Search - Part II

AIMA Sections 3.5 - 3.7

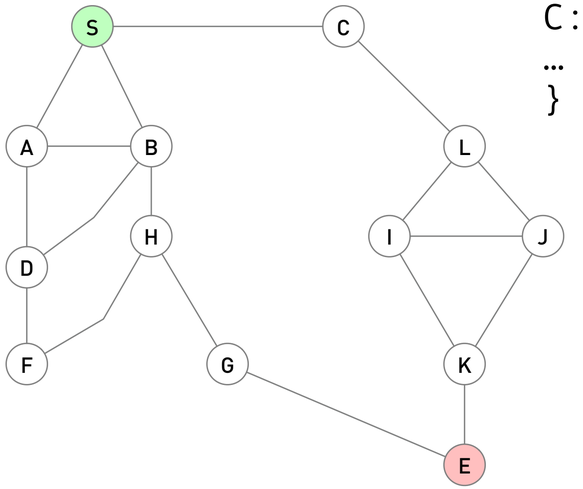
Artificial Intelligence - Manuel Pita

Recap: Uninformed Search

```
g #search graph as adjacency list
init #initial node from where we begin searching
goal #destination node we want to find
q = [(init, [init])] #keep next node to visit, and a list of the open path it is on
```

```
while q: #this is valid in python: while q is not empty
    h = Head(q) #remember this is always the first element
    r = Rest(q) #remember this is always a list of remaining elements
    if h[0] == goal:
       exit 😅
    else:
       e = Expand*(h) #get nodes directly reachable from h
       q = Combine(e,r) #put all past non visited nodes and future
```

```
g = {
S: {A,B,C},
A: {B,D,S},
B: {A,D,H,S},
C: {L},
…
}
```



q = e + r : DEPTH FIRST

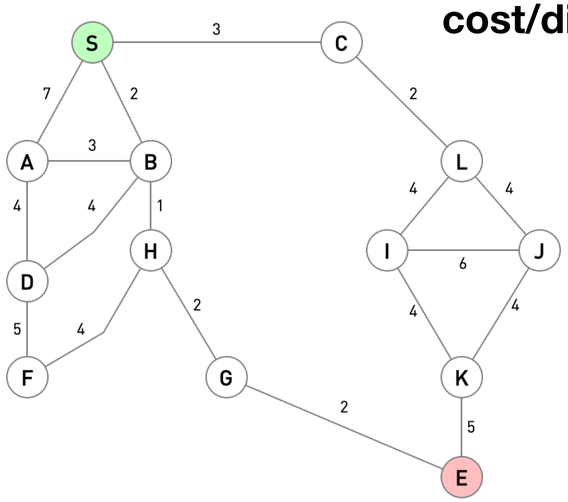
q = r + e : BREADTH FIRST

We do not have a measure to tell us what expanded node is best to follow, this is the sense in which we have no information. And this is why we systematise search so that we can explore all possible paths iteratively.

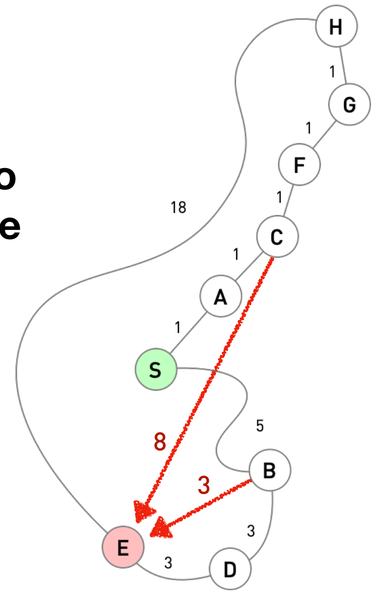
There is a very important interplay between *r* and *e* in keeping *q* updated with the paths the algorithm has not explored yet.

Informed Search

Information about the real cost/distance between pairs of nodes



Information about the estimates of how far we are to the goal from every other node



Graphs and Mr. Dijkstra

- **Edsger Dijkstra was born on May 11th, 1930. Died August 6th 2002**
- **Computer Scientist from the Netherlands**
- **Wrote many essays about computer programming and Graph Theory**
- **Received the Turing Prize in 1972, perhaps the equivalent to the Oscar in Computer Science**

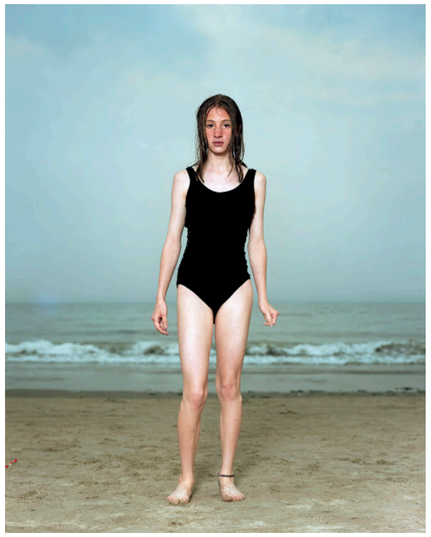
> “
> Elegance is not a dispensable luxury
> but a factor that decides between success and failure.
>
> **Edsger Wybe Dijkstra**

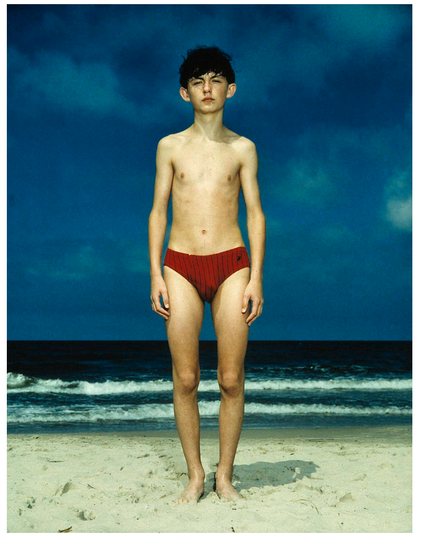
Side story, another Dijkstra

- **Rineke Dijkstra was born on June 2th, 1959**
- **Fine Art Photographer from the Netherlands**
- **Has taken many portraits of people right after critical vulnerability moments**
- **Received the Honorary Fellowship of the Royal Photographic Society**

> “For me it is essential to understand that everyone is alone. Not in the sense of loneliness, but rather in the sense that no one can completely understand someone else.
>
> **Rineke Dijkstra**

Side story, another Dijkstra

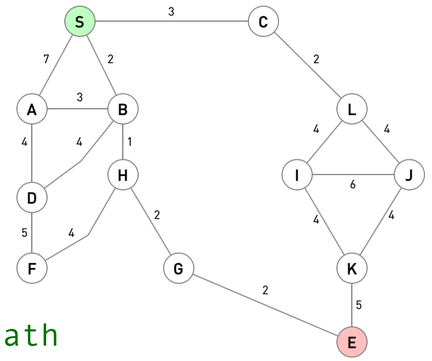




Dijkstra Algorithm

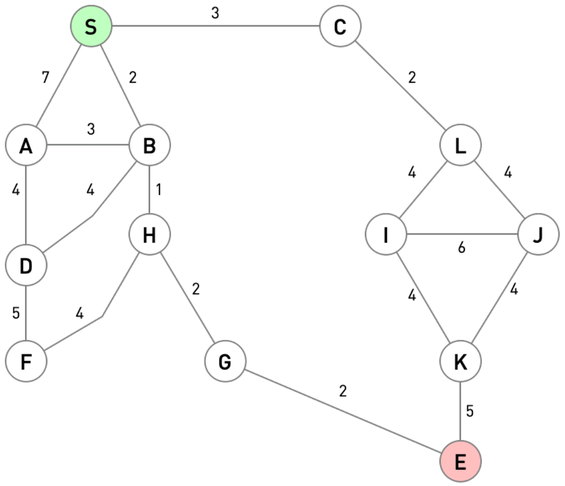
```
[1] g # search space, adjacency list but now including cost between nodes
[2] init # initial state
[3] goal # goal state
[4] q # list of tuples with three elements, terminal node, path to get to it and total cost from init
[5] # initialise all tuples for every node, with empty path and cost set to infinite,
    # except for init that starts with cost zero and [init] as its currently shortest path
    done = [] list of processed nodes # processed nodes end up here

[6] while q:
[7]    h = Head(q)
[8]    r = Rest(q)
[9]    if h[0] == goal:
[10]        exit # here we can exit but also alternatively just print path and continue
[11]   else:
[12]       e = Expand(h)
[13]   for node in e:
[14]       if accumulated cost from init to node less than current cost in tuple for node:
[15]          Update tuple for node with new cost and new path
[16]   done.append(h)
[17]   q = sort(e+r) #we combine all elements of the queue and sort from shortest to longest path
```




```
g = {
S: {(A,7),(B,2),(C,3)},
A: {(B,3),(D,4),(S,7)},
B: {(A,2),(D,4),(H,1),(S,2)},
C: {(L,3)},
…
}
```

Dijkstra Algorithm



Initialise

S	0	[S]
A	∞	[]
B	∞	[]
C	∞	[]
D	∞	[]
E	∞	[]
F	∞	[]

G	∞	[]
H	∞	[]
I	∞	[]
J	∞	[]
K	∞	[]
L	∞	[]

Iteration 1

S	0	[S]
B	2	[S,B]
C	3	[S,C]
A	7	[S,A]
D	∞	[]
E	∞	[]
F	∞	[]

G	∞	[]
H	∞	[]
I	∞	[]
J	∞	[]
K	∞	[]
L	∞	[]

Iteration 2

B	2	[S,B]
C	3	[S,C]
H	3	[S,B,H]
A	5	[S,B,A]
D	6	[S,B,D]
E	∞	[]
F	∞	[]
S	0	DONE

G	∞	[]
I	∞	[]
J	∞	[]
K	∞	[]
L	∞	[]

Iteration 3

C	3	[S,C]
H	3	[S,B,H]
A	5	[S,B,A]
L	5	[S,C,L]
D	6	[S,B,D]
E	∞	[]
F	∞	[]
B	2	DONE
S	0	DONE

G	∞	[]
I	∞	[]
J	∞	[]
K	∞	[]

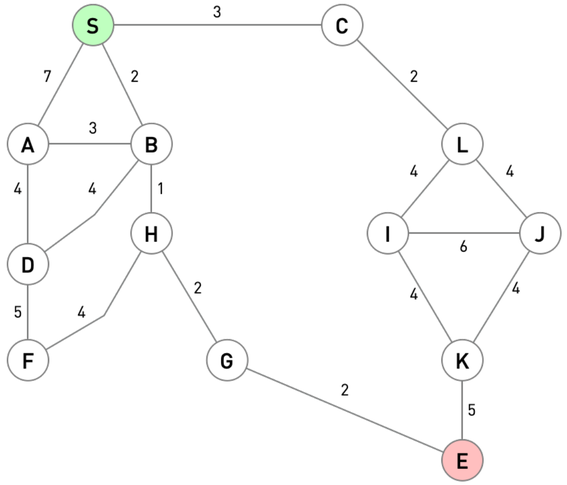
Iteration 4

H	3	[S,B,H]
A	5	[S,B,A]
L	5	[S,C,L]
G	5	[S,B,H,G]
D	6	[S,B,D]
F	7	[S,B,H,F]
E	∞	[]
C	3	DONE
B	2	DONE
S	0	DONE

I	∞	[]
J	∞	[]
K	∞	[]

Each node has its own card, with a cost to infinite, and empty shortest path, except for init node (S) which has zero cost (we are there) and itself as its path

Dijkstra Algorithm



Iteration 4

H	3	[S,B,H]			
A	5	[S,B,A]			
L	5	[S,C,L]			
G	5	[S,B,H,G]			
D	6	[S,B,D]	I	∞	[]
F	7	[S,B,H,F]	J	∞	[]
E	∞	[]	K	∞	[]

C	3	DONE	
B	2	DONE	
S	0	DONE	

Iteration 5

A	5	[S,B,A]			
L	5	[S,C,L]			
G	5	[S,B,H,G]			
D	6	[S,B,D]			
F	7	[S,B,H,F]			
E	∞	[]	J	∞	[]
I	∞	[]	K	∞	[]

H	3	DONE	
C	3	DONE	
B	2	DONE	
S	0	DONE	

Iteration 6

L	5	[S,C,L]			
G	5	[S,B,H,G]			
D	6	[S,B,D]			
F	7	[S,B,H,F]			
I	9	[S,C,L,I]			
J	9	[S,C,L,J]			
E	∞	[]	K	∞	[]

A	5	DONE	
H	3	DONE	
C	3	DONE	
B	2	DONE	
S	0	DONE	

Iteration 7

G	5	[S,B,H,G]			
D	6	[S,B,D]			
F	7	[S,B,H,F]			
E	7	[S,B,H,G,E]			
I	9	[S,C,L,I]			
J	9	[S,C,L,J]			
K	∞				

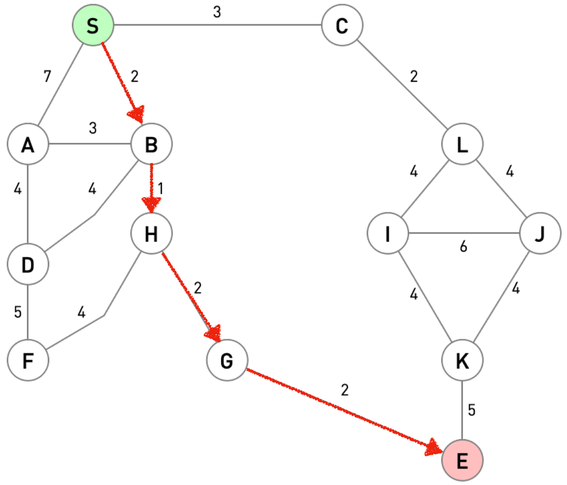
L	5	DONE	
A	5	DONE	
H	3	DONE	
C	3	DONE	
S	0	DONE	
B	2	DONE	

Iteration 8

D	6	[S,B,D]			
F	7	[S,B,H,F]			
E	7	[S,B,H,G,E]			
I	9	[S,C,L,I]			
J	9	[S,C,L,J]			
K	∞	[]			

G	5	DONE			
L	5	DONE	S	0	DONE
A	5	DONE	B	2	DONE
H	3	DONE	C	3	DONE

Dijkstra Algorithm



[S,B,H,G,E]

Iteration 8

D	6	[S,B,D]
F	7	[S,B,H,F]
E	7	[S,B,H,G,E]
I	9	[S,C,L,I]
J	9	[S,C,L,J]
K	∞	[]

Iteration 9

F	7	[S,B,H,F]
E	7	[S,B,H,G,E]
I	9	[S,C,L,I]
J	9	[S,C,L,J]
K	∞	[]

Iteration 10

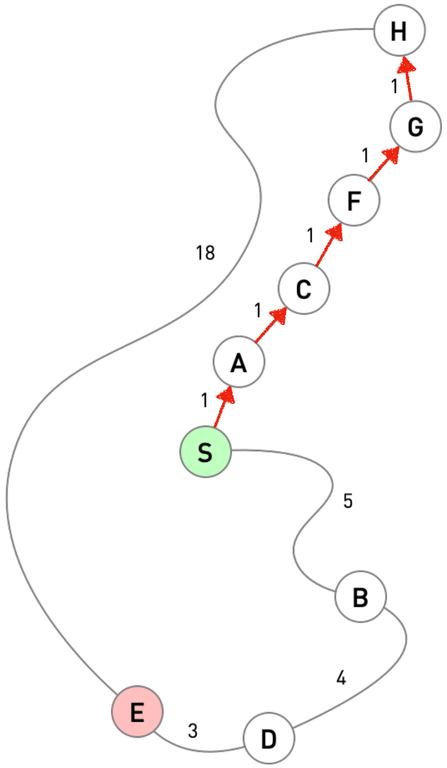
E	7	[S,B,H,G,E]
I	9	[S,C,L,I]
J	9	[S,C,L,J]
K	∞	[]

G	5	DONE					
L	5	DONE	S	0	DONE		
A	5	DONE	B	2	DONE		
H	3	DONE	C	3	DONE		

D	6	DONE	S	0	DONE		
G	5	DONE	B	2	DONE		
L	5	DONE	C	3	DONE		
A	5	DONE	H	3	DONE		

F	7	DONE					
D	6	DONE	S	0	DONE		
G	5	DONE	B	2	DONE		
L	5	DONE	C	3	DONE		
A	5	DONE	H	3	DONE		

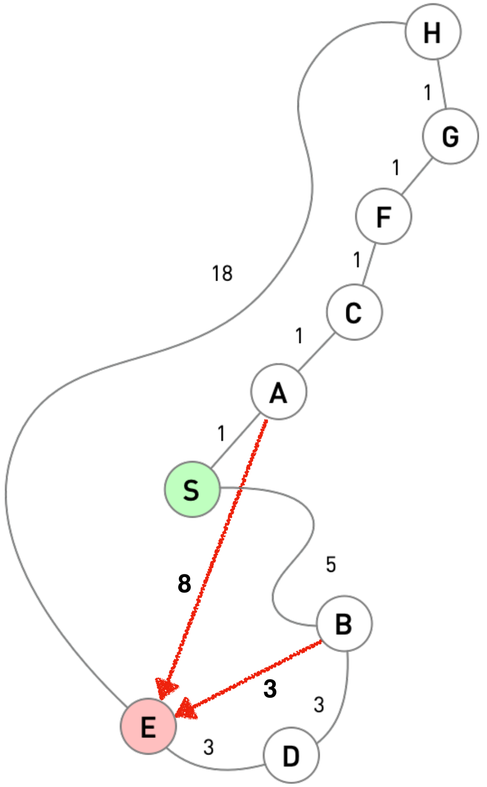
Dijkstra Algorithm: Limitations



- What path is going to be favoured by Dijkstra?
- It goes all the way to H before it tries B. Why?
- Because it follows a greedy approach based on the current shortest path
- The algorithm does not see that ‘shortest’ path is getting more distant to the goal
- How can we improve it?
- We need to add more information, about how far are we from the goal
- A small modification of the Dijkstra algorithm solves this problem and results in the algorithm known as A* (A-STAR)

Heuristics

- A heuristic is an estimate, but not any hunch, it is our very best educated guess
- A Heuristic H* gives us an estimate of how much effort, cost or distance to reach the goal
- In the domain of distances we always and safely say that a distance between two nodes will be — at least — its straight line distance. Never less.
- In a grid system, like New York City you can also use the so-called Manhattan distance. If you don't know what this, go find out what it is: we use it in video games sometimes.

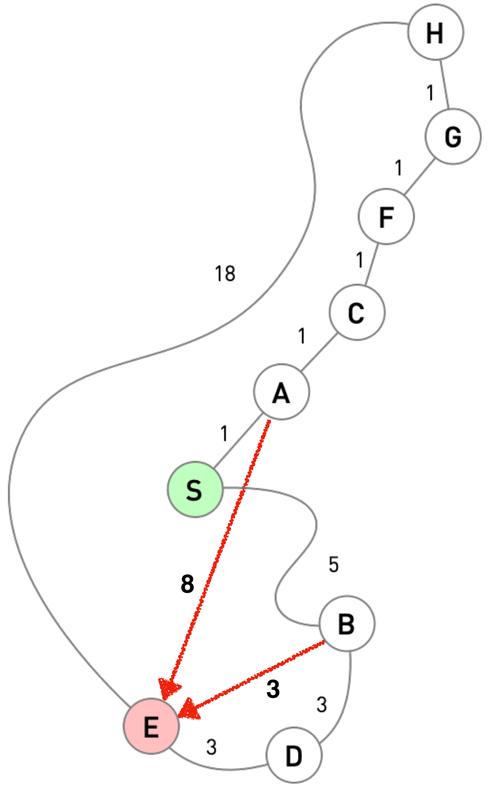


Node	Heuristic to Goal (E)
S	6
A	8
B	3
C	12
D	14
E	16
F	18
G	20
H	22

H* is always a relationship between every node and the goal state node

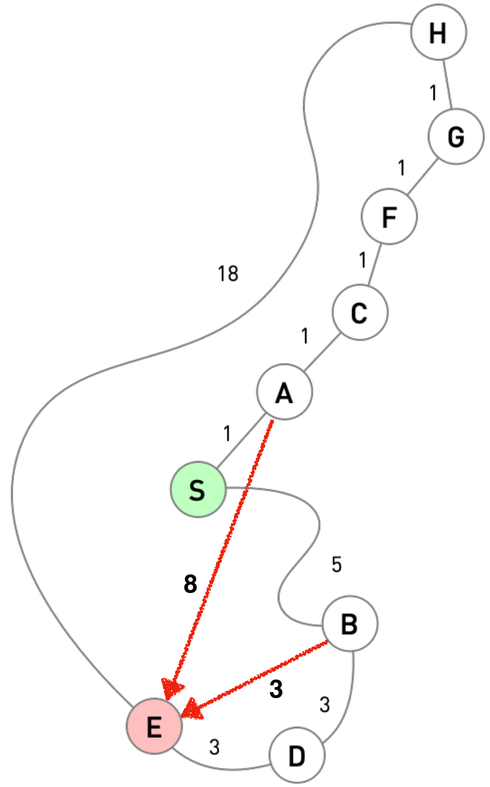
Heuristics: Admissibility and Dominance

- An admissible heuristic never overestimates the real cost/distance between two nodes
- In other words, the cost it estimates to reach the goal from a node X is never higher than the smallest possible cost from X to the goal
- Given two admissible heuristics H1* and H2* we choose the one that on average has higher estimates
- In other words given two admissible heuristics we pick the dominant one being the one that is closest to the real cost

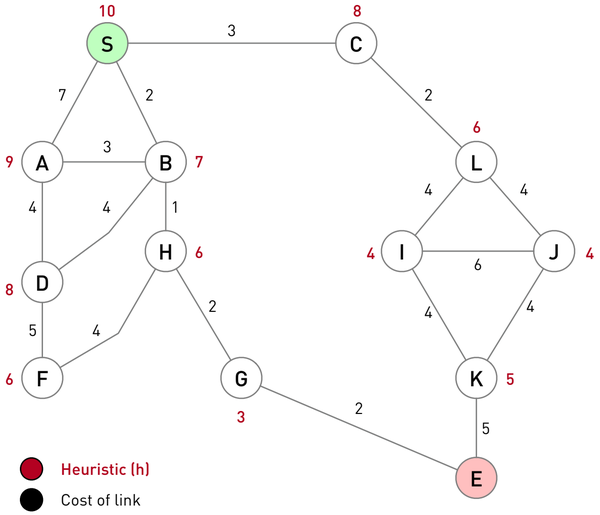


Admissibility and Dominance, What For?

- **Having an admissible heuristic guarantees that A* will always find the shortest path**
- **Having a dominant heuristic means that A* will take less iterations to find the shortest path**



A* Example



This is the head

Initialisation

S	0	10	10
[S]			

A	∞	0	∞
[]			

G	∞	0	∞
[]			

B	∞	0	∞
[]			

H	∞	0	∞
[]			

C	∞	0	∞
[]			

I	∞	0	∞
[]			

D	∞	0	∞
[]			

J	∞	0	∞
[]			

E	∞	0	∞
[]			

K	∞	0	∞
[]			

F	∞	0	∞
[]			

L	∞	0	∞
[]			

Iteration 1

S	0	10	10
[S]			

B	2	7	9
[S,B]			

G	∞	0	∞
[]			

C	3	8	11
[S,C]			

H	∞	0	∞
[]			

A	7	9	16
[S,A]			

I	∞	0	∞
[]			

D	∞	0	∞
[]			

J	∞	0	∞
[]			

E	∞	0	∞
[]			

K	∞	0	∞
[]			

F	∞	0	∞
[]			

L	∞	0	∞
[]			

Iteration 2

B	2	7	9
[S,B]			

H	3	6	9
[S,B,H]			

G	∞	0	∞
[]			

C	3	8	11
[S,C]			

D	6	8	14
[S,B,D]			

L	∞	0	∞
[]			

A	5	9	14
[S,B,A]			

I	∞	0	∞
[]			

E	∞	0	∞
[]			

J	∞	0	∞
[]			

F	∞	0	∞
[]			

K	∞	0	∞
[]			

Iteration 3

H	3	6	9
[S,B,H]			

G	5	3	8
[S,B,H,G]			

C	3	8	11
[S,C]			

D	6	8	14
[S,B,D]			

L	∞	0	∞
[]			

A	5	9	14
[S,B,A]			

I	∞	0	∞
[]			

F	7	6	14
[S,B,H,F]			

J	∞	0	∞
[]			

K	∞	0	∞
[]			

E	∞	0	∞
[]			

Iteration 4

G	5	3	8
[S,B,H,G]			

E	7	3	10
[S,B,H,G,E]			

C	3	8	11
[S,C]			

D	6	8	14
[S,B,D]			

A	5	9	14
[S,B,A]			

L	∞	0	∞
[]			

F	7	6	14
[S,B,H,F]			

I	∞	0	∞
[]			

K	∞	0	∞
[]			

J	∞	0	∞
[]			

Node | Cost so far (f) | Heuristic to goal (h) | Combined cost g = f+h

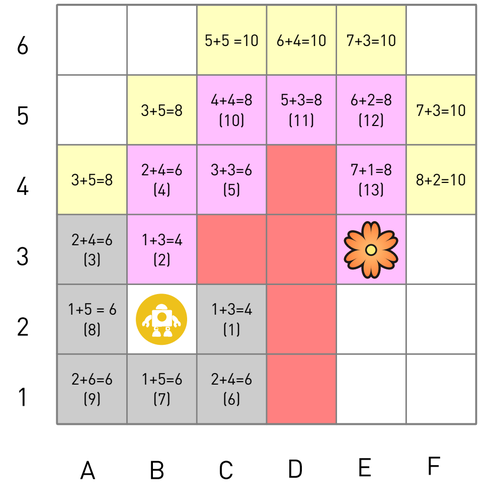
S	0	10	10
[S]			

Open path to node

During iteration 5, Node E is head of the Q
and we find the shortest path:
S,B,H,G,E

A Different Example

- **This is a grid world, with obstacles**
- **Robot starts at B2, must get flower at E3**
- **Robot cannot walk into obstacles (like D2)**
- **We will use the Manhattan distance as heuristic**
- **Robot can move straight to N, S, E, W (not diagonally)**



We can now expand any of the two paths with total value g=4

There are various paths with g=6. Prefer those with smallest heuristic values