Blindingly Fast Search

Chris Piech

CS 106B
Lecture 26
Nov 30, 2016

Nantes
France
Bay of Biscay
Zurich
Munich
Austria
Budapest
Hungary
Moldova
Cluj-Napoca
Chisinau
Odesa
Switzerland
Geneva
Lyon
Slovenia
Zagreb
Romania
Turin
Milan
Croatia
Bosnia and Herzegovina
Serbia
Bucharest
Toulouse
Monaco
Italy
Marseille
Bilbao
Andorra
Sofia
Bulgaria
Kosovo
Montenegro
Rome
Zaragoza
Barcelona
Macedonia (FYROM)
Plovdiv
Tirana
Albania
Naples
Madrid
Palma de Mallorca
Thessaloniki
Bursa
Spain
Greece
Izmir
Palermo
Athens
Catania
Antalya
Algiers
Tunis
Gibraltar
Oran
Malta
Mediterranean Sea
Fes
Tunisia
Tripoli
Benghazi
Alexandria
Cairo
Algeria
Libya
Egypt

Today's Goal

1. Dijkstra's Search
2. A-Star Search

Today's Route

Search Idea

Todo List

Repeat Until Finished

Todo List

Repeat Until Finished

Take the next path out of the todo list

Todo List

Repeat Until Finished

If it leads to the goal, we are done!

Todo List

Repeat Until Finished

Todo List

Otherwise make paths
with unvisited neighbors

Repeat Until Finished

Todo List

Repeat Until Finished

Todo List

Repeat…

Search Pseudocode

```
Make a todo list of paths
Put a path with just the start in the todo list
While the todo list isn't empty
  1. Take a "currPath" out of the todo list
  2. Call the last node in the currPath "currNode"
  3. If "currNode" is the goal, you are done.
  4. If you have seen currNode before, skip it.
  5. for all unseen neighbors of currNode
       Make a newPath which is currPath + neighbor
       Add newpath to the todo list
```

DFS Pseudocode

```
Make a Stack todo list of paths
Put a path with just the start in the todo list
While the todo list isn't empty
  1. Take a "currPath" out of the todo list
  2. Call the last node in the currPath "currNode"
  3. If "currNode" is the goal, you are done.
  4. If you have seen currNode before, skip it.
  5. for all unseen neighbors of currNode
       Make a newPath which is currPath + neighbor
       Add newpath to the todo list
```

BFS Pseudocode

```
Make a Queue todo list of paths
Put a path with just the start in the todo list
While the todo list isn't empty
  1. Take a "currPath" out of the todo list
  2. Call the last node in the currPath "currNode"
  3. If "currNode" is the goal, you are done.
  4. If you have seen currNode before, skip it.
  5. for all unseen neighbors of currNode
       Make a newPath which is currPath + neighbor
       Add newpath to the todo list
```

Breadth First Search

Todo List

In Breadth First Search the todo list is a Queue

Most Simple Graph

Breadth First Search

Enqueue the start state

Breadth First Search

Dequeue the first state

Breadth First Search

Check if it's the goal or if it has been visited

Breadth First Search

Find each successor state

Breadth First Search

Find each successor state

Breadth First Search

Find each successor state

Breadth First Search

Enqueue the successor states

Breadth First Search

Enqueue the successor states

Breadth First Search

Forget about the dequeued state

Breadth First Search

Back to the start of the while loop

Breadth First Search

Dequeue a state

Breadth First Search

Check if it’s a goal or if the state has been visited

Breadth First Search

Find each successor state

Breadth First Search

Enqueue each successor

Breadth First Search

Enqueue each successor

Breadth First Search

Back to the start of the while loop

Breadth First Search

Dequeue a path

Breadth First Search

Check if it's a goal

You have seen this before…

Word Ladder

Guaranteed Shortest

Guaranteed Shortest

Must explore all paths of length 1, before any paths of length 2

Guaranteed Shortest

Must explore all paths of length 2, before any paths of length 3

Guaranteed Shortest

Must explore all paths of length *k*, before any paths of length *k+1*

Guaranteed Shortest

When you find a path to an unseen node, it must be the shortest

Slimemold BFS

Weights?

Shortest Path

With Weights

With Weights

Realistic

Dijkstra’s

Sounds scary…

… really not so bad

Breadth First Search

Todo List

In Breadth First Search the todo list is a Queue

And thus it sees paths in order of increasing number of "hops"

Dijkstra

Todo List

In Breadth First Search the todo list is a PriorityQueue

And thus it sees paths in order of increasing path cost

BFS Pseudocode

```
Make a Queue to store a todo-list of paths
Put a path with just the start in the todo-list
While the todo-list isn't empty
  1. Take a path out of the todo-list
  2. Call the last node in the path "currNode"
  3. If "currNode" is the goal, you are done.
  4. If you have seen currNode before, skip it.
  5. for all neighbors of currNode
        Make a newPath = path + neighbor
        Add the new path to the todo-list
```

Dijkstra Pseudocode

```
Make a PriorityQueue todo-list of paths
Put a path with just the start in the todo-list
While the todo-list isn't empty
  1. Take a path out of the todo-list
  2. Call the last node in the path "currNode"
  3. If "currNode" is the goal, you are done.
  4. If you have seen currNode before, skip it.
  5. for all neighbors of currNode
        Make a newPath = path + neighbor
        Add the new path to the todo-list
        Priority is the path length!
```

We look at paths in order of *increasing* distance from start

Works with Weights

0

Works with Weights

3

Works with Weights

4

Works with Weights

5

Works with Weights

6

Works with Weights

8

Works with Weights

8

Works with Weights

9

Works with Weights

10

Works with Weights

Imagine a Path From the Todo List

Imagine a Path From the Todo List

If currNode hasn't been dequeued yet, is it possible that there is a path from start to currNode with lower cost?

NO

Imagine a Path From the Todo List

Is it possible that there is a path from start to currNode with lower cost?

NO

Imagine a Path From the Todo List

Is it possible that there is a path from start to currNode with lower cost?

NO

We are looking at paths in order of increasing cost, and we haven't seen currNode yet.
Thus, this is the shortest path to currNode

Common Mistakes

Should we check goal
before putting into todo?

Dijkstra Pseudocode

```
Make a PriorityQueue todo-list of paths
Put a path with just the start in the todo-list
While the todo-list isn't empty
  1. Take a path out of the todo-list
  2. Call the last node in the path "currNode"
  3. If "currNode" is the goal, you are done.
  4. If you have seen currNode before, skip it.
  5. for all neighbors of currNode
        Make a newPath = path + neighbor
        Add the new path to the todo-list
        Priority is the path length!
```

Bug!

```
Make a PriorityQueue todo-l
Put a path with just the st
While the todo-list isn't e
  1. Take a path out of the
  2. Call the last node in
  3. If "currNode" is the g
  4. If you have seen currNode before, skip it.
  5. for all neighbors of currNode
       Make a newPath = path + neighbor
       Add the new path to the todo-list
```

Return if neighbor is the goal

No

Check Goal After Dequeue

Check Goal After Dequeue

Check Goal After Dequeue

Check Goal After Dequeue

But this isn't the shortest path

Does Dijkstra work with negative edges?

Check Goal After Dequeue

Dijkstra's checks paths in order of distance from start

Check Goal After Dequeue

Dijkstra's checks paths in order of distance from start

Check Goal After Dequeue

Dijkstra's checks paths in order of distance from start

Check Goal After Dequeue

It will miss this path

No

Check Goal After Dequeue

And what about cycles?

No

What is the priority of the first path?

Dijkstra Pseudocode

```
Make a PriorityQueue todo-list of paths
Put a path with just the start in the todo-list
While the todo-list isn't empty
  1. Take a path out of the todo-list
  2. Call the last node in the path "currNode"
  3. If "currNode" is the goal, you are done.
  4. If you have seen currNode before, skip it.
  5. for all neighbors of currNode
        Make a newPath = path + neighbor
        Add the new path to the todo-list
        Priority is the path length!
```

What is up with the skipping part?

Dijkstra Pseudocode

```
Make a PriorityQueue todo-list of paths
Put a path with just the start in the todo-list
While the todo-list isn't empty
  1. Take a path out of the todo-list
  2. Call the last node in the path "currNode"
  3. If "currNode" is the goal, you are done.
  4. If you have seen currNode before, skip it.
  5. for all neighbors of currNode
        Make a newPath = path + neighbor
        Add the new path to the todo-list
        Priority is the path length!
```

If you have seen a node before…

Then a previous path had a shorter route to currNode

Skip this path!

That’s all for Dijkstra’s

Let’s Play!

See It Live

Path hops: 16

Path cost: 16

Today's Route

Today's Route

Dijkstra where each edge has cost 1

1?
1?
1?
1?

1?
1?
1?
1?

1?
1?
1?
1

1?
1?
1?
2?
1
2?
2?

1?
1?
1?
2?
1
2?
2?

1?
1
1?
2?
1
2?
2?

2?
1?
2?
1
1?
2?
1
2?
2?

2?
1?
2?
1
1?
2?
1
2?
2?

2?
1
2?
1
1?
2?
1
2?
2?

2?
2?
1
2?
2?
1
1?
2?
1
2?
2?

2?
2?
1
2?
2?
1
1
2?
1
2?
2?

2?
2? 1 2?
2? 1 1 2?
2? 1 2?
2?

2?
2?
1
2
2?
1
1
2?
2?
1
2?
2?

2? 3?
2? 1 2 3?
2? 1 1 2?
2? 1 2?
2?

2? 3?
2? 1 2 3?
2? 1 1 2?
2? 1 2
2?

2? 3?
2? 1 2 3?
2? 1 1 2?
2? 1 2 3?
2? 3?

2? 3?
2? 1 2 3?
2? 1 1 2
2? 1 2 3?
2? 3?

2? 3?
2? 1 2 3?
2? 1 1 2 3?
2? 1 2 3?
2? 3?

2? 3?
2? 1 2 3?
2? 1 1 2 3?
2 1 2 3?
2? 3?

2? 3?
2? 1 2 3?
2? 1 1 2 3?
3? 2 1 2 3?
3? 2? 3?

2? 3?
2? 1 2 3?
2? 1 1 2 3?
3? 2 1 2 3?
3? 2 3?
3?

2? 3?
3? 2? 1 2 3?
3? 2 1 1 2 3?
3? 2 1 2 3?
3? 2 3?
3?

3? 2? 3?
3? 2 1 2 3?
3? 2 1 1 2 3?
3? 2 1 2 3?
3? 2 3?
3?

3?
3? 2 3?
3? 2 1 2 3?
3? 2 1 1 2 3?
3? 2 1 2 3?
3? 2 3?
3?

3?
3? 2 3?
3? 2 1 2 3?
3? 2 1 1 2 3?
3? 2 1 2 3?
3? 2 3 4?
3? 4?

3?
3? 2 3?
3? 2 1 2 3?
3? 2 1 1 2 3?
3? 2 1 2 3 4?
3? 2 3 4?
3? 4?

3?
3? 2 3?
3? 2 1 2 3?
3? 2 1 1 2 3?
4? 3 2 1 2 3 4?
4? 3? 2 3 4?
3? 4?

3?
3? 2 3?
4? 3? 2 1 2 3?
4? 3 2 1 1 2 3?
4? 3 2 1 2 3 4?
4? 3? 2 3 4?
3? 4?

3? 4?
3? 2 3 4?
4? 3? 2 1 2 3?
4? 3 2 1 1 2 3?
4? 3 2 1 2 3 4?
4? 3? 2 3 4?
3? 4?

3? 4?
4? 3? 2 3 4?
4? 3 2 1 2 3?
4? 3 2 1 1 2 3?
4? 3 2 1 2 3 4?
4? 3? 2 3 4?
3? 4?

3? 4?
4? 3? 2 3 4?
4? 3 2 1 2 3 4?
4? 3 2 1 1 2 3?
4? 3 2 1 2 3 4?
4? 3? 2 3 4?
3? 4?

4? 3? 4?
4? 3 2 3 4?
4? 3 2 1 2 3 4?
4? 3 2 1 ★ 1 2 3? ★
4? 3 2 1 2 3 4?
4? 3? 2 3 4?
3? 4?

4?
4? 3 4?
4? 3 2 3 4?
4? 3 2 1 2 3 4?
4? 3 2 1 1 2 3?
4? 3 2 1 2 3 4?
4? 3? 2 3 4?
3? 4?

4?
4? 3 4?
4? 3 2 3 4?
4? 3 2 1 2 3 4?
4? 3 2 1 1 2 3?
4? 3 2 1 2 3 4?
4? 3 2 3 4?
4? 3? 4?

4?
4? 3 4?
4? 3 2 3 4?
4? 3 2 1 2 3 4?
4? 3 2 1 1 2 3?
4? 3 2 1 2 3 4?
4? 3 2 3 4?
4? 3 4?
4?

4?
4? 3 4?
4? 3 2 3 4?
4? 3 2 1 2 3 4?
4? 3 2 1 1 2 3 4?
4? 3 2 1 2 3 4?
4? 3 2 3 4?
4? 3 4?
4?

4?
5? 4? 3 4?
5? 4 3 2 3 4?
4? 3 2 1 2 3 4?
4? 3 2 1 1 2 3 4?
4? 3 2 1 2 3 4?
4? 3 2 3 4?
4? 3 4?
4?

				4?									
		5?	4?	3	4?	5?							
	5?	4	3	2	3	4	5?						
	4?	3	2	1	2	3	4?						
4?	3	2	1	☆	1	2	3	4?	☆				
	4?	3	2	1	2	3	4?						
		4?	3	2	3	4?							
			4?	3	4?								
				4?									

				4?									
		5?	4?	3	4?	5?							
	5?	4	3	2	3	4	5?						
	4?	3	2	1	2	3	4?	5?					
4?	3	2	1	★	1	2	3	4	5?				
	4?	3	2	1	2	3	4?	5?					
		4?	3	2	3	4?							
			4?	3	4?								
				4?									

				4?									
		5?	4?	3	4?	5?							
	5?	4	3	2	3	4	5?						
	4?	3	2	1	2	3	4	5?					
4?	3	2	1	★	1	2	3	4	5?				
	4?	3	2	1	2	3	4?	5?					
		4?	3	2	3	4?							
			4?	3	4?								
				4?									

				4?									
		5?	4?	3	4?	5?							
	5?	4	3	2	3	4	5?						
	4?	3	2	1	2	3	4	5?					
4?	3	2	1	★	1	2	3	4	5?				
	4?	3	2	1	2	3	4?	5?					
	5?	4	3	2	3	4?							
		5?	4?	3	4?								
				4?									

				4?									
		5?	4?	3	4?	5?							
	5?	4	3	2	3	4	5?						
5?	4	3	2	1	2	3	4	5?					
4?	3	2	1	★	1	2	3	4	5?				
	4?	3	2	1	2	3	4?	5?					
	5?	4	3	2	3	4?							
		5?	4?	3	4?								
				4?									

				4?									
		5?	4?	3	4?	5?							
	5?	4	3	2	3	4	5?						
5?	4	3	2	1	2	3	4	5?					
4	3	2	1	★	1	2	3	4	5?				
5?	4?	3	2	1	2	3	4?	5?					
	5?	4	3	2	3	4?							
		5?	4?	3	4?								
				4?									

				4?									
		5?	4?	3	4?	5?							
	5?	4	3	2	3	4	5?						
5?	4	3	2	1	2	3	4	5?					
4	3	2	1	★	1	2	3	4	5?				
5?	4	3	2	1	2	3	4?	5?					
	5?	4	3	2	3	4?							
		5?	4?	3	4?								
				4?									

			5?	4?									
		5?	4	3	4?	5?							
	5?	4	3	2	3	4	5?						
5?	4	3	2	1	2	3	4	5?					
4	3	2	1	★	1	2	3	4	5?				
5?	4	3	2	1	2	3	4?	5?					
	5?	4	3	2	3	4?							
		5?	4?	3	4?								
				4?									

			5?	4?									
		5?	4	3	4?	5?							
	5?	4	3	2	3	4	5?						
5?	4	3	2	1	2	3	4	5?					
4	3	2	1	★	1	2	3	4	5?				
5?	4	3	2	1	2	3	4	5?					
	5?	4	3	2	3	4?	5?						
		5?	4?	3	4?								
				4?									

			5?	4?									
		5?	4	3	4?	5?							
	5?	4	3	2	3	4	5?						
5?	4	3	2	1	2	3	4	5?					
4	3	2	1	★	1	2	3	4	5?				
5?	4	3	2	1	2	3	4	5?					
	5?	4	3	2	3	4?	5?						
		5?	4?	3	4	5?							
				4?	5?								

			5?	4?									
		5?	4	3	4?	5?							
	5?	4	3	2	3	4	5?						
5?	4	3	2	1	2	3	4	5?					
4	3	2	1	★	1	2	3	4	5?				
5?	4	3	2	1	2	3	4	5?					
	5?	4	3	2	3	4?	5?						
		5?	4?	3	4	5?							
			5?	4	5?								

			5?	4?									
		5?	4	3	4?	5?							
	5?	4	3	2	3	4	5?						
5?	4	3	2	1	2	3	4	5?					
4	3	2	1	★	1	2	3	4	5?				
5?	4	3	2	1	2	3	4	5?					
	5?	4	3	2	3	4?	5?						
		5?	4	3	4	5?							
			5?	4	5?								

			5?	4?									
		5?	4	3	4?	5?							
	5?	4	3	2	3	4	5?						
5?	4	3	2	1	2	3	4	5?					
4	3	2	1	★	1	2	3	4	5?				
5?	4	3	2	1	2	3	4	5?					
	5?	4	3	2	3	4	5?						
		5?	4	3	4	5?							
			5?	4	5?								

5? 4? 5?
5? 4 3 4 5?
5? 4 3 2 3 4 5?
5? 4 3 2 1 2 3 4 5?
4 3 2 1 1 2 3 4 5?
5? 4 3 2 1 2 3 4 5?
5? 4 3 2 3 4 5?
5? 4 3 4 5?
5? 4 5?

			5?	4	5?								
		5?	4	3	4	5?							
	5?	4	3	2	3	4	5?						
5?	4	3	2	1	2	3	4	5?					
4	3	2	1	★	1	2	3	4	5?				
5?	4	3	2	1	2	3	4	5?					
	5?	4	3	2	3	4	5?						
		5?	4	3	4	5?							
			5?	4	5?								

			5?	4	5?	6?							
		5?	4	3	4	5	6?						
	5?	4	3	2	3	4	5?						
5?	4	3	2	1	2	3	4	5?					
4	3	2	1	★	1	2	3	4	5?				
5?	4	3	2	1	2	3	4	5?					
	5?	4	3	2	3	4	5?						
		5?	4	3	4	5?							
			5?	4	5?								

			5?	4	5?	6?							
		5?	4	3	4	5	6?						
	5?	4	3	2	3	4	5?						
5?	4	3	2	1	2	3	4	5?					
4	3	2	1	★	1	2	3	4	5?				
5?	4	3	2	1	2	3	4	5?					
	5?	4	3	2	3	4	5	6?					
		5?	4	3	4	5?	6?						
			5?	4	5?								

			5?	4	5?	6?							
	6?	5?	4	3	4	5	6?						
6?	5	4	3	2	3	4	5?						
5?	4	3	2	1	2	3	4	5?					
4	3	2	1	★	1	2	3	4	5?				
5?	4	3	2	1	2	3	4	5?					
	5?	4	3	2	3	4	5	6?					
		5?	4	3	4	5?	6?						
			5?	4	5?								

			5?	4	5?	6?							
	6?	5?	4	3	4	5	6?						
6?	5	4	3	2	3	4	5?						
5?	4	3	2	1	2	3	4	5?					
4	3	2	1	★	1	2	3	4	5?				
5?	4	3	2	1	2	3	4	5?					
	5?	4	3	2	3	4	5	6?					
	6?	5	4	3	4	5?	6?						
		6?	5?	4	5?								

			5?	4	5?	6?							
	6?	5?	4	3	4	5	6?						
6?	5	4	3	2	3	4	5?						
5?	4	3	2	1	2	3	4	5?					
4	3	2	1	☆	1	2	3	4	☆				
5?	4	3	2	1	2	3	4	5?					
	5?	4	3	2	3	4	5	6?					
	6?	5	4	3	4	5?	6?						
		6?	5?	4	5?								

Why are we looking in this direction?

Why are we looking in this direction?
We haven't taken into account the cost from that node to the goal

Dijkstra Priority

priority(u) = distance(s, u)

Priority of the path that ends in u

Ideal Priority

priority(u) = distance(s, u) + futureCost(u, t)

Priority of the path that ends in u

*note we will revise this slightly

Future Cost?

```
function h(node,goal) {
  dRows = abs(node.row - goal.row);
  dCols = abs(node.col - goal.col);
  return dRows + dCols
}
```

1?
1?
1?
1?

1?
1?
1?
1?

1?
1?
1 +
4?
1?

1?
1?
1 +
4?
1?

1 +
6?
1?
1 +
4?
1?

1 +
6?
1?
1 +
4?
1?

1 +
6?
1?
1 +
4?
1 +
6?

1 +
6?
1?
1 +
4?
1 +
6?

1 +
6?
1 +
6?
1 +
4?
1 +
6?

1 +
6?
1 +
6?
1 +
4?
1 +
6?

1 +
6?
1 +
6?
1
1 +
6?

1 + 6?
2 + 5?
1 + 6?
1
2 + 3?
1 + 6?
2 + 5?

1 +
6?
2 +
5?
1 +
6?
1
2 +
3?
1 +
6?
2 +
5?

1 + 6?
2 + 5?
1 + 6?
1
2
1 + 6?
2 + 5?

1 + 6?
2 + 5?
3 + 4?
1 + 6?
1
2
3 + 2?
1 + 6?
2 + 5?
3 + 4?

1 + 6?
2 + 5?
3 + 4?
1 + 6?
1
2
3 + 2?
1 + 6?
2 + 5?
3 + 4?

1 + 6?
2 + 5?
3 + 4?
1 + 6?
1
2
3
1 + 6?
2 + 5?
3 + 4?

	1 + 6?	2 + 5?	3 + 4?	4 + 3?	
1 + 6?	☆	1	2	3	4 + 1?
	1 + 6?	2 + 5?	3 + 4?	4 + 3?	

☆

1 + 6?
2 + 5?
3 + 4?
4 + 3?
1 + 6?
1
2
3
4 + 1?
1 + 6?
2 + 5?
3 + 4?
4 + 3?

	1 + 6?	2 + 5?	3 + 4?	4 + 3?	
1 + 6?	☆	1	2	3	4
	1 + 6?	2 + 5?	3 + 4?	4 + 3?	

☆

1 + 6?	2 + 5?	3 + 4?	4 + 3?	5 + 2?	
1 + 6?	★	1	2	3	4	5 + 0? ★
1 + 6?	2 + 5?	3 + 4?	4 + 3?	5 + 2?	

1 + 6?
2 + 5?
3 + 4?
4 + 3?
5 + 2?
1 + 6?
1
2
3
4
5 + 6?
1 + 6?
2 + 5?
3 + 4?
4 + 3?
5 + 2?

	1 + 6?	2 + 5?	3 + 4?	4 + 3?	5 + 2?	
1 + 6?	☆	1	2	3	4	☆
	1 + 6?	2 + 5?	3 + 4?	4 + 3?	5 + 2?	

That was easy…

… a little too easy

Lets say we have unknown blocks

Ideal Priority

priority(u) = distance(s, u) + futureCost(u, t)

Priority of the path that ends in u

*note we will revise this slightly

A-Star Priority

priority(u) = distance(s, u) + heuristic(u, t)

Priority of the path that ends in u

Admissible Heuristic

Definition: An admissible heuristic always underestimates the true cost.

Thus: "even in the best case scenario, this path is still terrible…"

Admissible Heuristic

Just Relax

Columns apart
Rows apart

Manhattan Distance

```
function h(start,goal) {
  dRows = abs(start.row - goal.row);
  dCols = abs(start.col - goal.col);
  return dRows + dCols
}
```

Recall Dijkstra

```
Make a PriorityQueue todo-list of paths
Put a path with just the start in the todo-list
While the todo-list isn't empty
  1. Take a path out of the todo-list
  2. Call the last node in the path "currNode"
  3. If "currNode" is the goal, you are done.
  4. If you have seen currNode before, skip it.
  5. for all neighbors of currNode
       Make a newPath = path + neighbor
       Add the new path to the todo-list
       Priority = pathLength
```

A Star

```
Make a PriorityQueue todo-list of paths
Put a path with just the start in the todo-list
While the todo-list isn't empty
  1. Take a path out of the todo-list
  2. Call the last node in the path "currNode"
  3. If "currNode" is the goal, you are done.
  4. If you have seen currNode before, skip it.
  5. for all neighbors of currNode
       Make a newPath = path + neighbor
       Add the new path to the todo-list
       Priority = pathLength + h(neighbor, goal)
```

A Star

```
Make a PriorityQueue todo-list of paths
Put a path with just the start in the todo-list
While the todo-list isn't empty
  1. Take a path out of the todo-list
  2. Call                                  currNode"
  3. If "                                  are done.
  4. If you have seen currNode before, skip it.
  5. for all neighbors of currNode
       Make a newPath = path + neighbor
       Add the new path to the todo-list
       Priority = pathLength + h(neighbor, goal)
```

What is the priority of the start path?

A Star

```
Make a PriorityQueue todo-list of paths
Put a path with just the start in the todo-list
While the todo-list isn't empty
  1. Take a path out of the todo-list
  2. Call                                    urrNode"
  3. If "                                    e done.
  4. If y                                    skip it.
  5. for all neighbors of currNode
       Make a newPath = path + neighbor
       Add the new path to the todo-list
       Priority = pathLength + h(neighbor, goal)
```

What is the priority of the start path?

h(start, goal)

1 +
6?
1 +
6?
1 +
4?
1 +
6?

1 +
6?
1 +
6?
1
1 +
6?

1 + 6?
2 + 5?
1 + 6?
1
2 + 3?
1 + 6?
2 + 5?

1 + 6?
2 + 5?
1 + 6?
1
2
1 + 6?
2 + 5?

1 + 6?
2 + 5?
3 + 4?
1 + 6?
1
2
1 + 6?
2 + 5?
3 + 4?

1 + 6?
2 + 5?
3 + 4?
1
1
2
1 + 6?
2 + 5?
3 + 4?

2 + 7?
1 + 6?
2 + 5?
3 + 4?
2 + 7?
1
1
2
2 + 7?
1 + 6?
2 + 5?
3 + 4?

2 + 7?
1 + 6?
2 + 5?
3 + 4?
2 + 7?
1
1
2
2 + 7?
1 + 6?
2
3 + 4?

2 + 7?
1 + 6?
2 + 5?
3 + 4?
2 + 7?
1
1
2
2 + 7?
1 + 6?
2
3 + 4?
3 + 6?

2 + 7?
1 + 6?
2
3 + 4?
2 + 7?
1
1
2
2 + 7?
1 + 6?
2
3 + 4?
3 + 6?

3 + 6?
2 + 7?
1 + 6?
2
3 + 4?
2 + 7?
1
1
2
2 + 7?
1 + 6?
2
3 + 4?
3 + 6?

3 + 6?
2 + 7?
1
2
3 + 4?
2 + 7?
1
1
2
2 + 7?
1 + 6?
2
3 + 4?
3 + 6?

2 + 7?
3 + 6?
2 + 7?
1
2
3 + 4?
2 + 7?
1
1
2
2 + 7?
1 + 6?
2
3 + 4?
3 + 6?

2 + 7?
3 + 6?
2 + 7?
1
2
3 + 4?
2 + 7?
1
1
2
2 + 7?
1
2
3 + 4?
3 + 6?

2 + 7?
3 + 6?
2 + 7?
1
2
3 + 4?
2 + 7?
1
1
2
2 + 7?
1
2
3 + 4?
2 + 7?
3 + 6?

2 + 7?
3 + 6?
2 + 7?
1
2
3
2 + 7?
1
1
2
2 + 7?
1
2
3 + 4?
2 + 7?
3 + 6?

2 + 7?
3 + 6?
4 + 5?
2 + 7?
1
2
3
2 + 7?
1
1
2
2 + 7?
1
2
3 + 4?
2 + 7?
3 + 6?

2 + 7?
3 + 6?
4 + 5?
2 + 7?
1
2
3
2 + 7?
1
1
2
2 + 7?
1
2
3
2 + 7?
3 + 6?

2 + 7?
3 + 6?
4 + 5?
2 + 7?
1
2
3
2 + 7?
1
1
2
2 + 7?
1
2
3
2 + 7?
3 + 6?
4 + 5?

2 + 7?
3 + 6?
4 + 5?
2 + 7?
1
2
3
2 + 7?
1
1
2
2
1
2
3
2 + 7?
3 + 6?
4 + 5?

2 + 7?
3 + 6?
4 + 5?
2 + 7?
1
2
3
2 + 7?
1
1
2
3 + 8?
2
1
2
3
3 + 8?
2 + 7?
3 + 6?
4 + 5?

2
3 + 6?
4 + 5?
2 + 7?
1
2
3
2 + 7?
1
1
2
3 + 8?
2
1
2
3
3 + 8?
2 + 7?
3 + 6?
4 + 5?

3 + 8?
3 + 8?
2
3 + 6?
4 + 5?
2 + 7?
1
2
3
2 + 7?
1
1
2
3 + 8?
2
1
2
3
3 + 8?
2 + 7?
3 + 6?
4 + 5?

3 + 8?
3 + 8?
2
3 + 6?
4 + 5?
2 + 7?
1
2
3
2 + 7?
1
1
2
3 + 8?
2
1
2
3
3 + 8?
2 + 7?
3 + 6?
4

3 + 8?
3 + 8?
2
3 + 6?
4 + 5?
2 + 7?
1
2
3
2 + 7?
1
1
2
3 + 8?
2
1
2
3
3 + 8?
2 + 7?
3 + 6?
4
5 + 4?
5 + 6?

3 + 8?
3 + 8?
2
3 + 6?
4 + 5?
2 + 7?
1
2
3
2 + 7?
1
1
2
3 + 8?
2
1
2
3
3 + 8?
2
3 + 6?
4
5 + 4?
5 + 6?

3 + 8?
3 + 8?
2
3 + 6?
4 + 5?
2 + 7?
1
2
3
2 + 7?
1
1
2
3 + 8?
2
1
2
3
3 + 8?
2
3 + 6?
4
5 + 4?
3 + 8?
5 + 6?

3 + 8?
3 + 8?
2
3 + 6?
4 + 5?
2 + 7?
1
2
3
2 + 7?
1
1
2
3 + 8?
2
1
2
3
3 + 8?
2
3
4
5 + 4?
3 + 8?
5 + 6?

3 + 8?
3 + 8?
2
3 + 6?
4 + 5?
2 + 7?
1
2
3
2 + 7?
1
1
2
3 + 8?
2
1
2
3
3 + 8?
2
3
4
5 + 4?
3 + 8?
4 + 7?
5 + 6?

3 + 8?
3 + 8?
2
3 + 6?
4 + 5?
2 + 7?
1
2
3
2
1
1
2
3 + 8?
2
1
2
3
3 + 8?
2
3
4
5 + 4?
3 + 8?
4 + 7?
5 + 6?

3 + 8?
3 + 8?
2
3 + 6?
4 + 5?
3 + 8?
2 + 7?
1
2
3
3 + 8?
2
1
1
2
3 + 8?
2
1
2
3
3 + 8?
2
3
4
5 + 4?
3 + 8?
4 + 7?
5 + 6?

3 + 8?
3 + 8?
2
3 + 6?
4 + 5?
3 + 8?
2
1
2
3
3 + 8?
2
1
1
2
3 + 8?
2
1
2
3
3 + 8?
2
3
4
5 + 4?
3 + 8?
4 + 7?
5 + 6?

3 + 8?
3 + 8?
2
3
4 + 5?
3 + 8?
2
1
2
3
3 + 8?
2
1
1
2
3 + 8?
2
1
2
3
3 + 8?
2
3
4
5 + 4?
3 + 8?
4 + 7?
5 + 6?

3 + 8?
4 + 7?
3 + 8?
2
3
4 + 5?
3 + 8?
2
1
2
3
3 + 8?
2
1
1
2
3 + 8?
2
1
2
3
3 + 8?
2
3
4
5 + 4?
3 + 8?
4 + 7?
5 + 6?

3 + 8?
4 + 7?
3 + 8?
2
3
4
3 + 8?
2
1
2
3
3 + 8?
2
1
1
2
3 + 8?
2
1
2
3
3 + 8?
2
3
4
5 + 4?
3 + 8?
4 + 7?
5 + 6?

3 + 8?
4 + 7?
5 + 6?
3 + 8?
2
3
4
5 + 4?
3 + 8?
2
1
2
3
3 + 8?
2
1
1
2
3 + 8?
2
1
2
3
3 + 8?
2
3
4
5 + 4?
3 + 8?
4 + 7?
5 + 6?

3 + 8?
4 + 7?
5 + 6?
3 + 8?
2
3
4
5 + 4?
3 + 8?
2
1
2
3
3 + 8?
2
1
1
2
3 + 8?
2
1
2
3
3 + 8?
2
3
4
5
3 + 8?
4 + 7?
5 + 6?

3 + 8?
4 + 7?
5 + 6?
3 + 8?
2
3
4
5 + 4?
3 + 8?
2
1
2
3
3 + 8?
2
1
1
2
3 + 8?
2
1
2
3
3 + 8?
2
3
4
5
6 + 3?
3 + 8?
4 + 7?
5 + 6?
6 + 5?

3 + 8?
4 + 7?
5 + 6?
3 + 8?
2
3
4
5 + 4?
3 + 8?
2
1
2
3
3 + 8?
2
1
1
2
3 + 8?
2
1
2
3
3 + 8?
2
3
4
5
6
3 + 8?
4 + 7?
5 + 6?
6 + 5?

3 + 8?
4 + 7?
5 + 6?
3 + 8?
2
3
4
5 + 4?
3 + 8?
2
1
2
3
3 + 8?
2
1
1
2
3 + 8?
2
1
2
3
7 + 2?
3 + 8?
2
3
4
5
6
7 + 2?
3 + 8?
4 + 7?
5 + 6?
6 + 5?
7 + 4?

3 + 8?
4 + 7?
5 + 6?
3 + 8?
2
3
4
5
3 + 8?
2
1
2
3
3 + 8?
2
1
1
2
3 + 8?
2
1
2
3
7 + 2?
3 + 8?
2
3
4
5
6
7 + 2?
3 + 8?
4 + 7?
5 + 6?
6 + 5?
7 + 4?

3 + 8?
4 + 7?
5 + 6?
6 + 5?
3 + 8?
2
3
4
5
6 + 3?
3 + 8?
2
1
2
3
3 + 8?
2
1
1
2
3 + 8?
2
1
2
3
7 + 2?
3 + 8?
2
3
4
5
6
7 + 2?
3 + 8?
4 + 7?
5 + 6?
6 + 5?
7 + 4?

				3 + 8?	4 + 7?	5 + 6?	6 + 5?						
			3 + 8?	2	3	4	5	6 + 3?					
		3 + 8?	2	1	2	3							
	3 + 8?	2	1	☆	1	2			☆				
		3 + 8?	2	1	2	3		7					
			3 + 8?	2	3	4	5	6	7 + 2?				
				3 + 8?	4 + 7?	5 + 6?	6 + 5?	7 + 4?					

				3 + 8?	4 + 7?	5 + 6?	6 + 5?						
			3 + 8?	2	3	4	5	6 + 3?					
		3 + 8?	2	1	2	3							
	3 + 8?	2	1	☆	1	2		8 + 1?	☆				
		3 + 8?	2	1	2	3		7	8 + 1?				
			3 + 8?	2	3	4	5	6	7 + 2?				
				3 + 8?	4 + 7?	5 + 6?	6 + 5?	7 + 4?					

				3 + 8?	4 + 7?	5 + 6?	6 + 5?						
			3 + 8?	2	3	4	5	6 + 3?					
		3 + 8?	2	1	2	3							
	3 + 8?	2	1	☆	1	2		8 + 1?	☆				
		3 + 8?	2	1	2	3		7	8 + 1?				
			3 + 8?	2	3	4	5	6	7				
				3 + 8?	4 + 7?	5 + 6?	6 + 5?	7 + 4?					

				3 + 8?	4 + 7?	5 + 6?	6 + 5?						
			3 + 8?	2	3	4	5	6 + 3?					
		3 + 8?	2	1	2	3							
	3 + 8?	2	1	☆	1	2		8 + 1?	☆				
		3 + 8?	2	1	2	3		7	8 + 1?				
			3 + 8?	2	3	4	5	6	7	8 + 3?			
				3 + 8?	4 + 7?	5 + 6?	6 + 5?	7 + 4?	8 + 3?				

				3 + 8?	4 + 7?	5 + 6?	6 + 5?						
			3 + 8?	2	3	4	5	6 + 3?					
		3 + 8?	2	1	2	3							
	3 + 8?	2	1	☆	1	2		8	☆				
		3 + 8?	2	1	2	3		7	8 + 1?				
			3 + 8?	2	3	4	5	6	7	8 + 3?			
				3 + 8?	4 + 7?	5 + 6?	6 + 5?	7 + 4?	8 + 3?				

				3 + 8?	4 + 7?	5 + 6?	6 + 5?						
			3 + 8?	2	3	4	5	6 + 3?					
		3 + 8?	2	1	2	3		9 + 2?					
	3 + 8?	2	1	☆	1	2		8	9 + 0?				
		3 + 8?	2	1	2	3		7	8 + 1?				
			3 + 8?	2	3	4	5	6	7	8 + 3?			
				3 + 8?	4 + 7?	5 + 6?	6 + 5?	7 + 4?	8 + 3?				

				3 + 8?	4 + 7?	5 + 6?	6 + 5?						
			3 + 8?	2	3	4	5	6					
		3 + 8?	2	1	2	3		9 + 2?					
	3 + 8?	2	1	☆	1	2		8	9 + 0?				
		3 + 8?	2	1	2	3		7	8 + 1?				
			3 + 8?	2	3	4	5	6	7	8 + 3?			
				3 + 8?	4 + 7?	5 + 6?	6 + 5?	7 + 4?	8 + 3?				

				3 + 8?	4 + 7?	5 + 6?	6 + 5?	7 + 4?					
			3 + 8?	2	3	4	5	6	7 + 2?				
		3 + 8?	2	1	2	3		7 + 2?					
	3 + 8?	2	1	★	1	2		8	9 + 0?				
		3 + 8?	2	1	2	3		7	8 + 1?				
			3 + 8?	2	3	4	5	6	7	8 + 3?			
				3 + 8?	4 + 7?	5 + 6?	6 + 5?	7 + 4?	8 + 3?				

				3 + 8?	4 + 7?	5 + 6?	6 + 5?	7 + 4?					
			3 + 8?	2	3	4	5	6	7 + 2?				
		3 + 8?	2	1	2	3		7 + 2?					
	3 + 8?	2	1	☆	1	2		8	☆				
		3 + 8?	2	1	2	3		7	8 + 1?				
			3 + 8?	2	3	4	5	6	7	8 + 3?			
				3 + 8?	4 + 7?	5 + 6?	6 + 5?	7 + 4?	8 + 3?				

What Dijkstra would have Searched

8	7	6	5	4	5	6	7	8	9?				
7	6	5	4	3	4	5	6	7	8	9?			
6	5	4	3	2	3	4	5	6	7	8	9?		
5	4	3	2	1	2	3		7	8	9?			
4	3	2	1	☆	1	2		8	☆				
5	4	3	2	1	2	3		7	8	9?			
6	5	4	3	2	3	4	5	6	7	8	9?		
7	6	5	4	3	4	5	6	7	8	9?			
8	7	6	5	4	5	6	7	8	9?				

Why underestimate?

You only ignore paths that in the best case ar
worse than your current path.

Using our heuristic, the path from start to goal that goes through this node is at least cost 11

Imagine we Overestimate

Imagine we Overestimate

See It Live

Path hops: 16

Path cost: 16

$h(start,goal) = 0?$

Admissible!

A Star

```
Make a PriorityQueue todo-list of paths
Put a path with just the start in the todo-list
While the todo-list isn't empty
  1. Take a path out of the todo-list
  2. Call the last node in the path "currNode"
  3. If "currNode" is the goal, you are done.
  4. If you have seen currNode before, skip it.
  5. for all neighbors of currNode
       Make a newPath = path + neighbor
       Add the new path to the todo-list
       Priority = pathLength + h(neighbor, goal)
```

A Star

```
Make a PriorityQueue todo-list of paths
Put a path with just the start in the todo-list
While the todo-list isn't empty
  1. Take a path out of the todo-list
  2. Call the last node in the path "currNode"
  3. If "currNode" is the goal, you are done.
  4. If you have seen currNode before, skip it.
  5. for all neighbors of currNode
       Make a newPath = path + neighbor
       Add the new path to the todo-list
       Priority = pathLength + 0
```

Hi Dijkstra

```
Make a PriorityQueue todo-list of paths
Put a path with just the start in the todo-list
While the todo-list isn't empty
  1. Take a path out of the todo-list
  2. Call the last node in the path "currNode"
  3. If "currNode" is the goal, you are done.
  4. If you have seen currNode before, skip it.
  5. for all neighbors of currNode
       Make a newPath = path + neighbor
       Add the new path to the todo-list
       Priority = pathLength + 0
```

Want the largest admissible heuristic possible

Today's Route

Today's Route

How Does Google Maps Work?

You Will Find Out!

Road Map Node

Road Map Node

Road Map Edge

Road Map Edge

Road Map Edge Cost

Road Map Path Cost

Could Google Just Precompute?

How many nodes in google maps graph?

~ 75 million

n^2

6×10^{15}

1 petasecond = 31.7 million years

Can you think of a heuristic?

Road Map Heuristic

Road Map Heuristic

Must Underestimate This Time

Direct Highway

$$\text{Heuristic} = \frac{\text{Distance on surface of earth}}{\text{Speed on fastest highway}}$$

Distance to Landmarks

Landmark Heuristic

Distance < abs(A – B)

Best of All Heuristics

$$h = \max(h_1, h_2, ..., h_n)$$

Today's Goal

1. Dijkstra's Search
2. A-Star Search