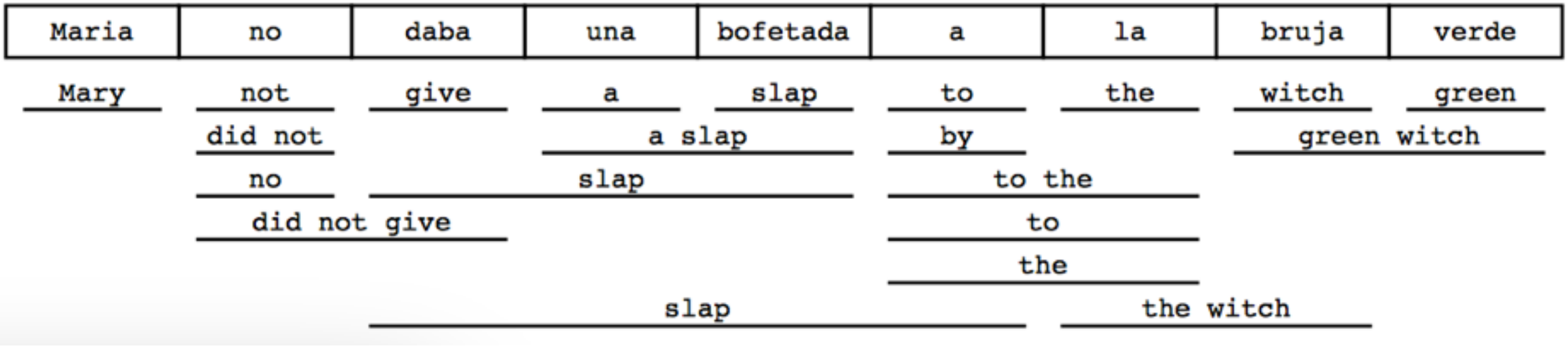
Phrase Tables

$\bar{\mathbf{f}}$	$\bar{\mathbf{e}}$	$p(\bar{\mathbf{f}} \mid \bar{\mathbf{e}})$
das Thema	the issue	0.41
	the point	0.72
	the subject	0.47
	the thema	0.99
es gibt	there is	0.96
	there are	0.72
morgen	tomorrow	0.9
fliege ich	will I fly	0.63
	will fly	0.17
	I will fly	0.13

Phrase-Based Decoding



Translation Options



Core algorithm for decoding

Implemented in Pharaoh

Described in a paper by Philipp Koehn

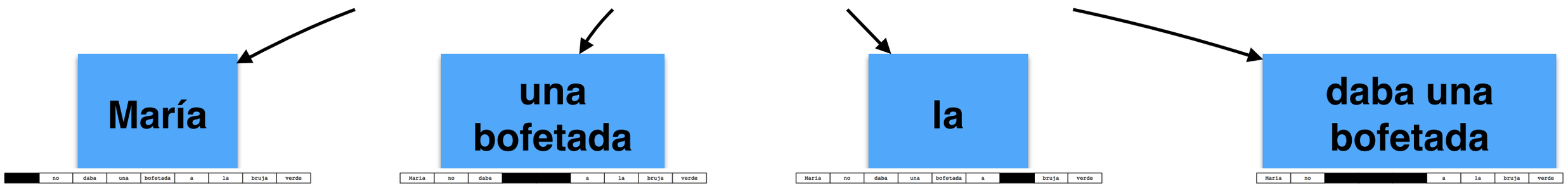
Iteratively:

- Pick a foreign phrase to translate
- Add it to the English output
- English translation is built left-to-right

Stepping through:

Maria	no	daba	una	bofetada	a	la	bruja	verde
Mary	not	give	a	slap	to	the	witch	green
	did not		a slap		by		green witch	
	no	slap			to the			
	did not give				to			
					the			
		slap				the witch		

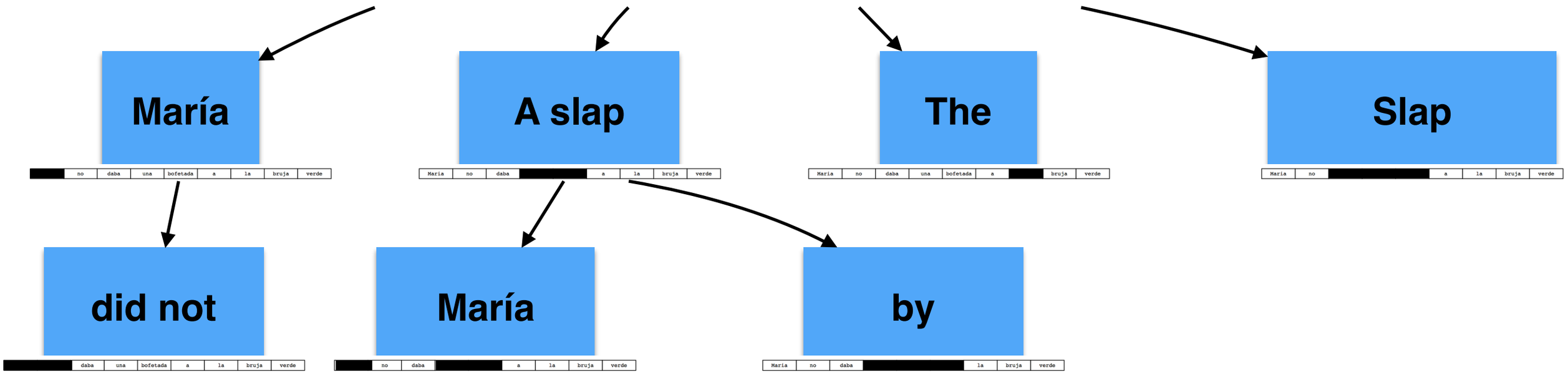
Pick a phrase to be the first English word.



Stepping through:

Maria	no	daba	una	bofetada	a	la	bruja	verde
Mary	not	give	a	slap	to	the	witch	green
	did not		a slap		by		green witch	
	no	slap			to the			
	did not give				to			
					the			
		slap				the witch		

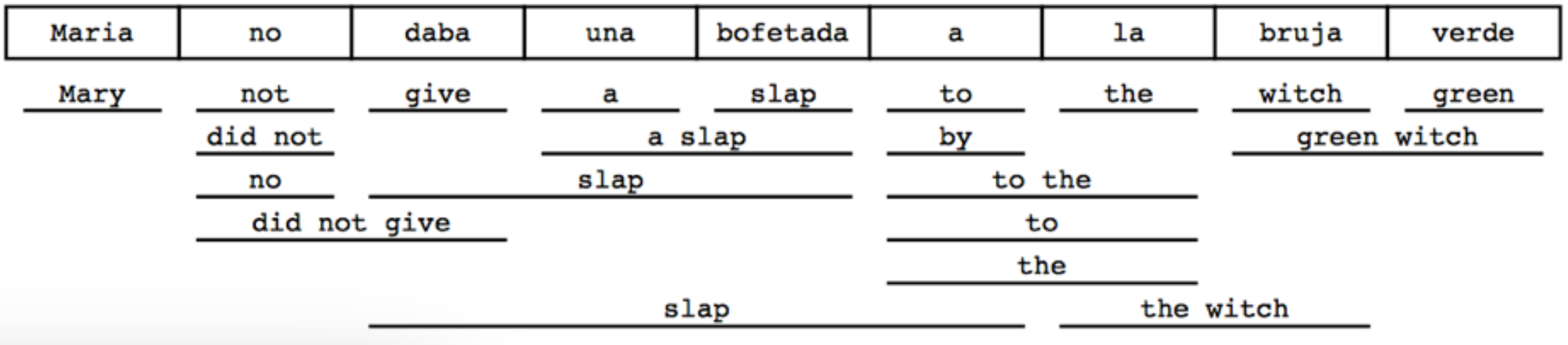
Pick a phrase to be the next English word.



Size of Search Space

For an N-word foreign sentence…

- How many possible phrases?
- How many possible re-orderings?
- (And however many translation candidates our phrase table comes up with…)



Big (exponential) search space!

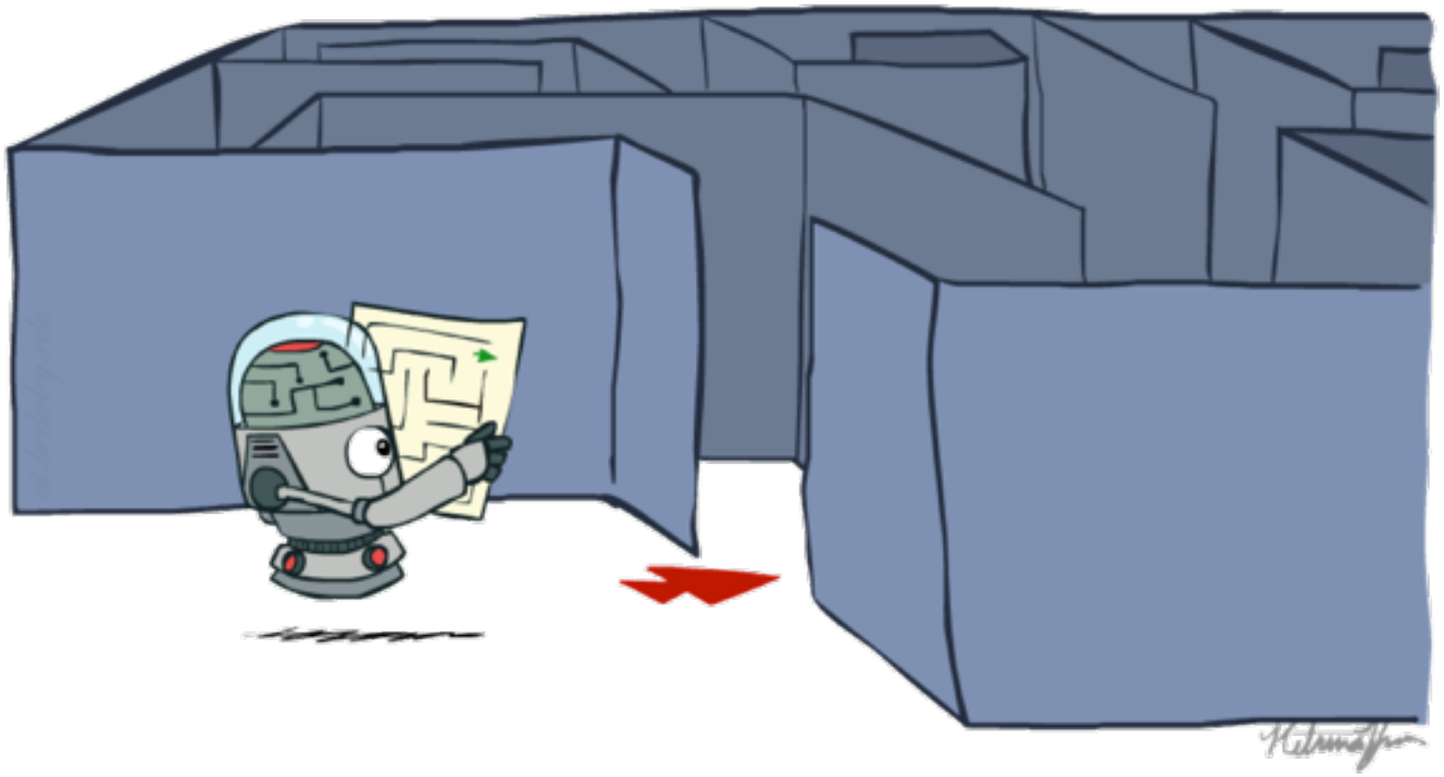
1	0
2	1
3	5
4	25
5	141
6	911
7	6703
8	55 581
9	513 929
10	5 248 891

Conclusion: Exhaustive search is not going to scale well enough for us.

Aside: Informed Search

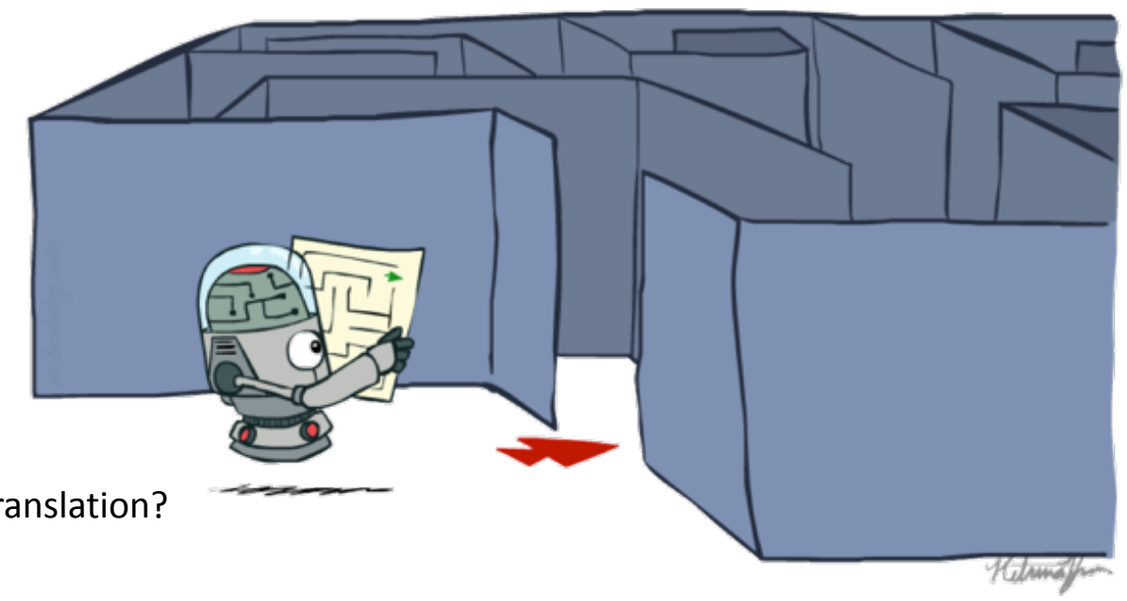
[The following slides were created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley. All CS188 materials are available at http://ai.berkeley.edu.]

Overview: Search



Overview: Search

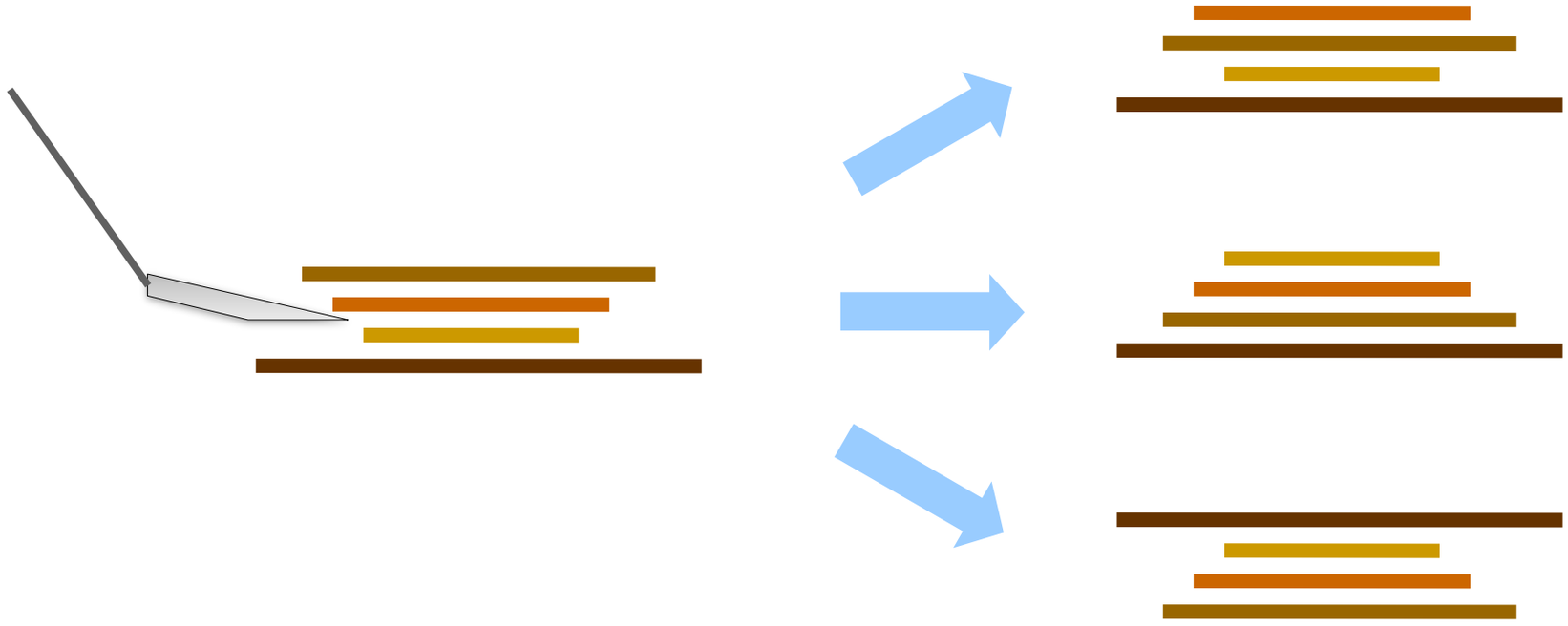
- Search problem:
  - States (configurations of the world)
  - Actions and costs
  - Successor function (world dynamics)
  - Start state and goal test



- Search tree:
  - Nodes: represent plans for reaching states
    - What should your node implementations have for phrase translation?
  - Plans have costs (sum of action costs)

- Search algorithm:
  - Systematically builds a search tree
  - Chooses an ordering of the fringe (unexplored nodes)
  - Optimal: finds least-cost plans

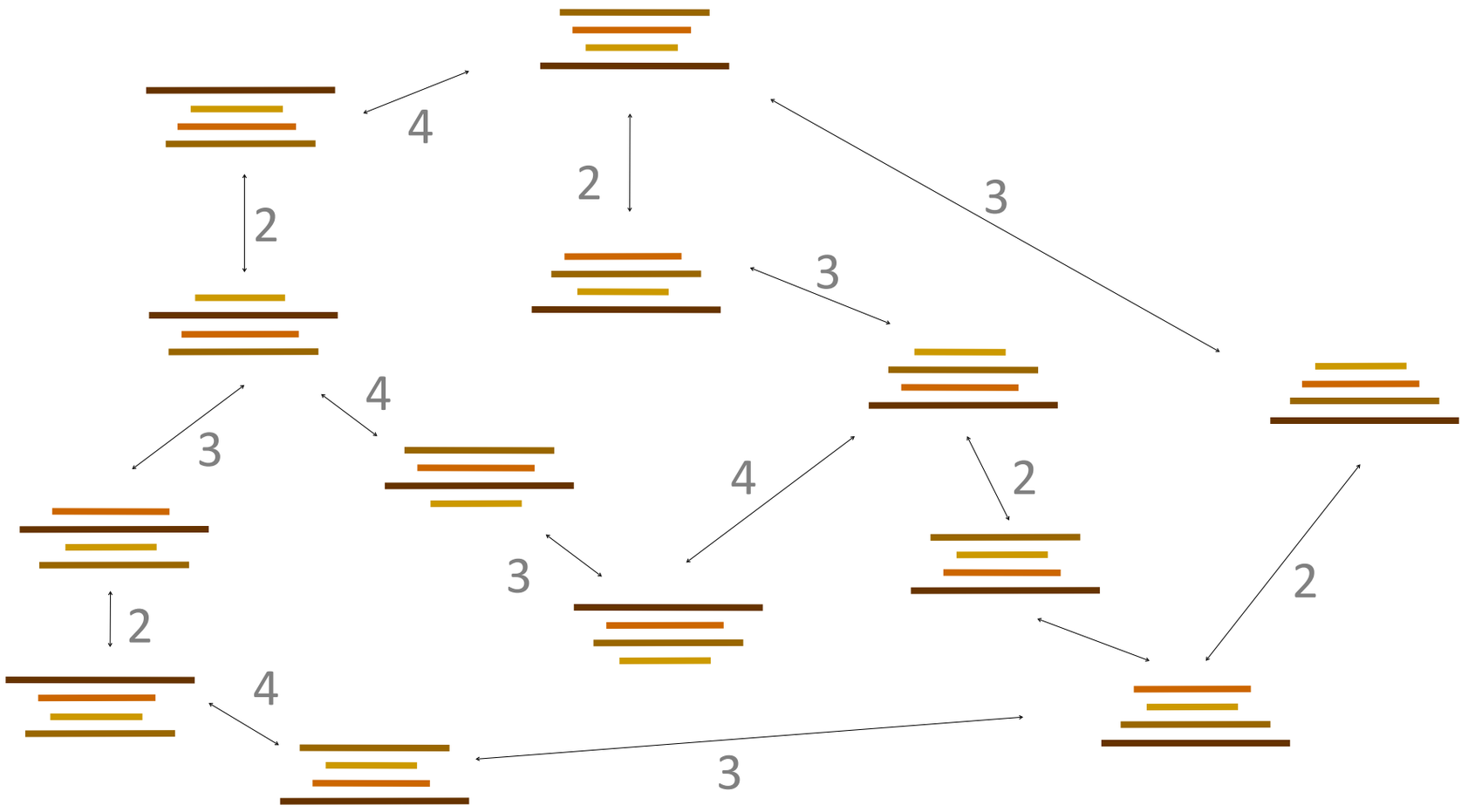
Example: Pancake Problem



Cost: Number of pancakes flipped

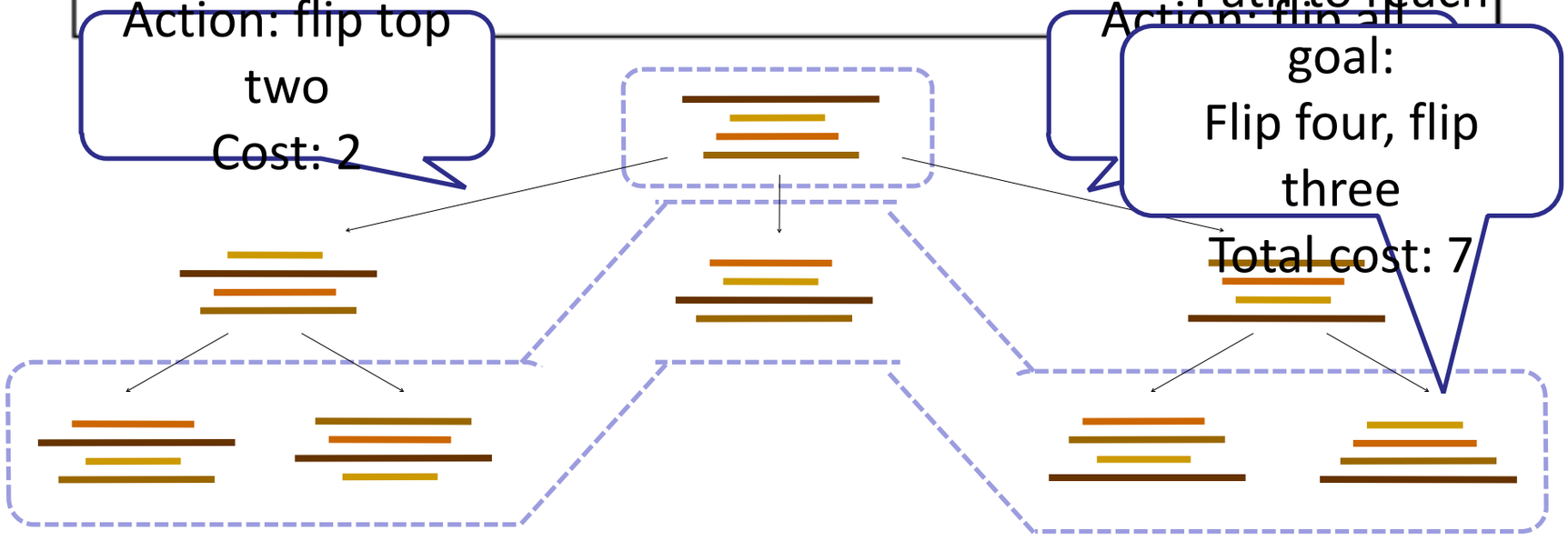
Example: Pancake Problem

State space graph with costs as weights



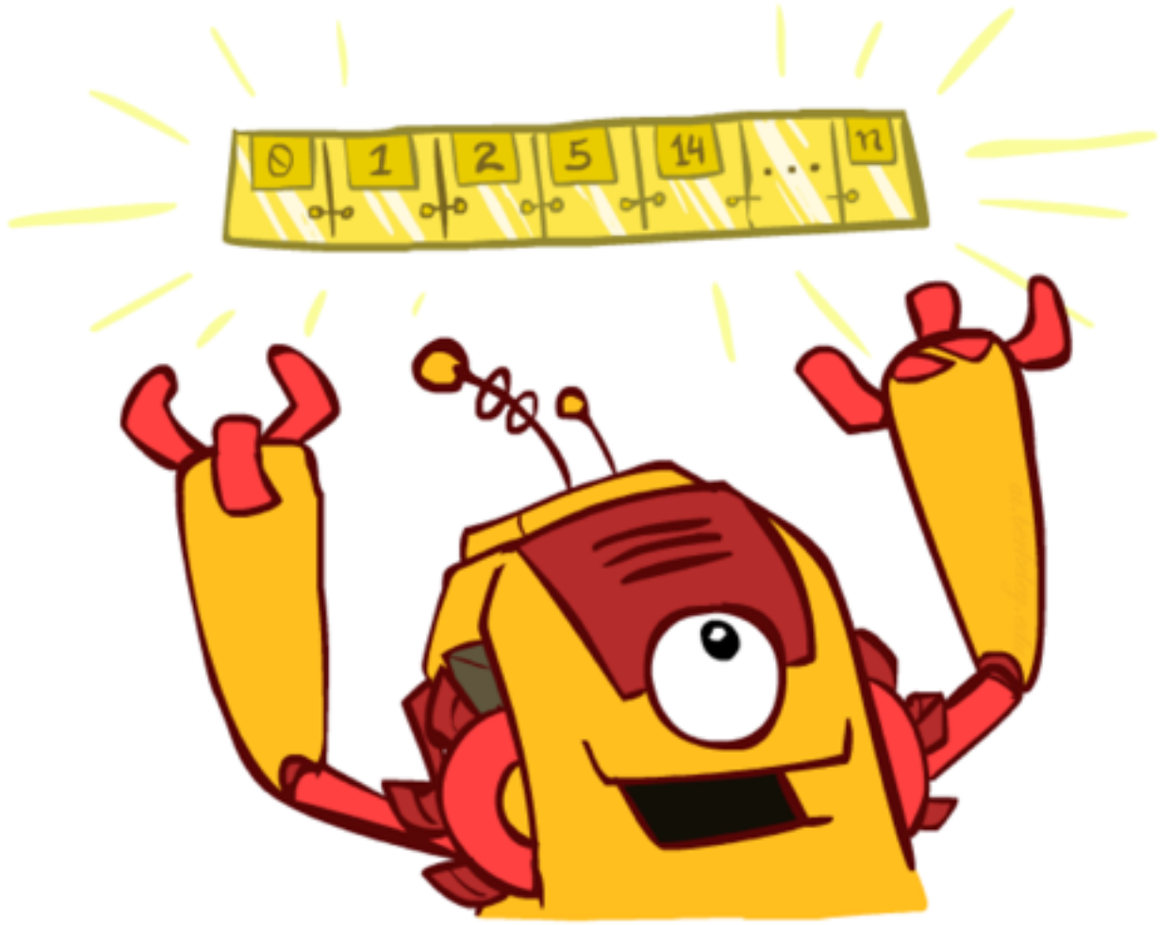
General Tree Search

```
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
```



The One Queue

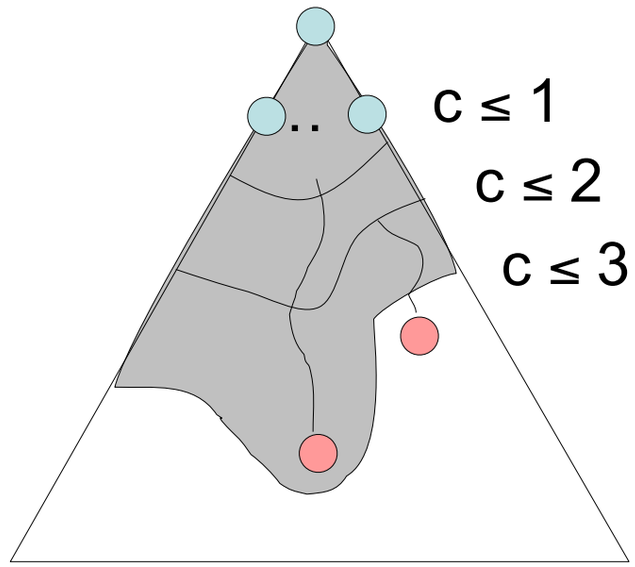
- Many search algorithms are the same except for fringe strategies
  - Conceptually, all fringes are priority queues (i.e. collections of nodes with attached priorities)
- Practically, for DFS and BFS, you can avoid the log(n) overhead from an actual priority queue, by using stacks and queues
- Can even code one implementation that takes a variable queuing object



Uninformed Search

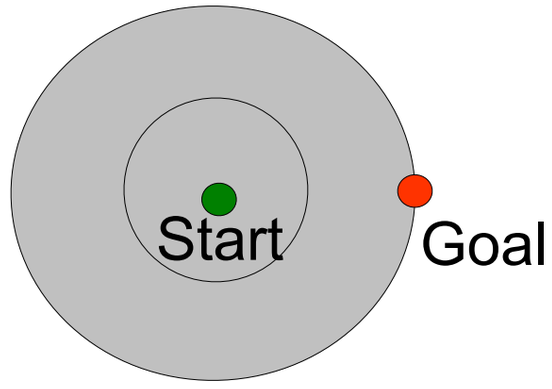
Uniform Cost Search

- Strategy: expand lowest path cost

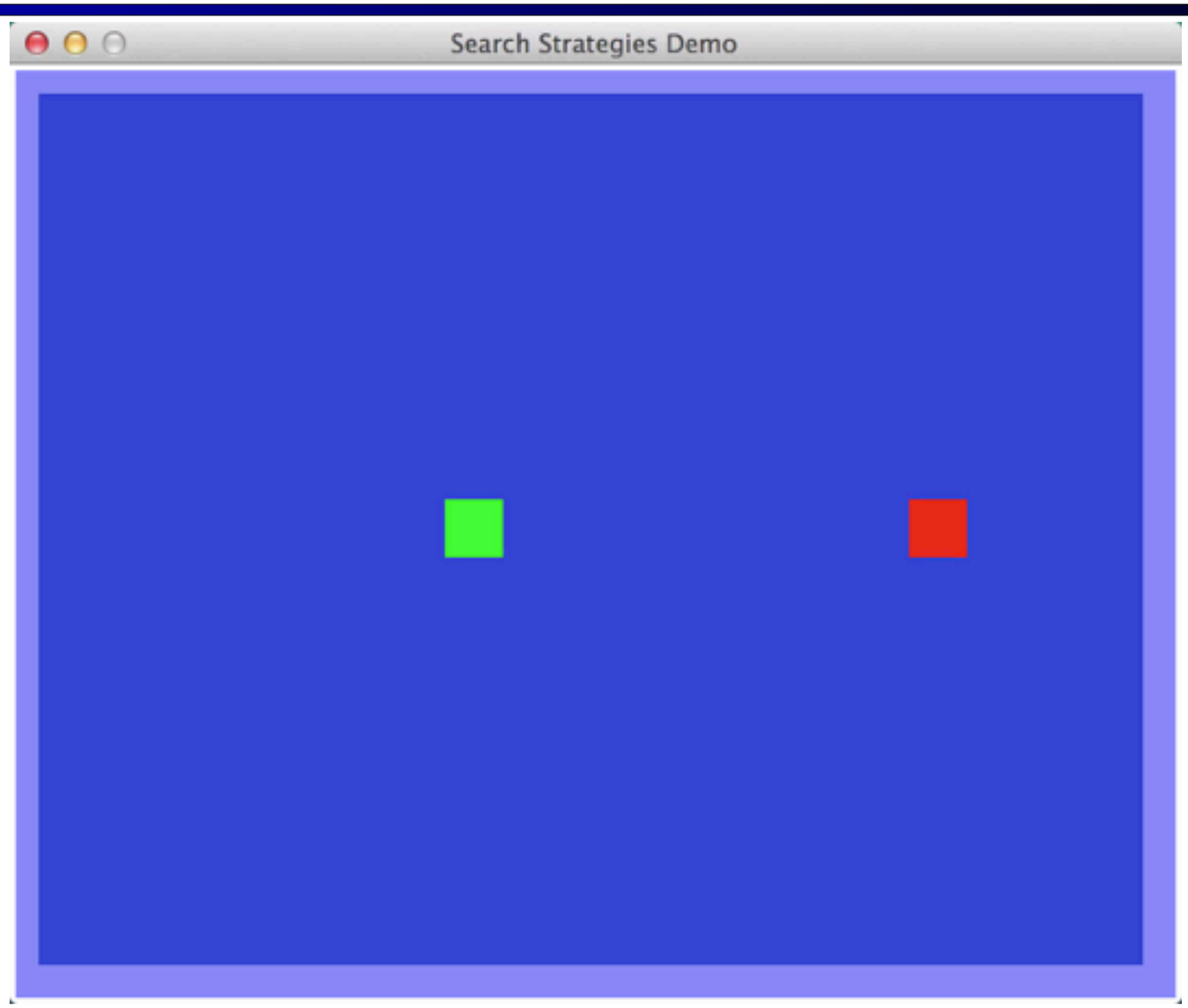


- The good: UCS is complete and optimal!

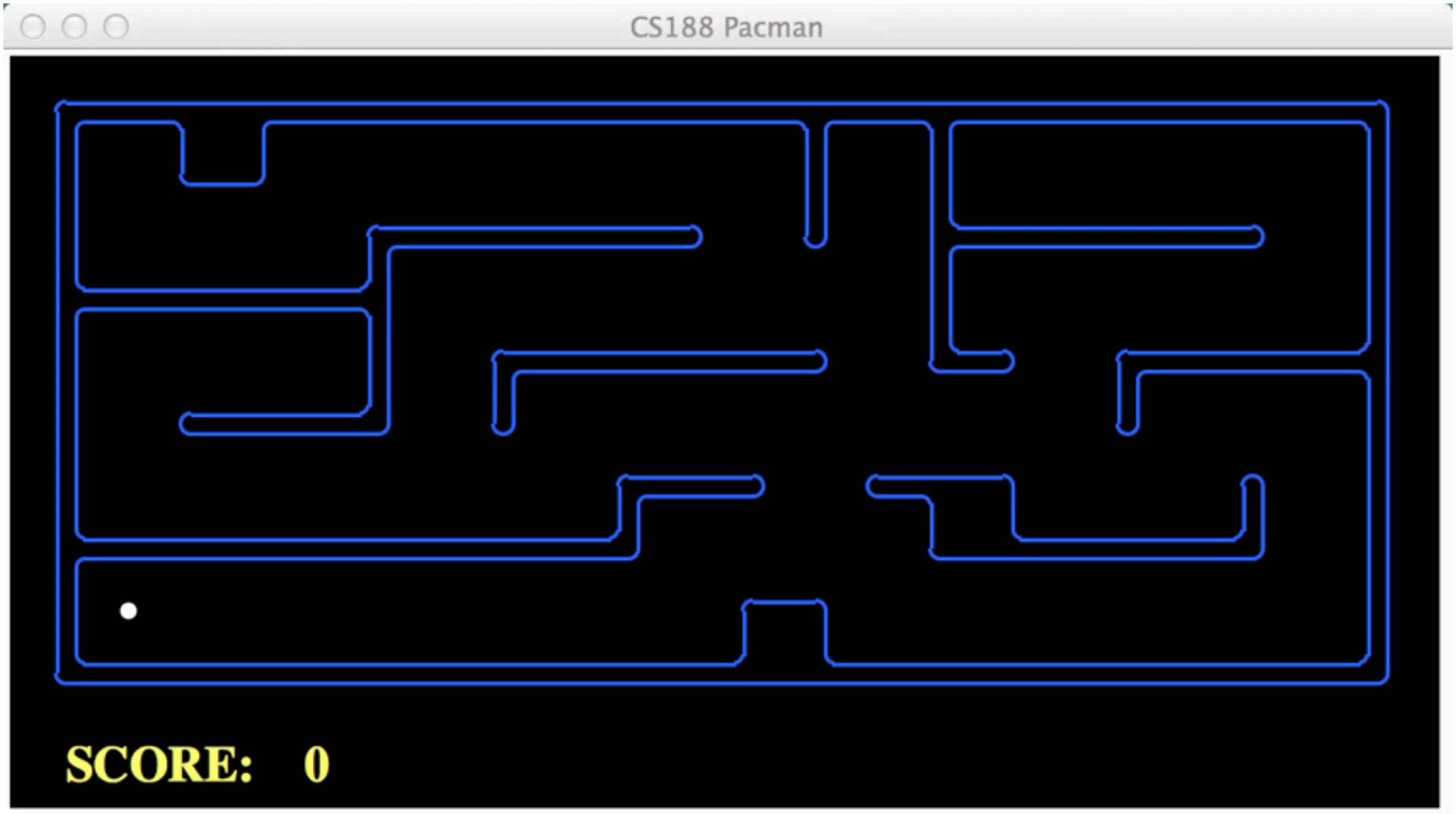
- The bad:
  - Explores options in every “direction”
  - No information about goal location



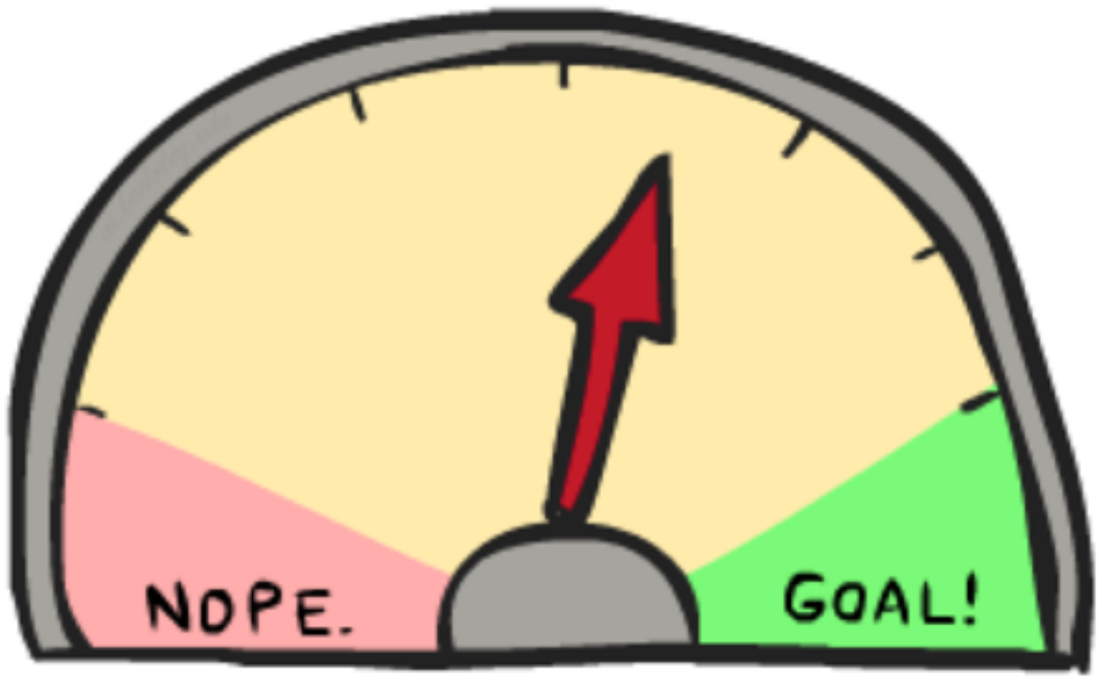
Video of Demo Contours UCS Empty



Video of Demo Contours UCS Pacman Small Maze

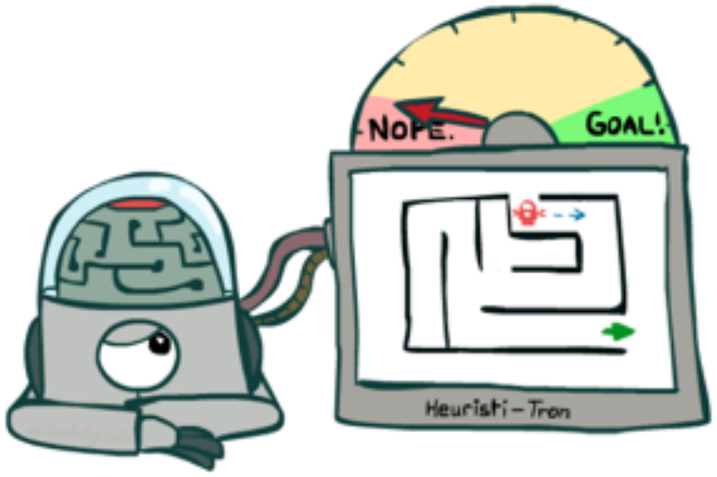


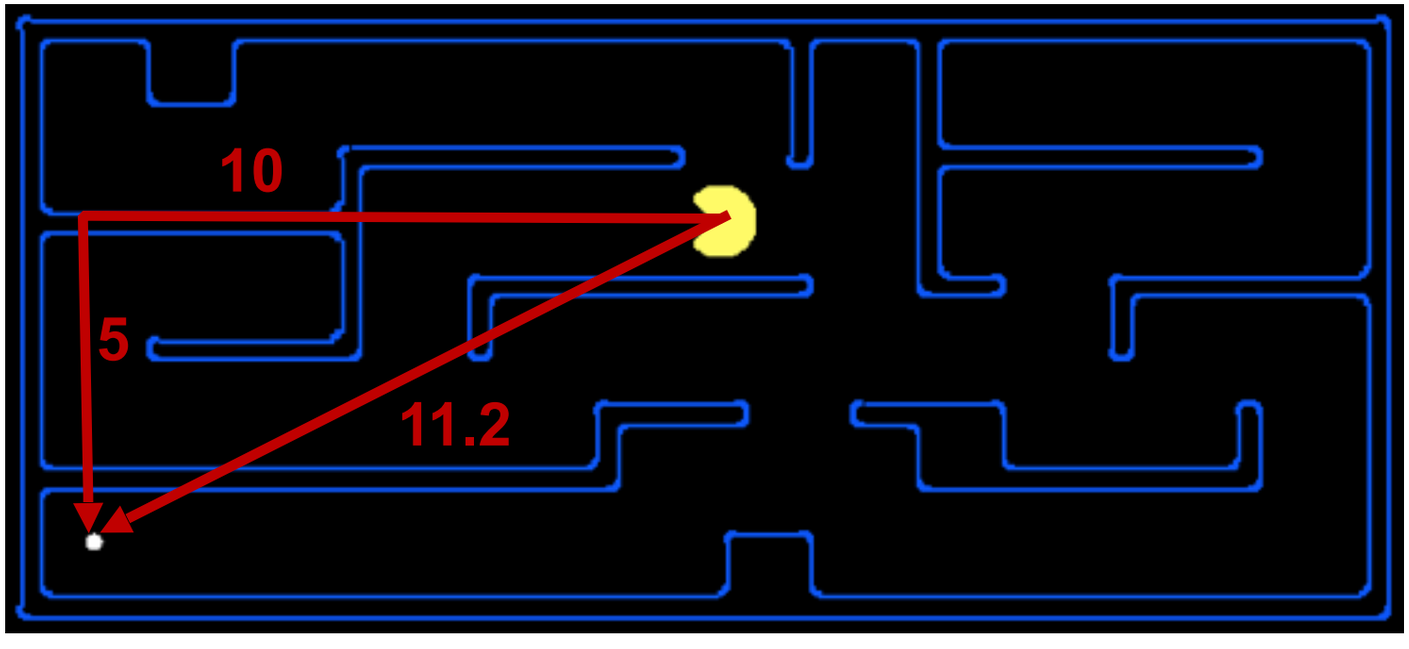
Informed Search

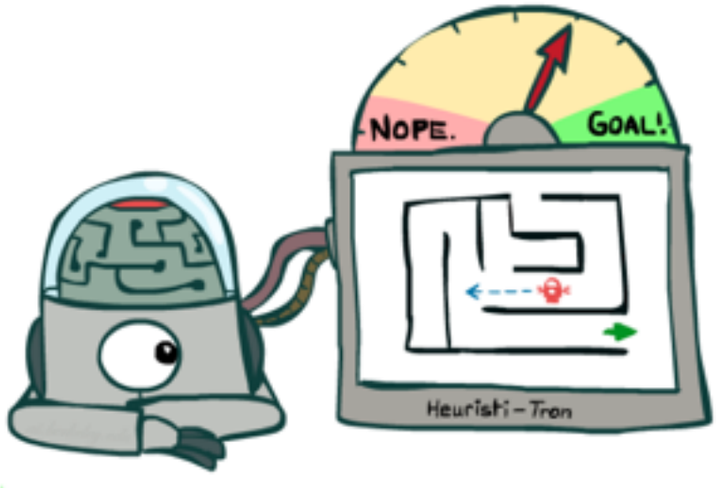


Search Heuristics

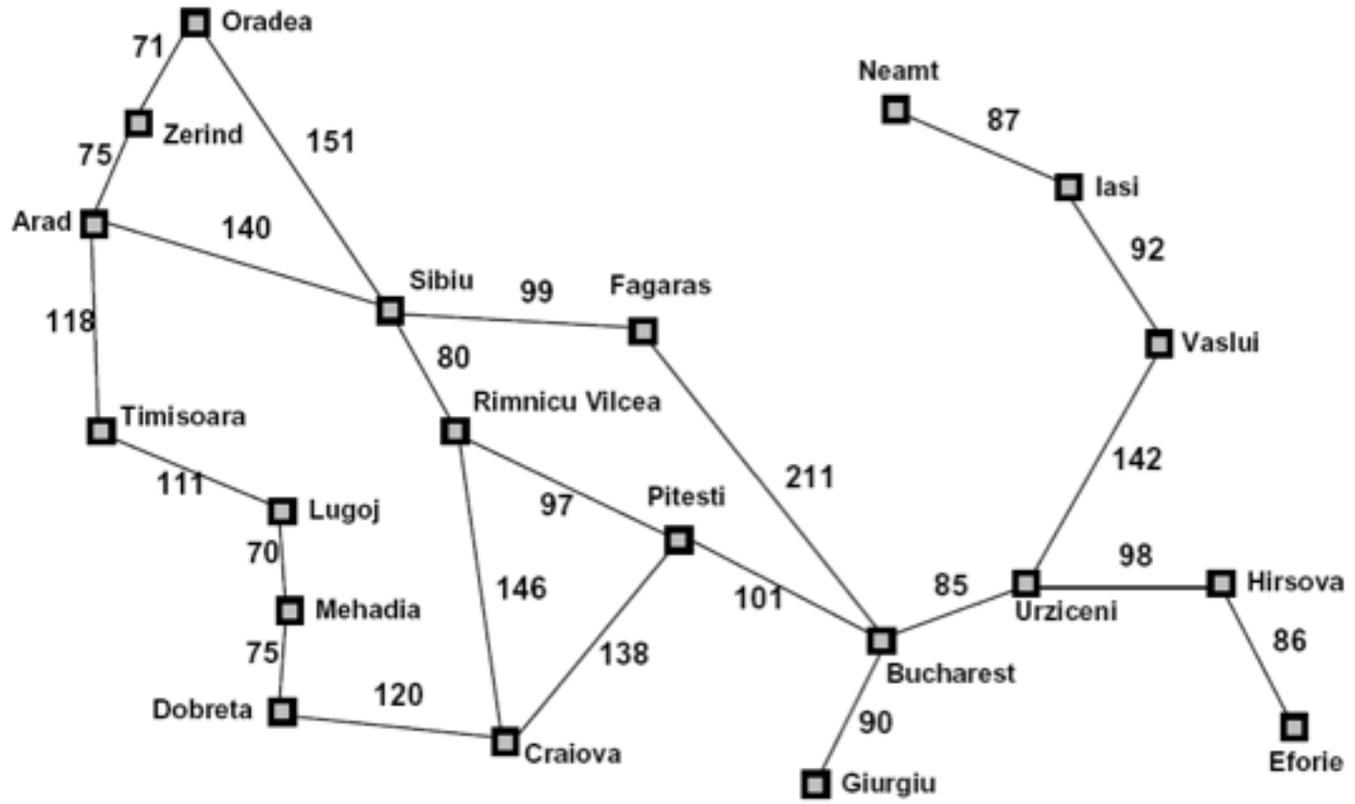
- A heuristic is:
  - A function that *estimates* how close a state is to a goal
  - Designed for a particular search problem
  - Examples: Manhattan distance, Euclidean distance for pathing







Example: Heuristic Function

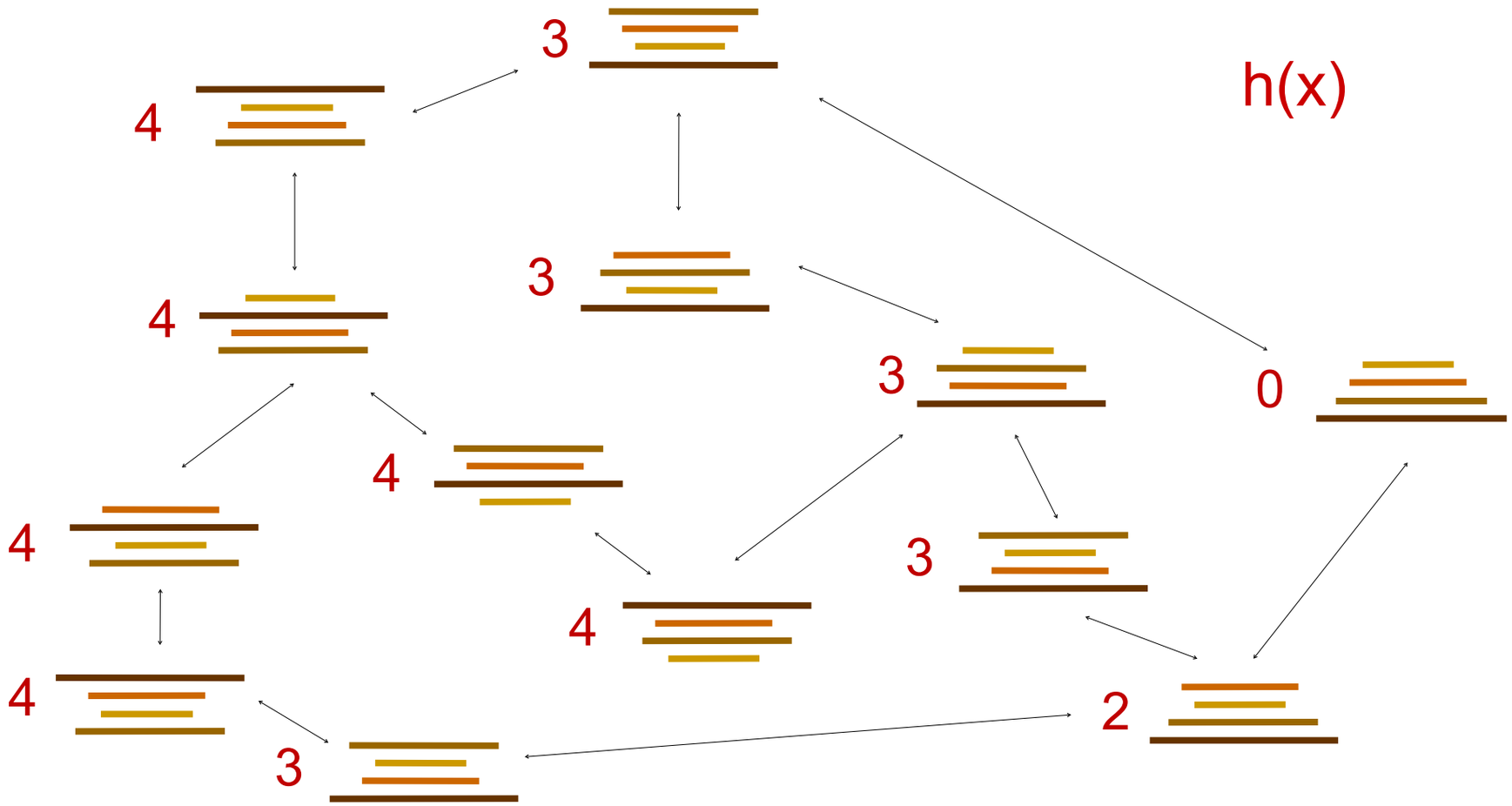


Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

h(x)

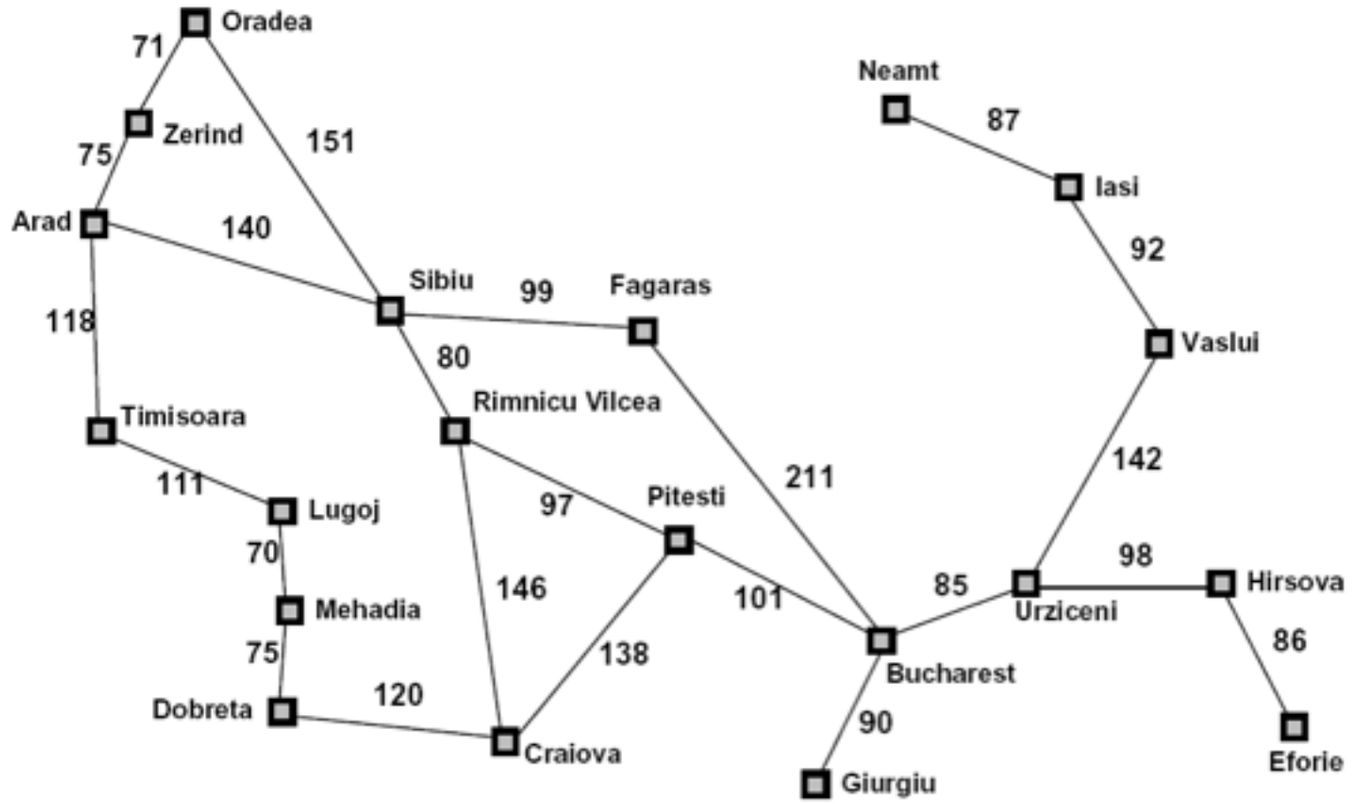
Example: Heuristic Function

Heuristic: the number of the largest pancake that is still out of place



Greedy Search

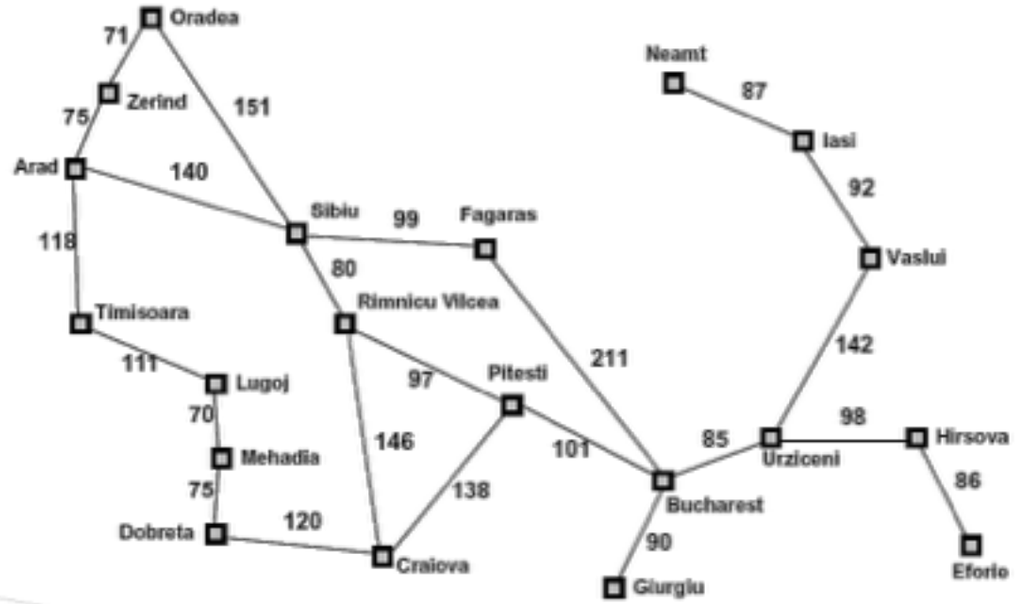
Example: Heuristic Function



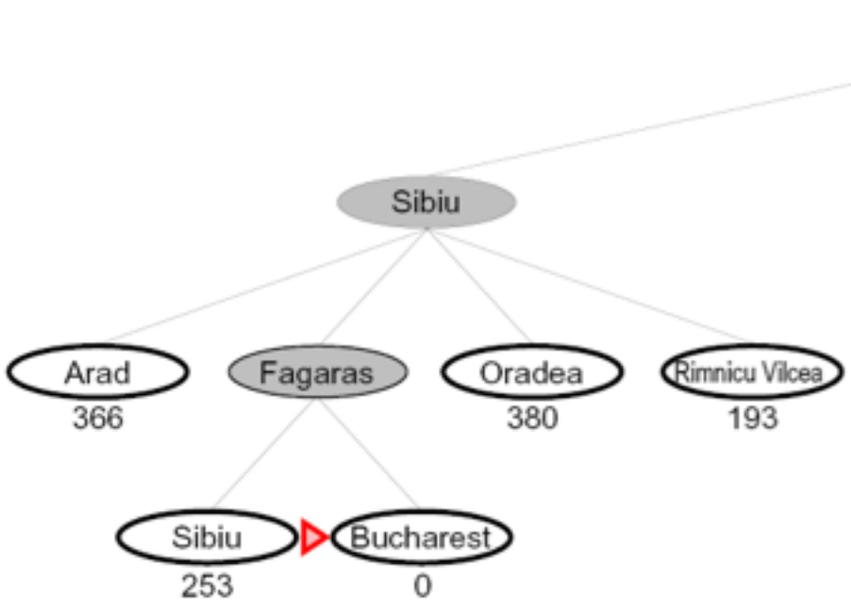
Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

h(x)

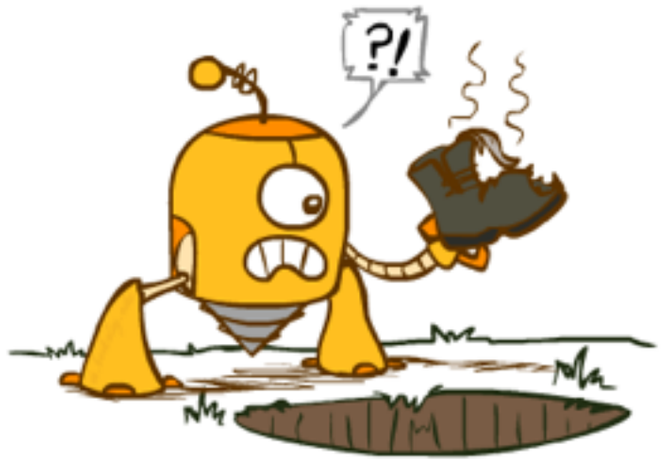
Greedy Search



- Expand the node that seems closest…

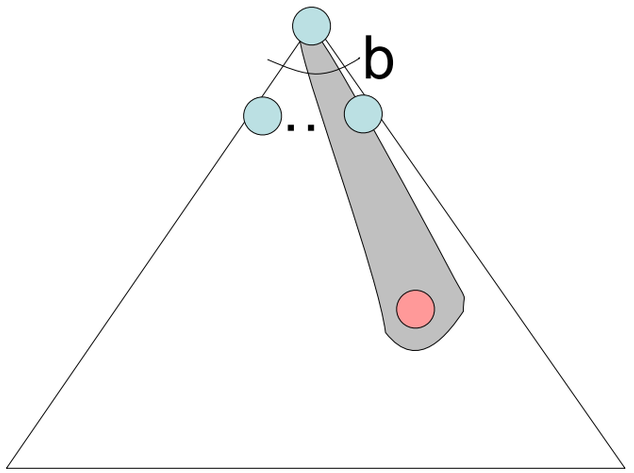


- What can go wrong?

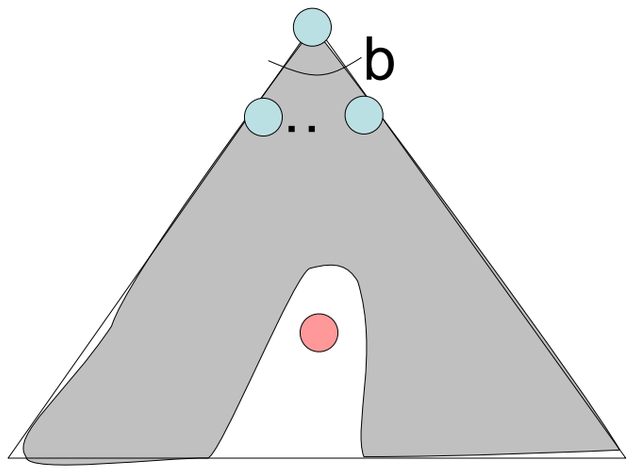


Greedy Search

- Strategy: expand a node that you think is closest to a goal state
  - Heuristic: estimate of distance to nearest goal for each state

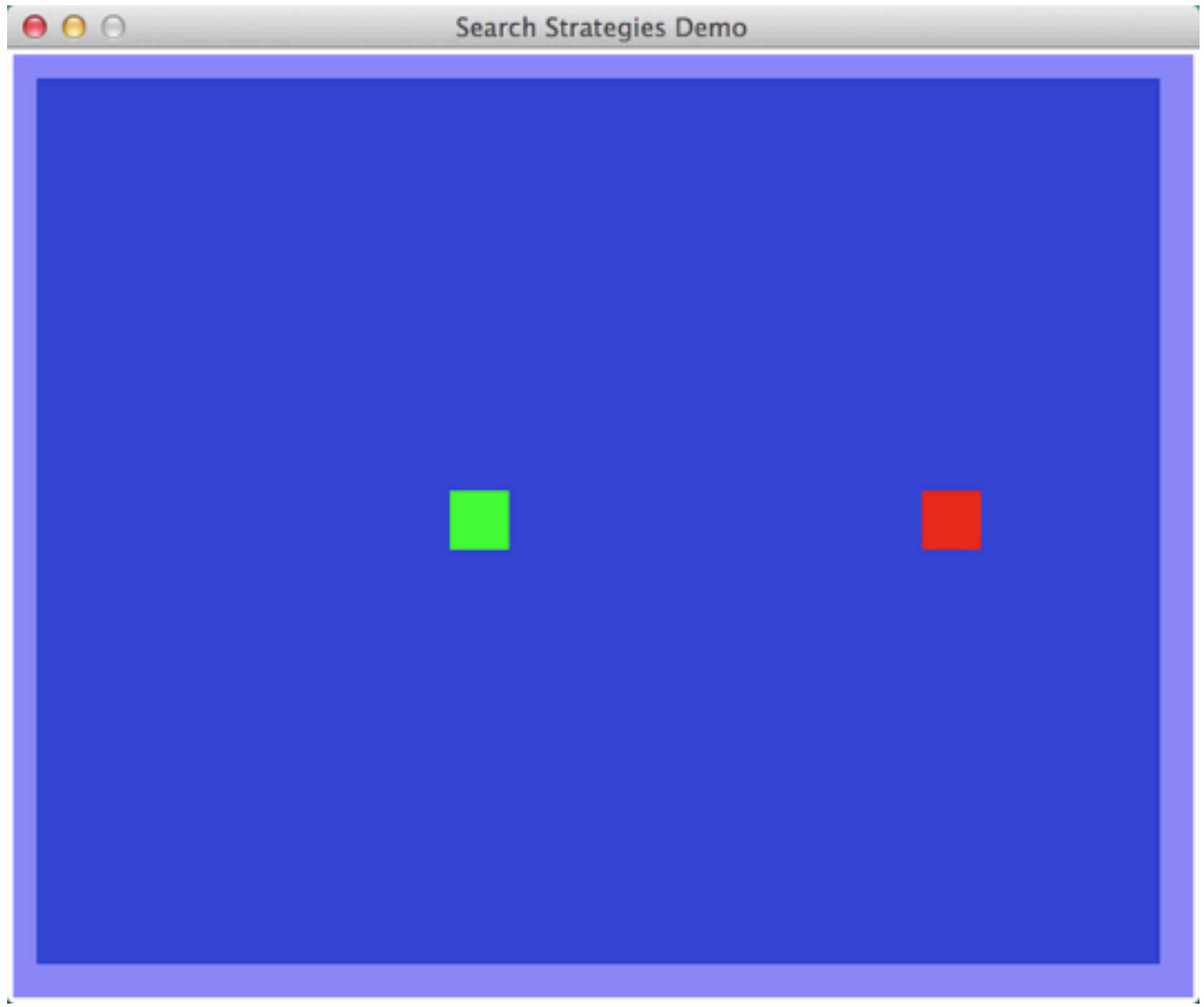


- A common case:
  - Best-first takes you straight to the (wrong) goal

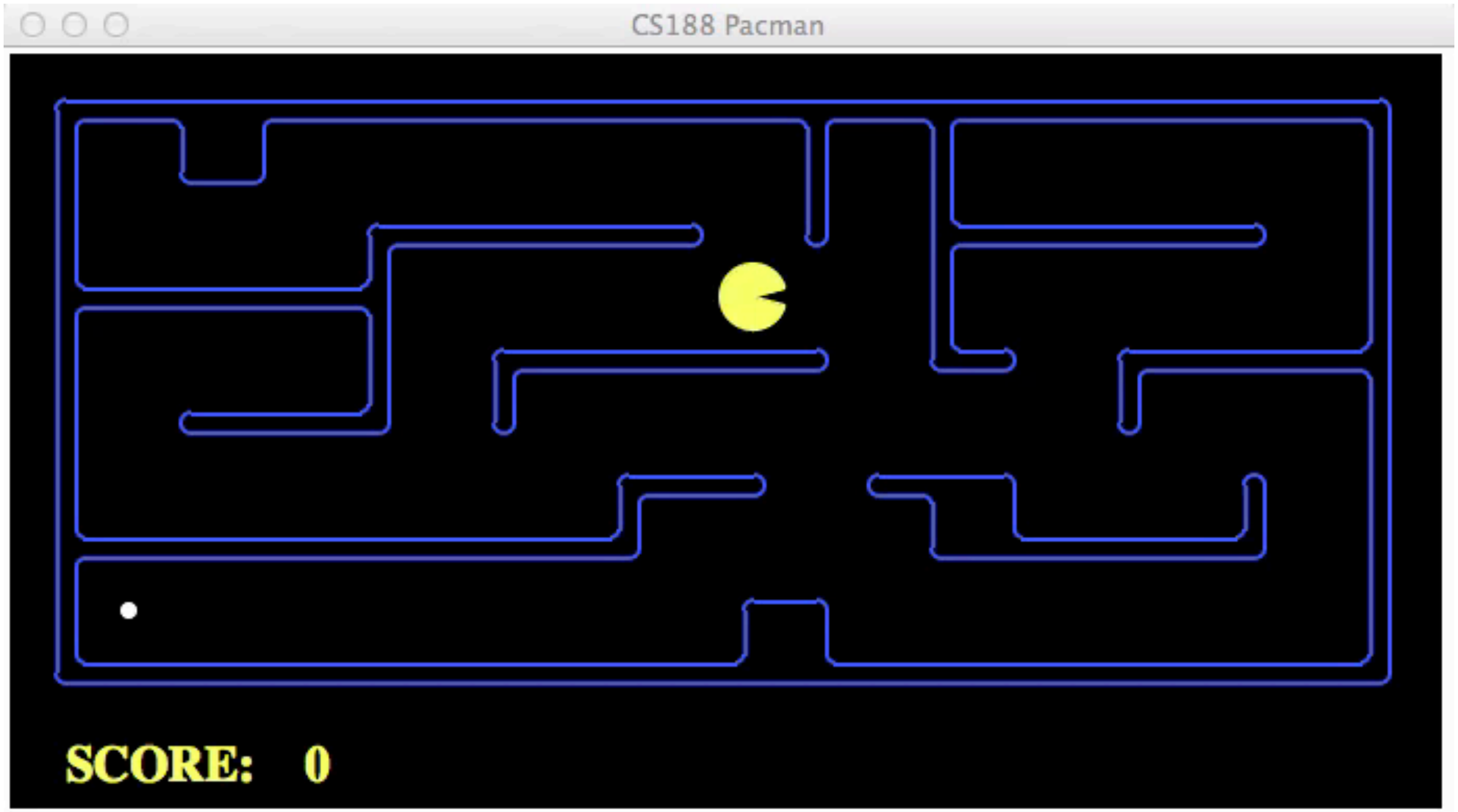


- Worst-case: like a badly-guided DFS

Video of Demo Contours Greedy (Empty)



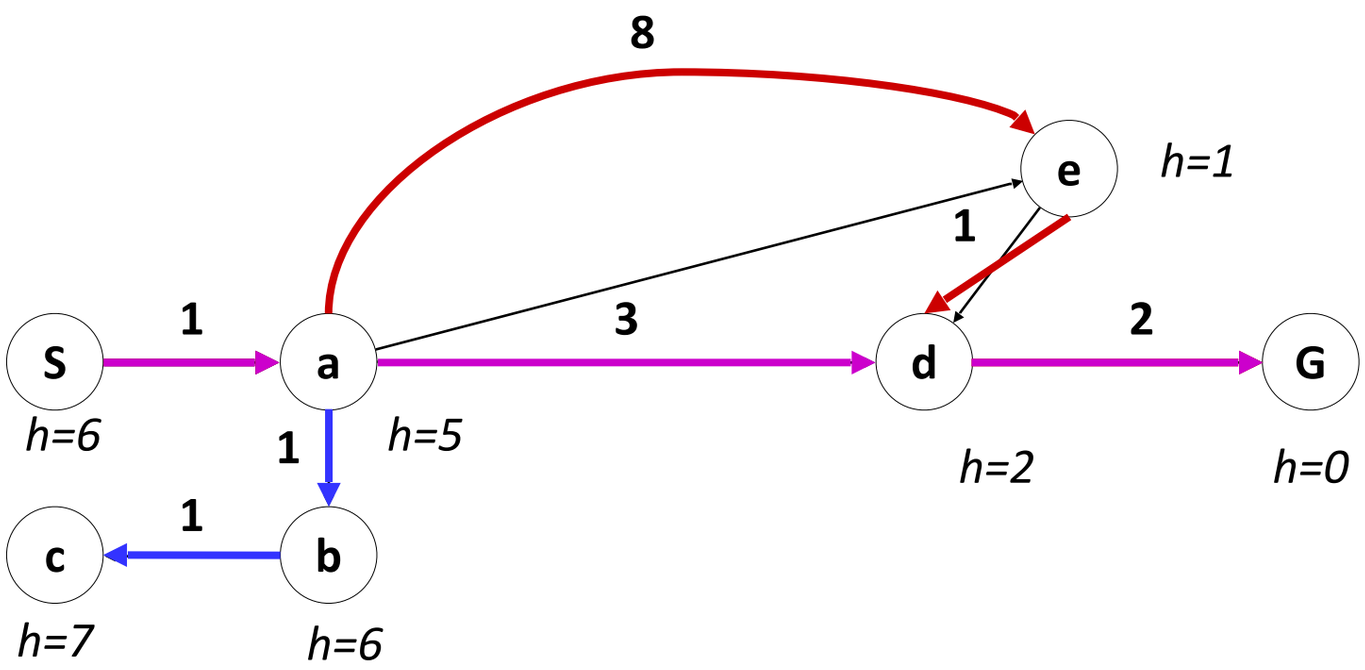
Video of Demo Contours Greedy (Pacman Small Maze)

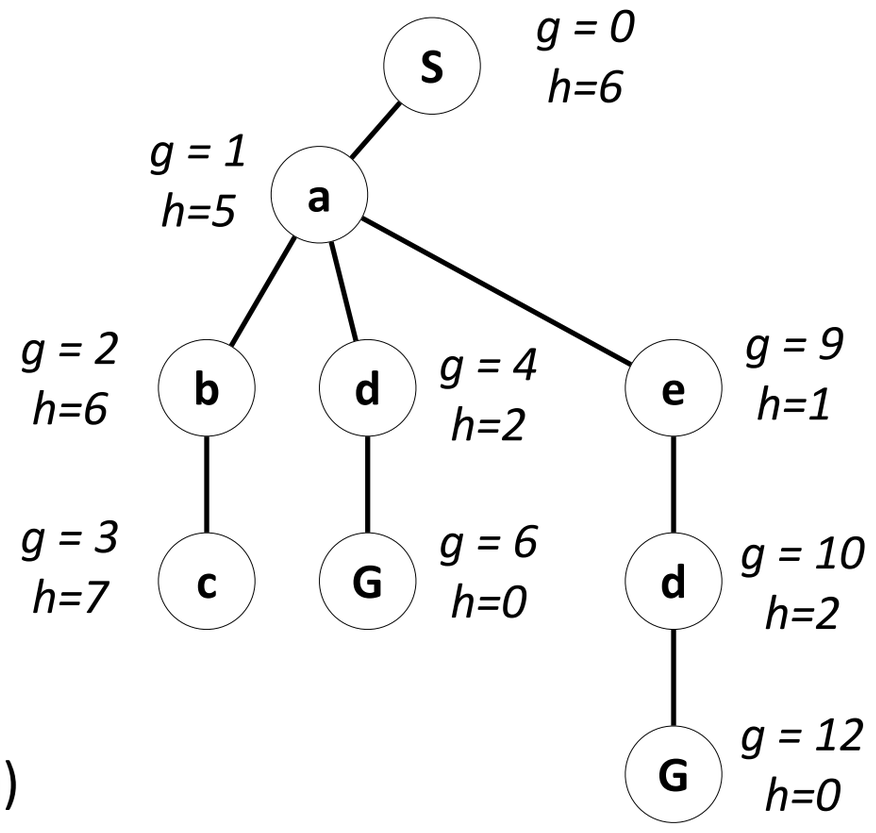


Beam Search

Combining UCS and Greedy

- Uniform-cost orders by path cost, or *backward cost* g(n)
- Greedy orders by goal proximity, or *forward cost* h(n)



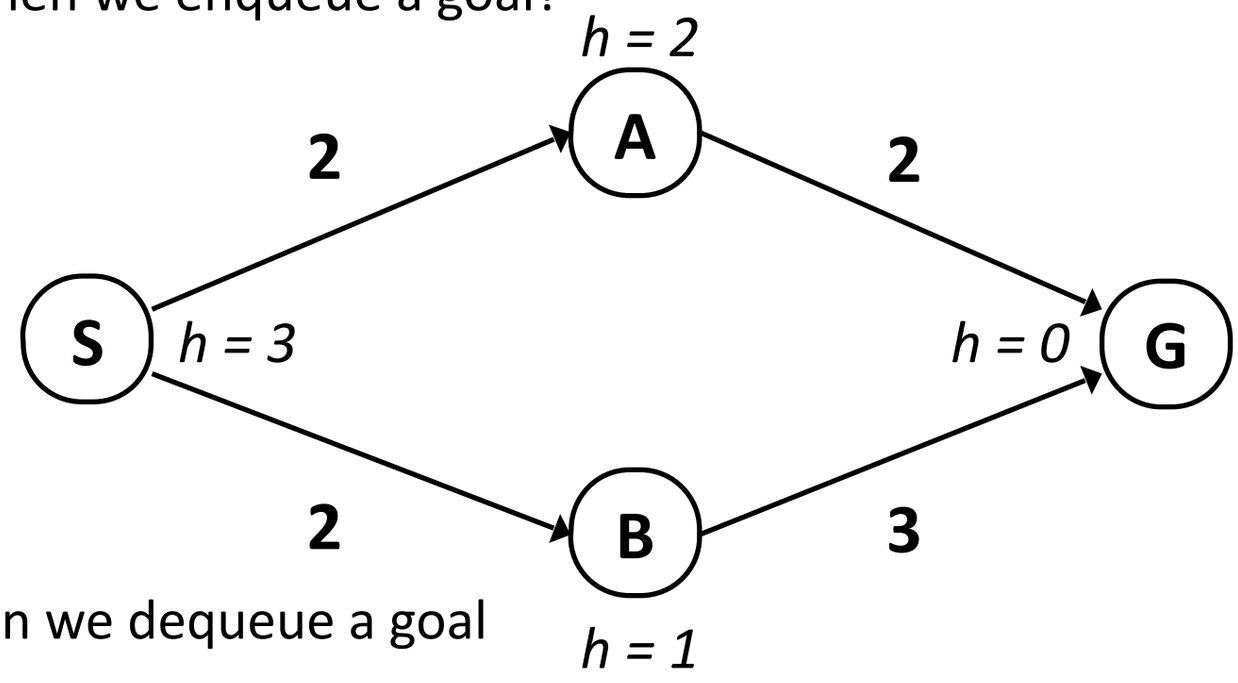


- Beam Search orders by the sum: f(n) = g(n) + h(n)

Example: Teg Grenager

When should Beam search terminate?

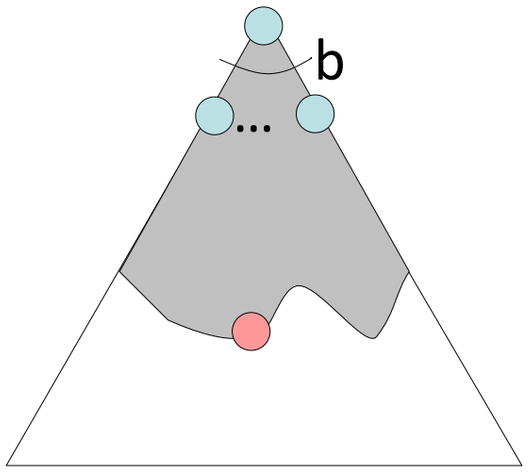
- Should we stop when we enqueue a goal?



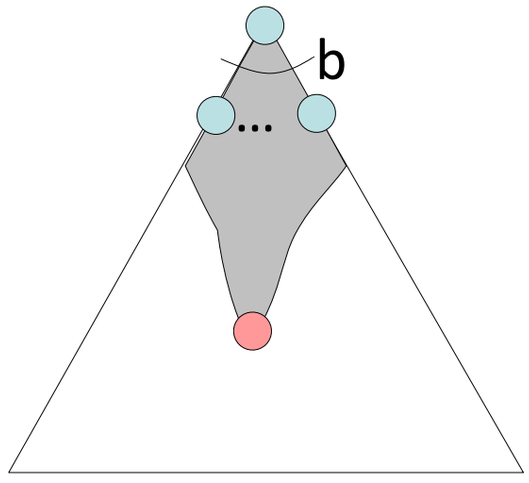
- No: only stop when we dequeue a goal

Properties of Beam Search

Uniform-Cost

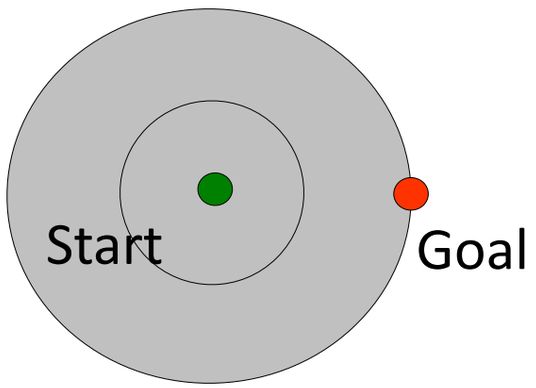


Beam Search

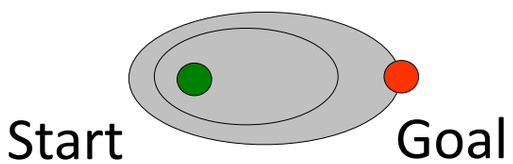


UCS vs Beam Search Contours

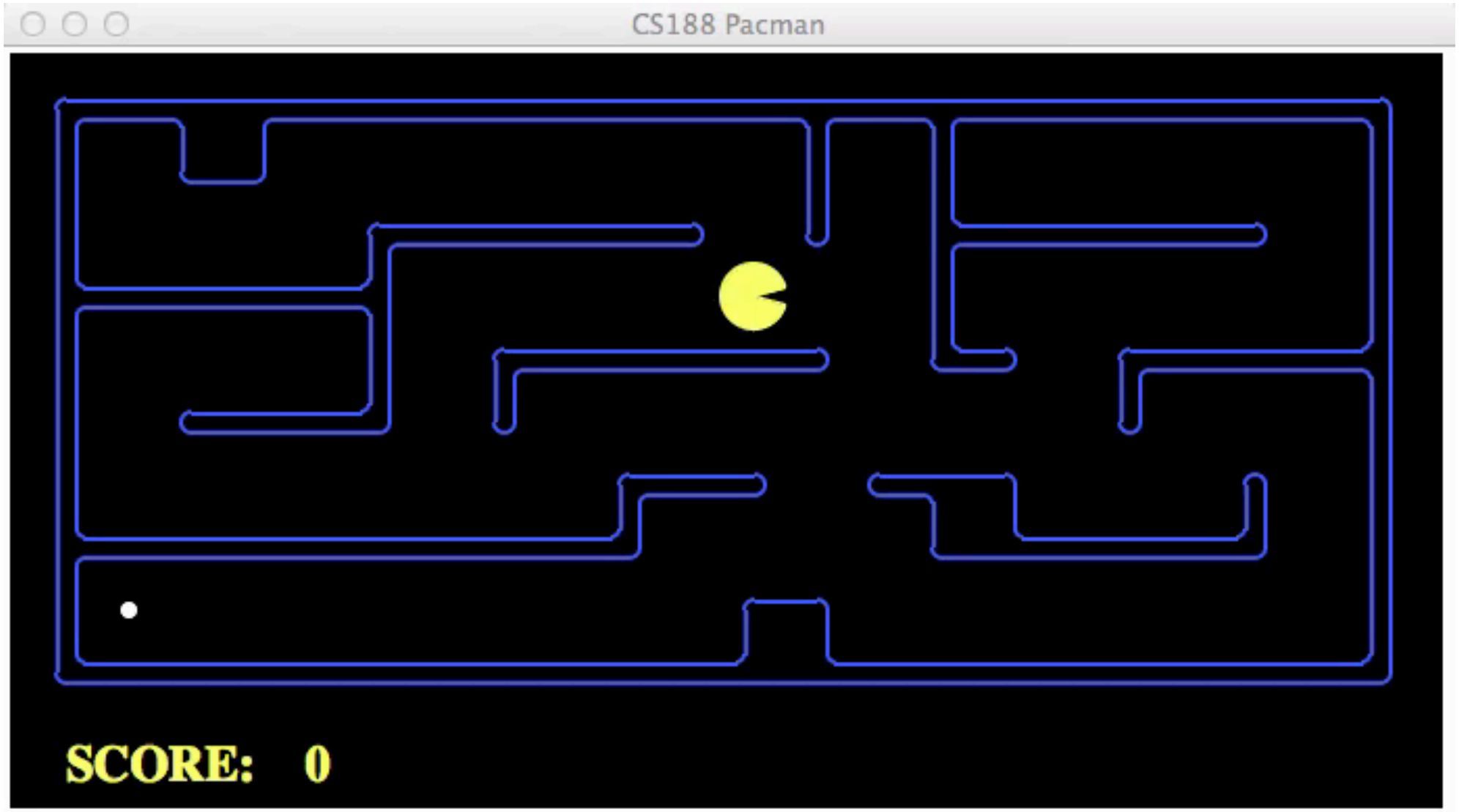
- Uniform-cost expands equally in all "directions"



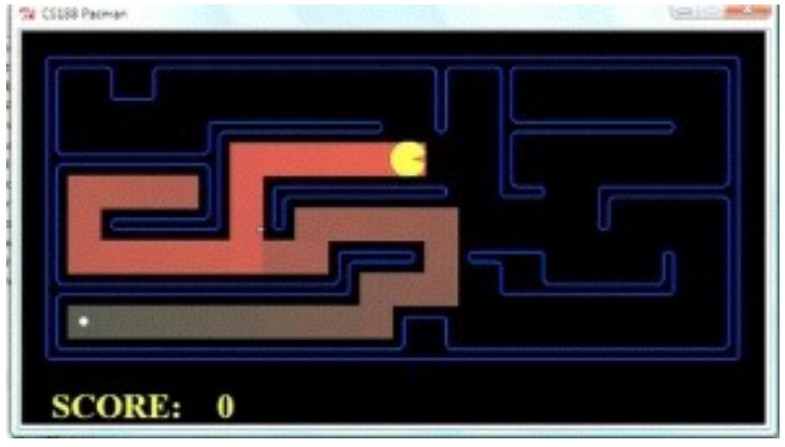
- Beam search expands mainly toward the goal, but does hedge its bets



Video of Demo Contours (Pacman Small Maze) – Beam

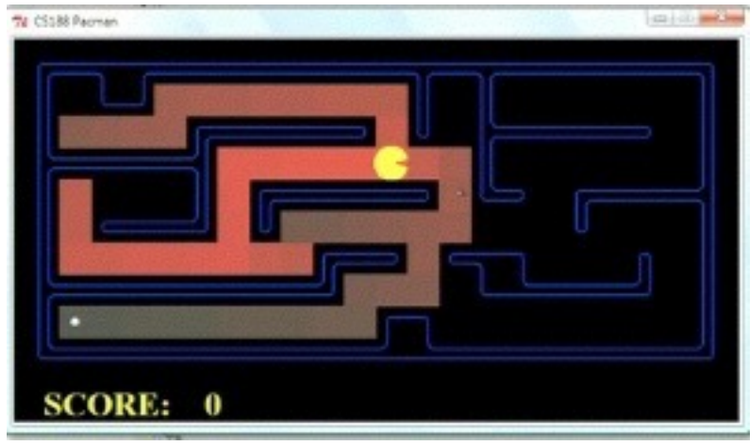


Comparison



Greedy

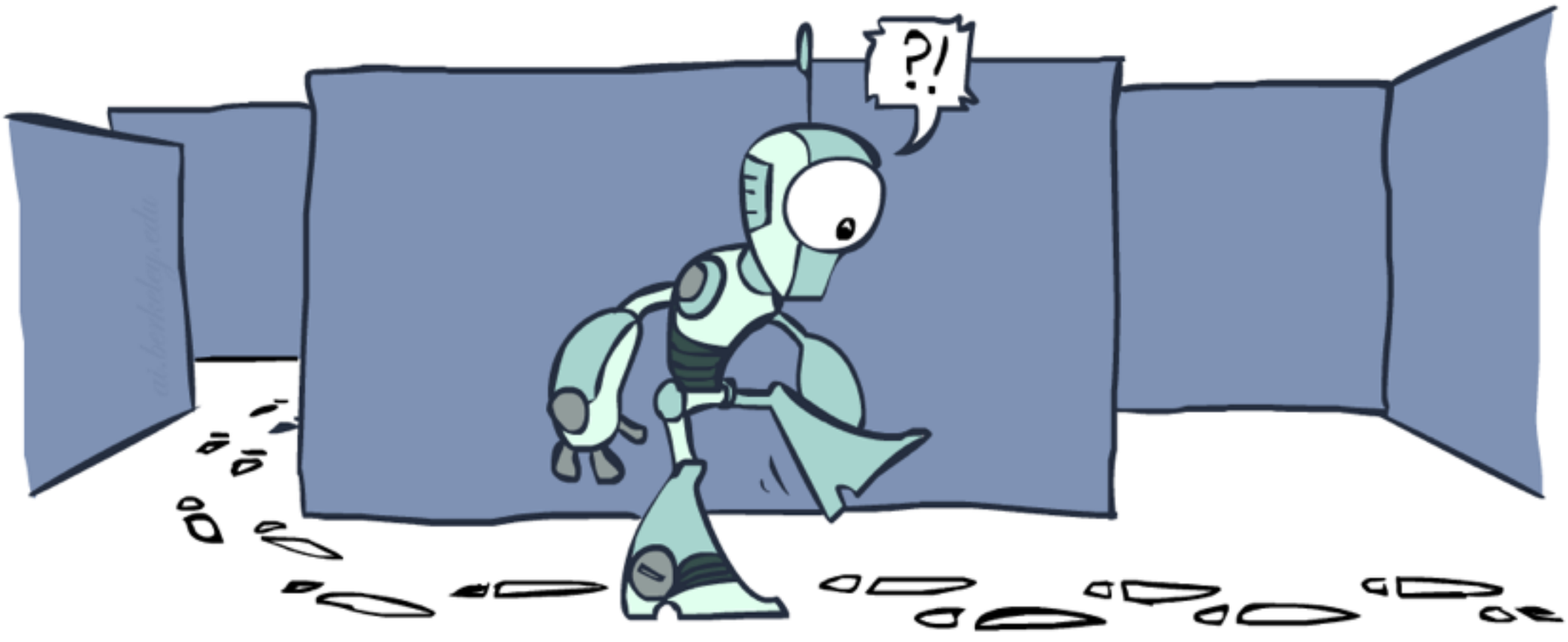
Uniform Cost



Beam Search

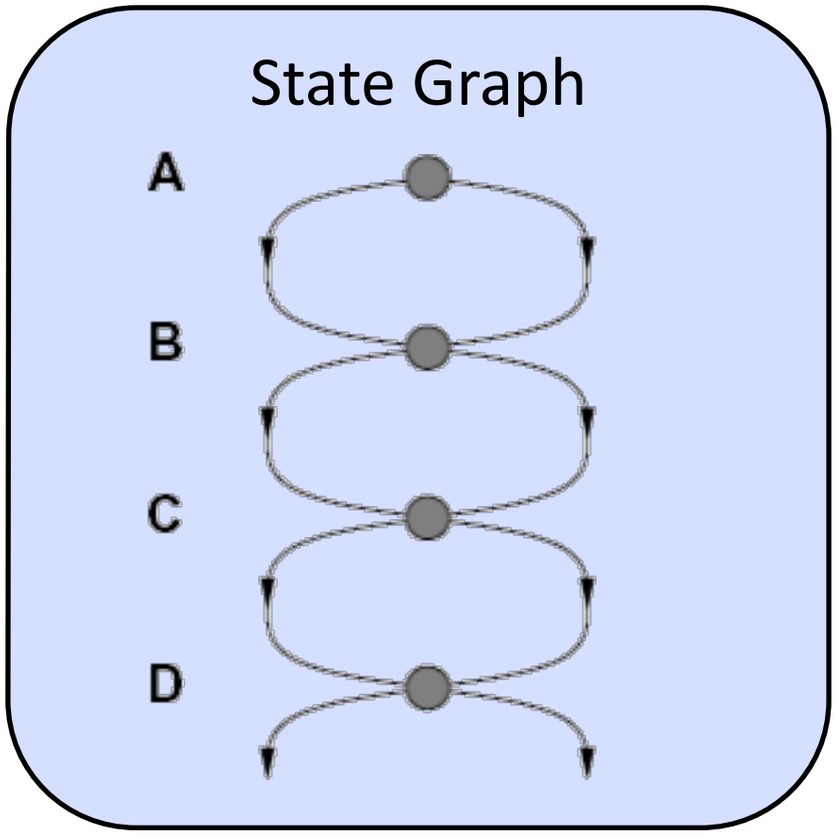
- UCS: >9000 nodes
- Beam: 182 nodes

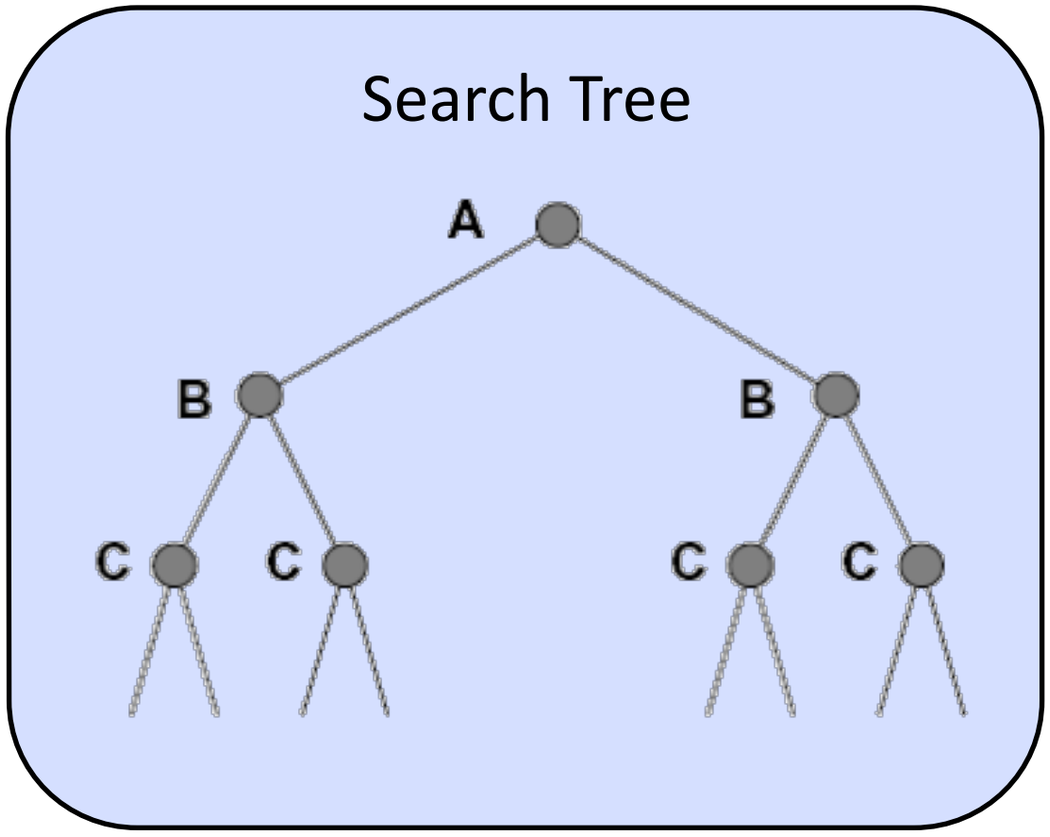
Graph Search



Tree Search: Extra Work!

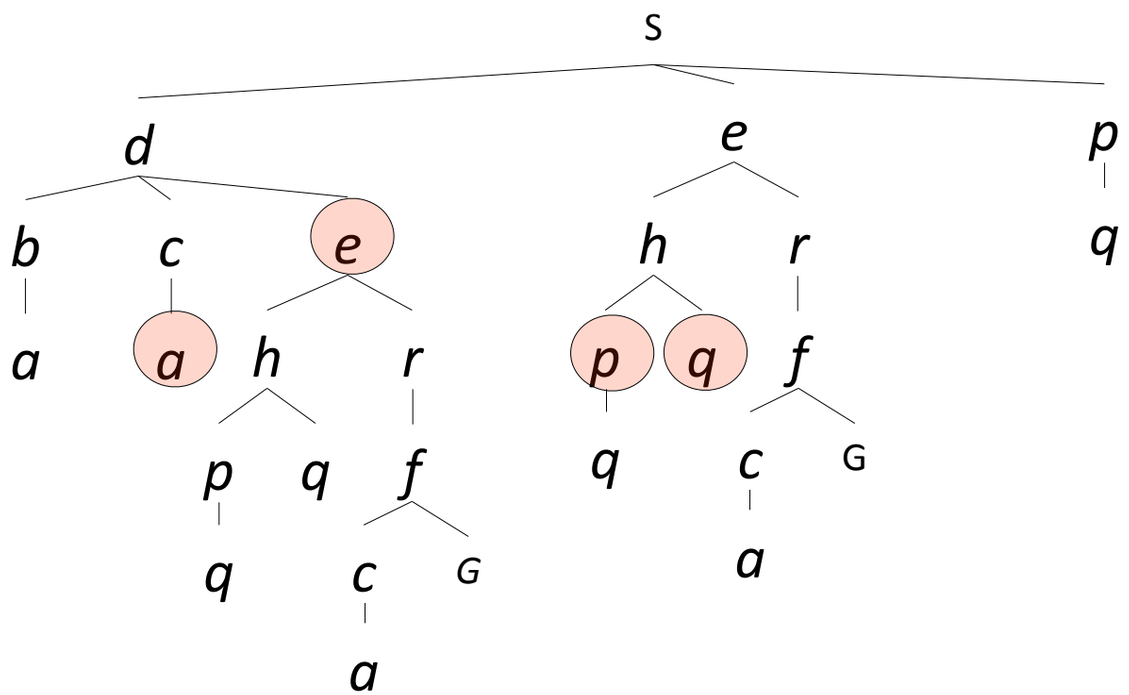
- Failure to detect repeated states can cause exponentially more work.





Graph Search

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



Graph Search

- Idea: never expand a state twice
- How to implement:
  - Tree search + set of expanded states (“closed set”)
  - Expand the search tree node-by-node, but…
  - Before expanding a node, check to make sure its state has never been expanded before
  - If not new, skip it, if new add to closed set
- Important: store the closed set as a set, not a list
- Can graph search wreck completeness? Why/why not?
- How about optimality?

Beam Search: Summary

Beam Search: Summary

- Beam search uses both backward costs and (estimates of) forward costs

- Beam search sacrifices optimality (except under very limited conditions) for massive speed increases