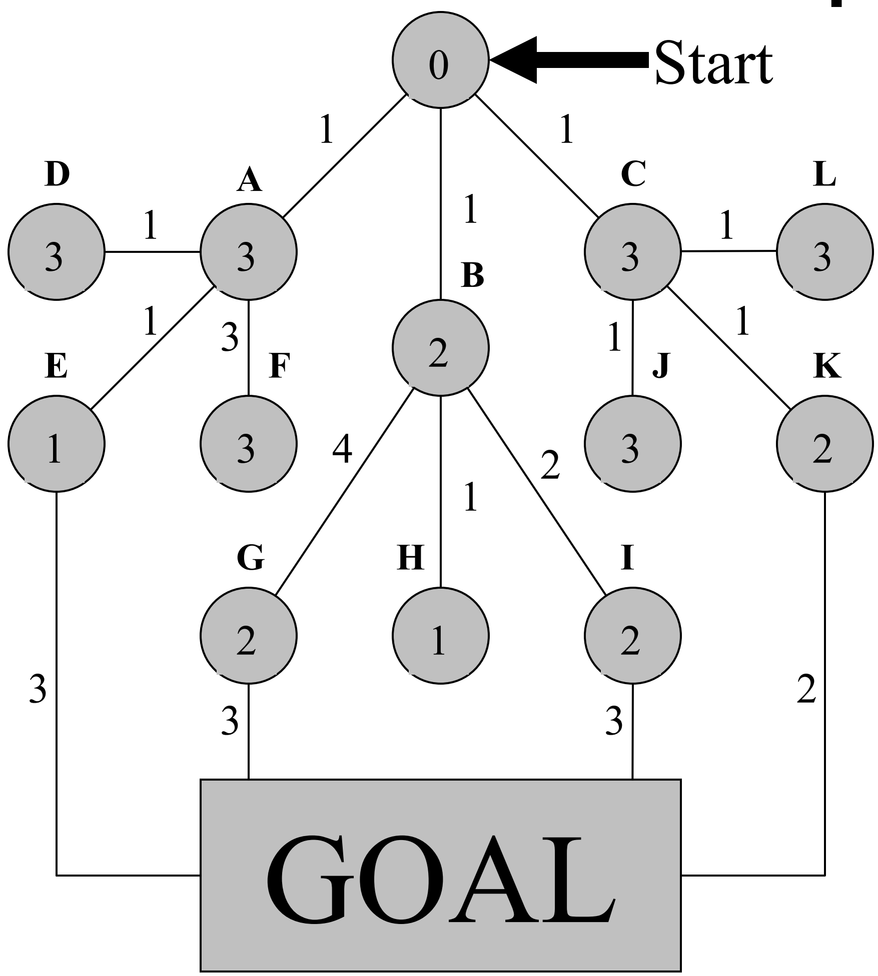
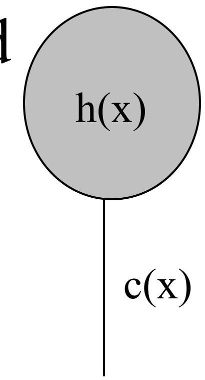
Example (1/5)



Legend



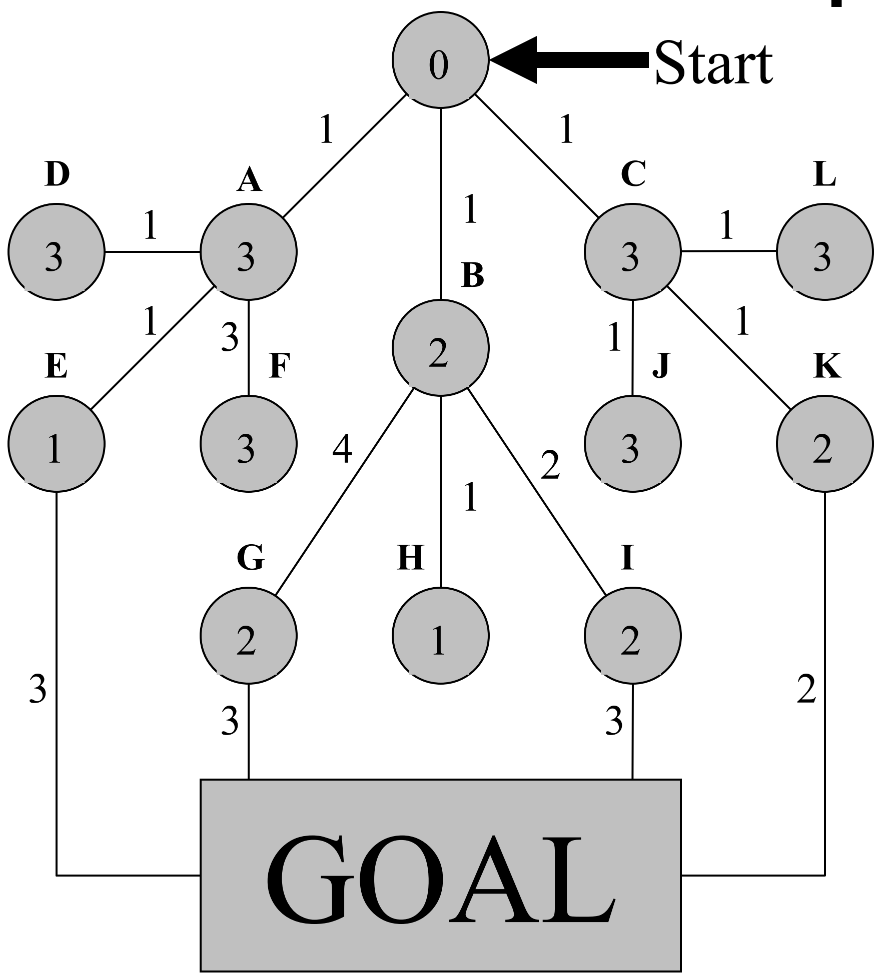
Priority = g(x) + h(x)

Note:

g(x) = sum of all previous arc costs, c(x), from start to x

Example: c(H) = 2

Example (2/5)



First expand the start node

B(3)
A(4)
C(4)

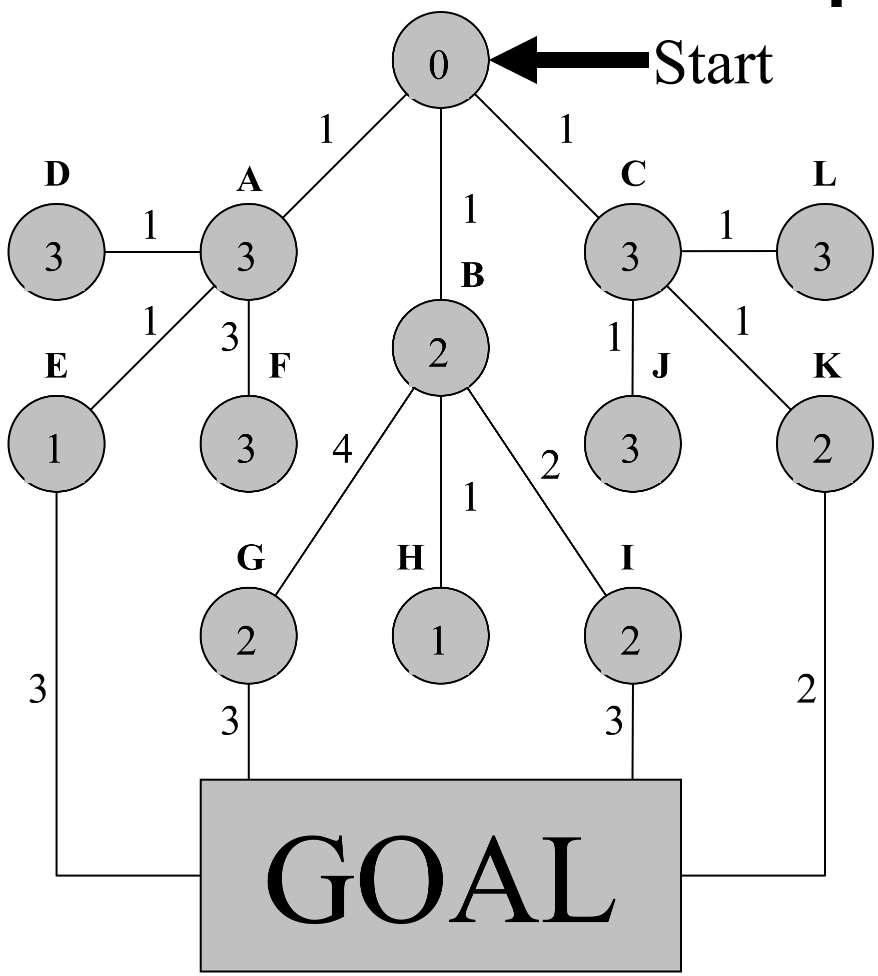
If goal not found, expand the first node in the priority queue (in this case, B)

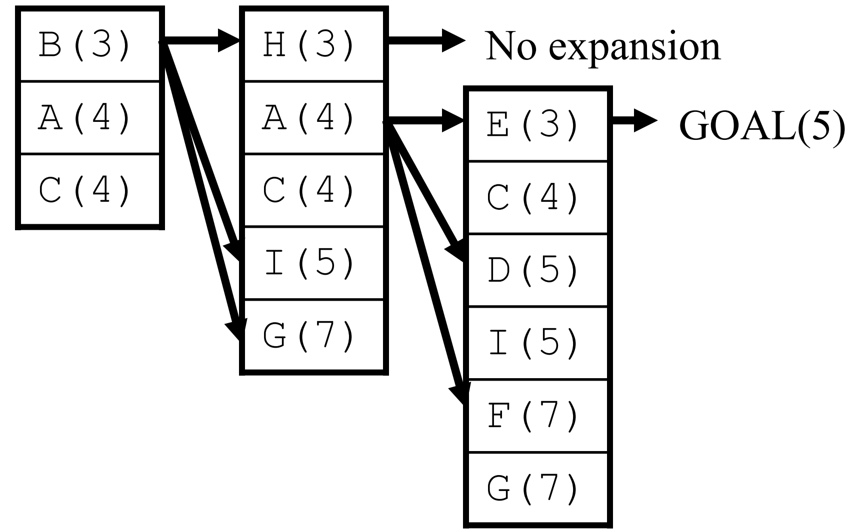
H(3)
A(4)
C(4)
I(5)
G(7)

Insert the newly expanded nodes into the priority queue and continue until the goal is found, or the priority queue is empty (in which case no path exists)

Note: for each expanded node, you also need a pointer to its respective parent. For example, nodes A, B and C point to Start

Example (3/5)

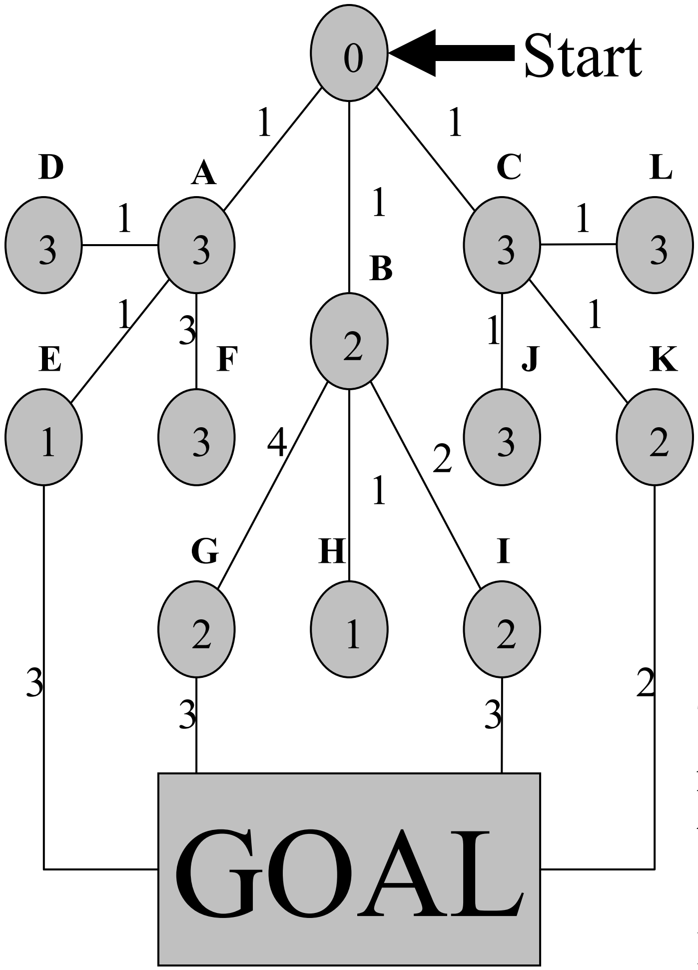


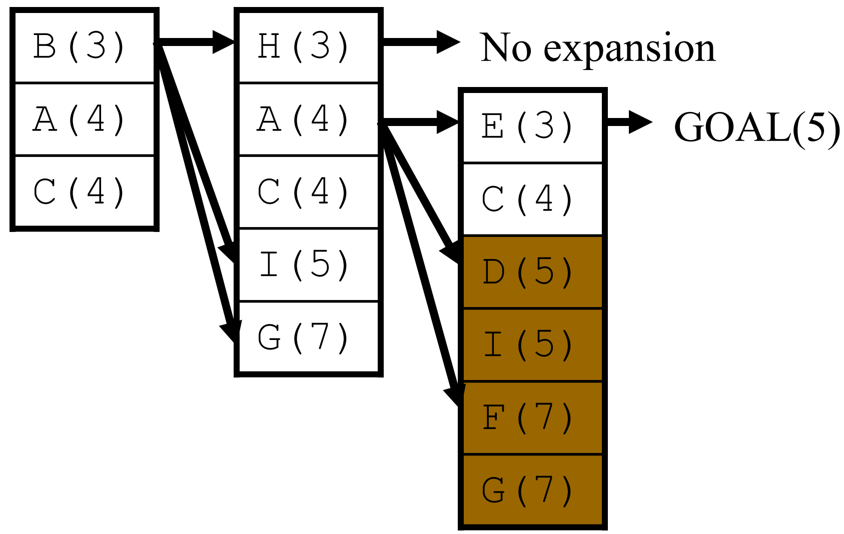


We've found a path to the goal:
Start => A => E => Goal
(*from the pointers*)

Are we done?

Example (4/5)



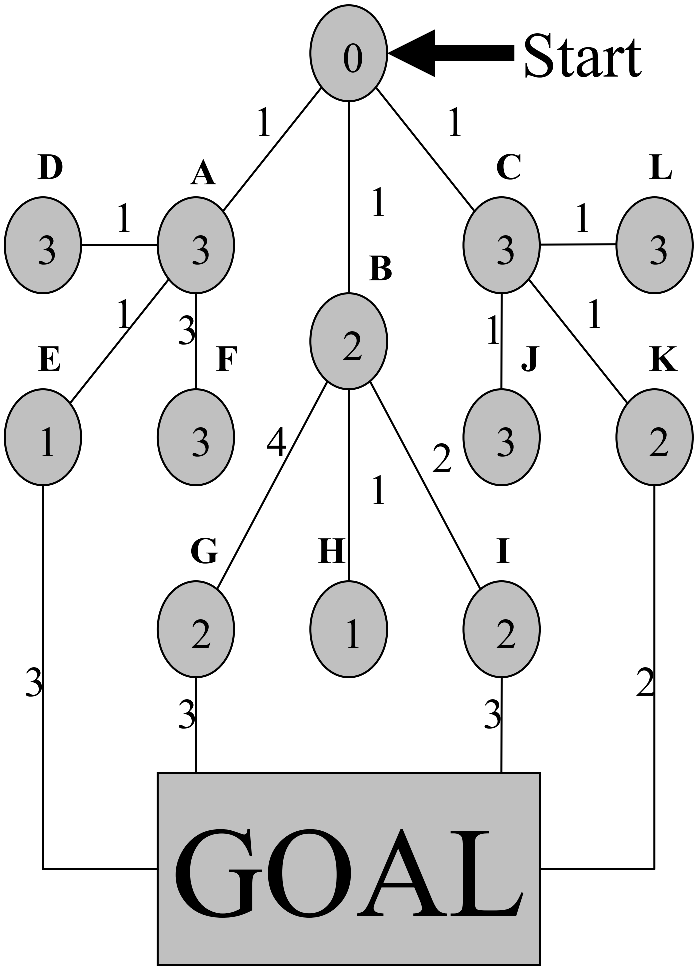


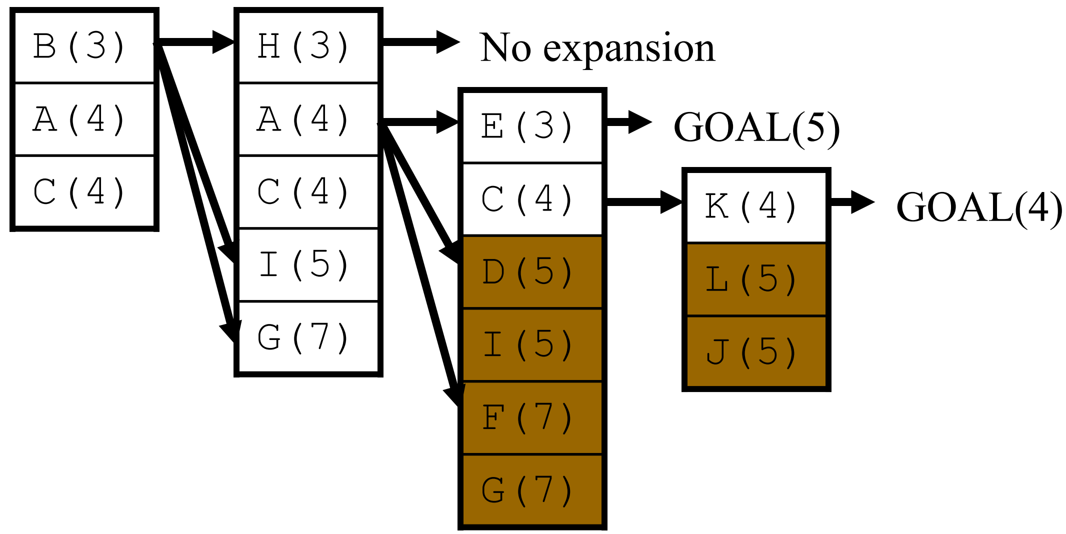
There might be a shorter path, but assuming non-negative arc costs, nodes with a lower priority than the goal cannot yield a better path.

In this example, nodes with a priority greater than or equal to 5 can be pruned.

Why don't we expand nodes with an equivalent priority? (why not expand nodes D and I?)

Example (5/5)





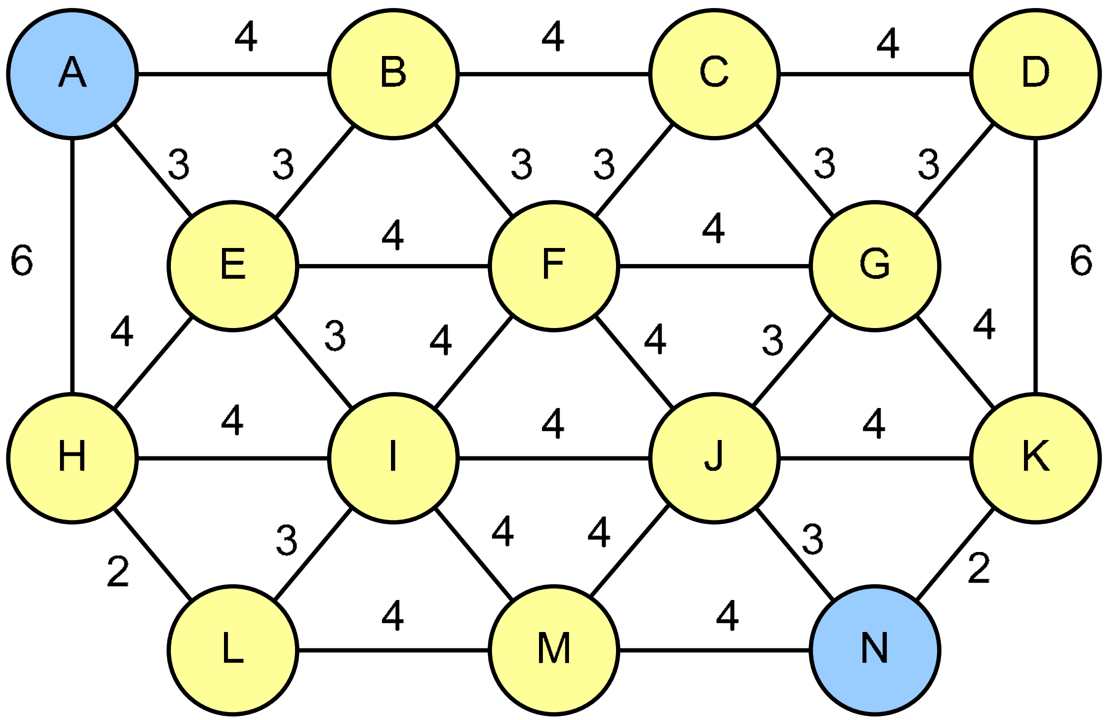
We can continue to throw away nodes with priority levels lower than the lowest goal found.

As we can see from this example, there was a shorter path through node K. To find the path, simply follow the back pointers.

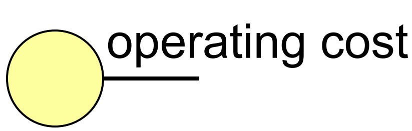
Therefore the path would be:
Start => C => K => Goal

If the priority queue still wasn't empty, we would continue expanding while throwing away nodes with priority lower than 4.
(remember, lower numbers = higher priority)

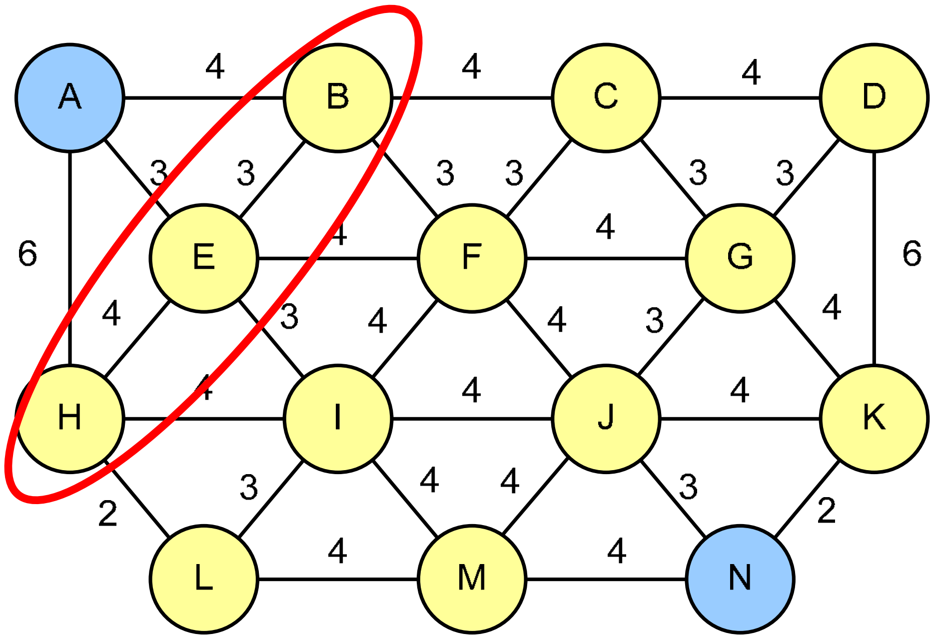
A*: Example (1/6)

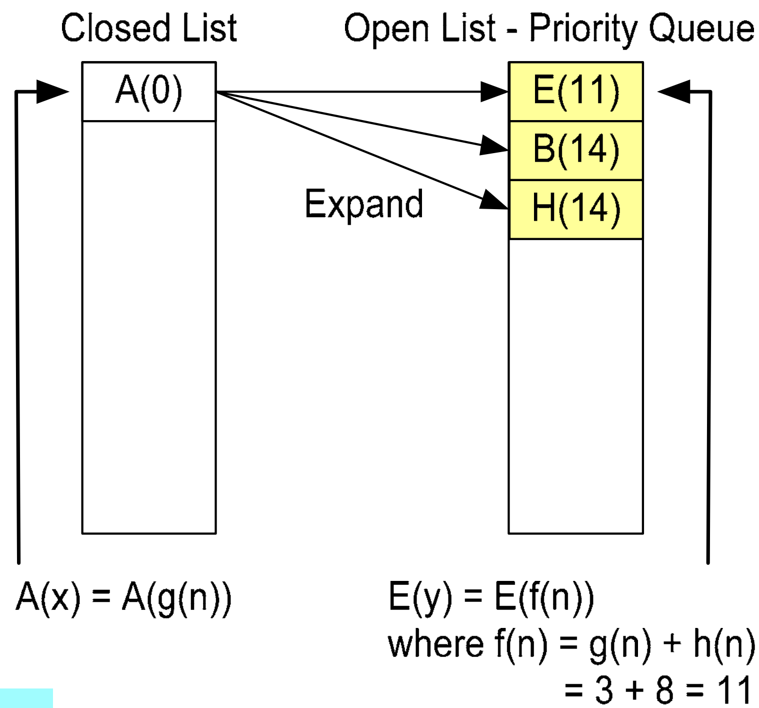


Heuristics	
A = 14	H = 8
B = 10	I = 5
C = 8	J = 2
D = 6	K = 2
E = 8	L = 6
F = 7	M = 2
G = 6	N = 0

Legend 

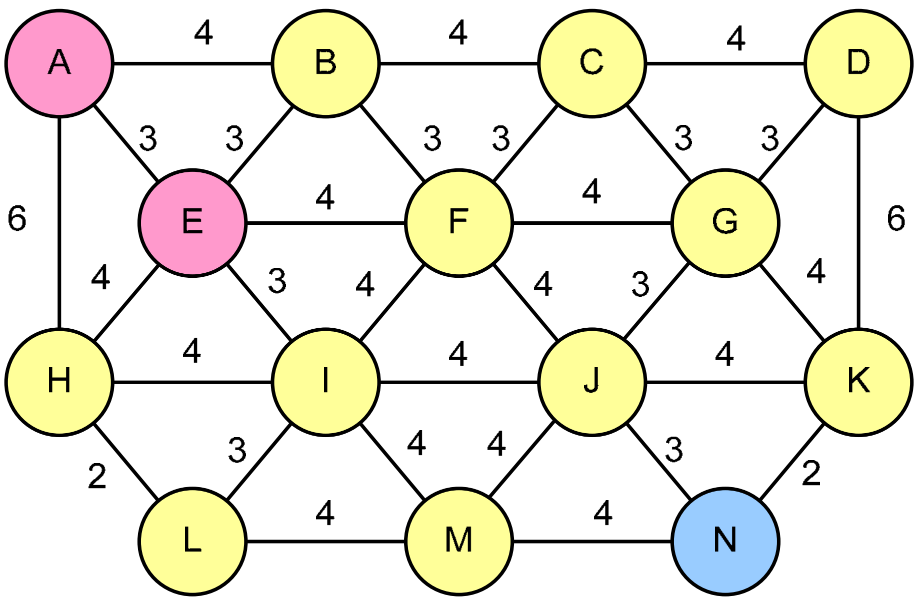
A*: Example (2/6)

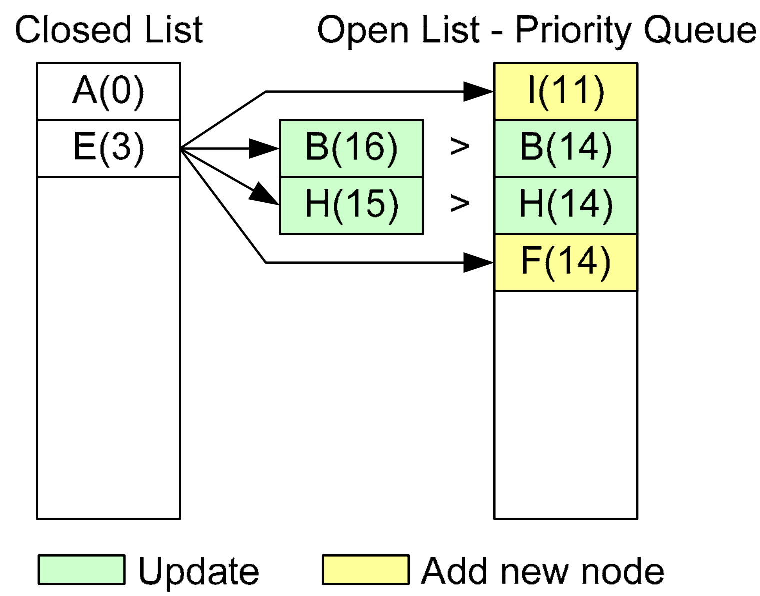




Heuristics
A = 14, B = 10, C = 8, D = 6, E = 8, F = 7, G = 6
H = 8, I = 5, J = 2, K = 2, L = 6, M = 2, N = 0

A*: Example (3/6)

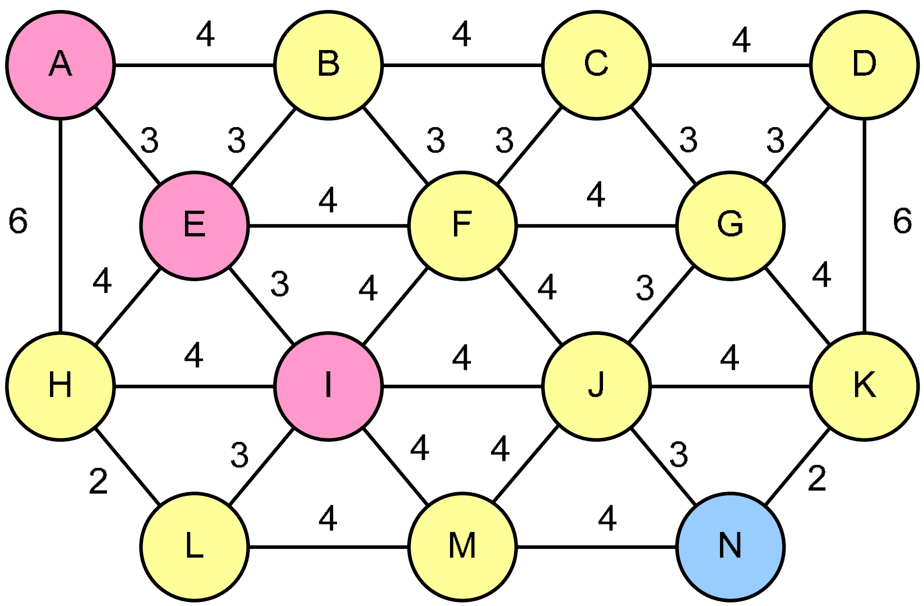


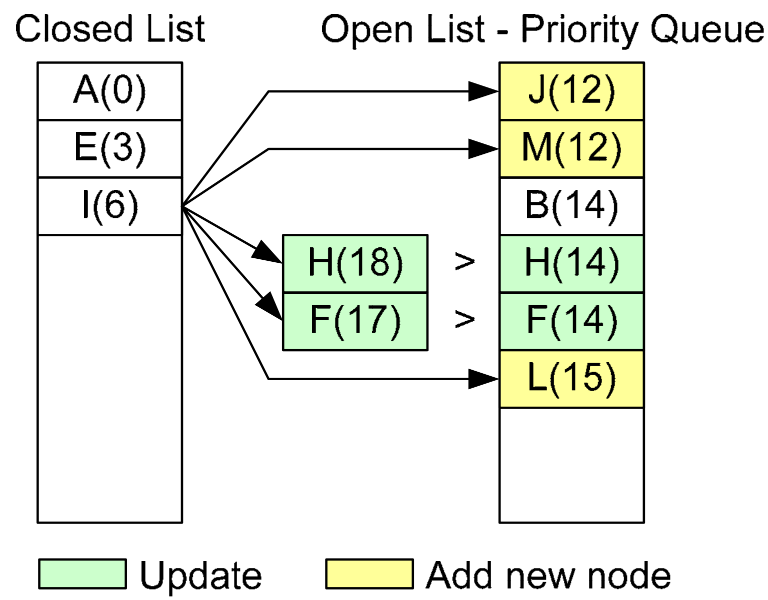


Heuristics
A = 14, B = 10, C = 8, D = 6, E = 8, F = 7, G = 6
H = 8, I = 5, J = 2, K = 2, L = 6, M = 2, N = 0

Since A → B is smaller than A → E → B, the f-cost value of B in an open list needs not be updated

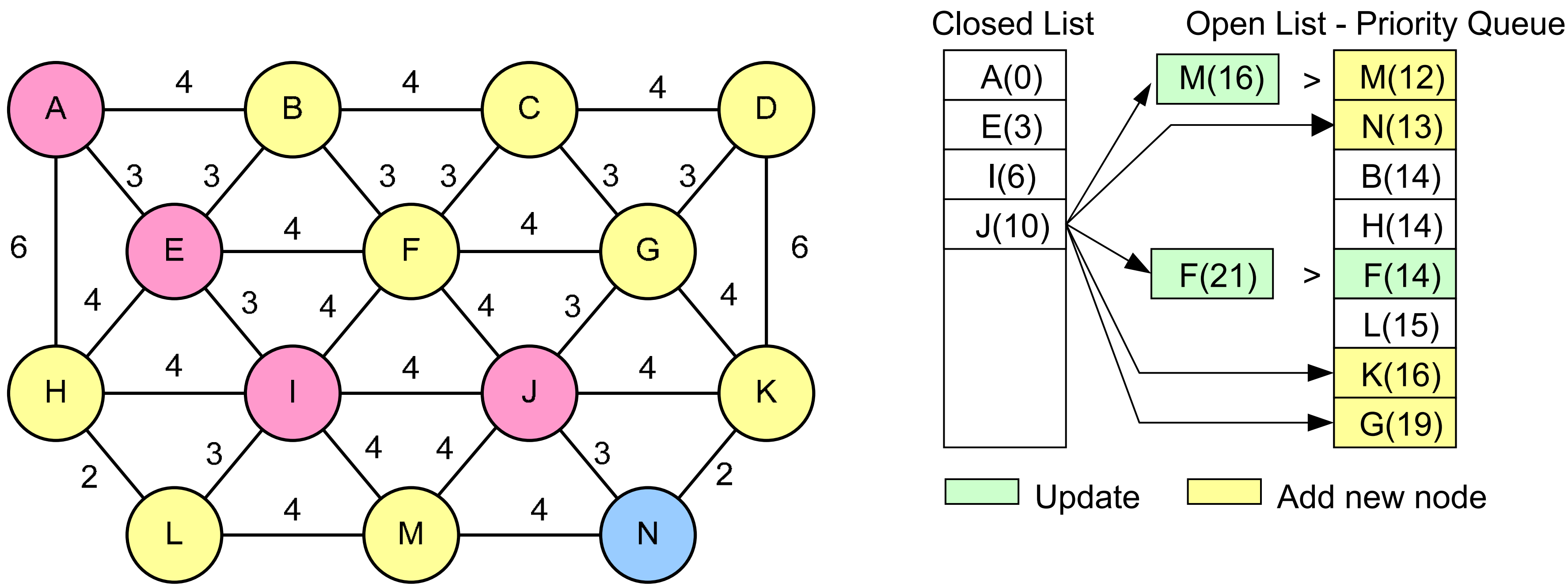
A*: Example (4/6)





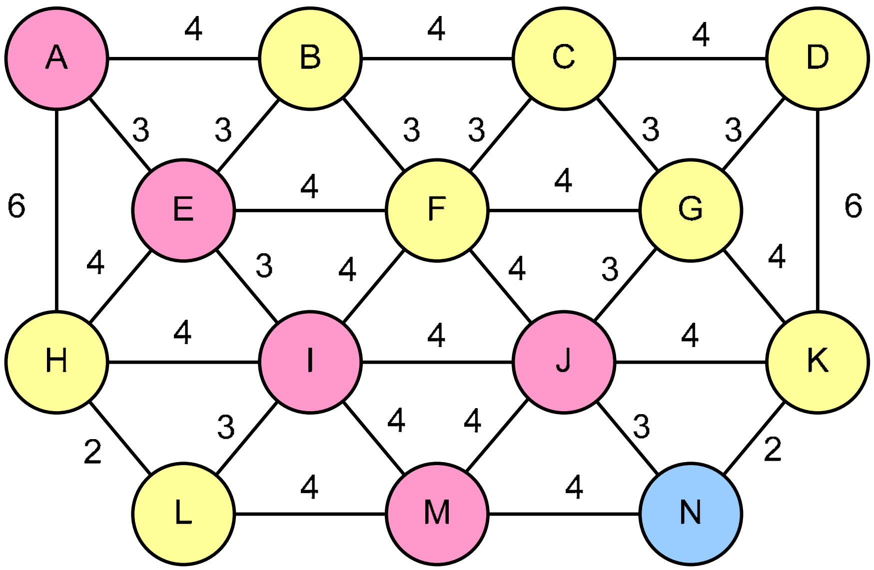
Heuristics
A = 14, B = 10, C = 8, D = 6, E = 8, F = 7, G = 6
H = 8, I = 5, J = 2, K = 2, L = 6, M = 2, N = 0

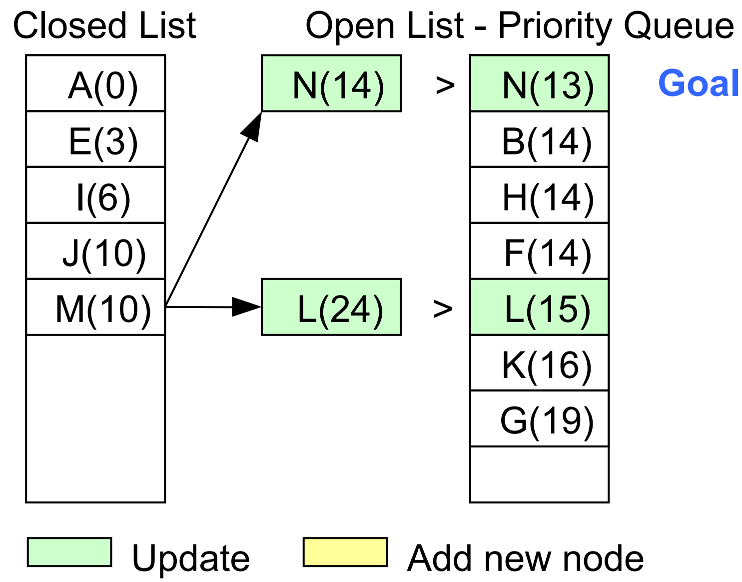
A*: Example (5/6)



Heuristics
A = 14, B = 10, C = 8, D = 6, E = 8, F = 7, G = 6
H = 8, I = 5, J = 2, K = 2, L = 6, M = 2, N = 0

A*: Example (6/6)

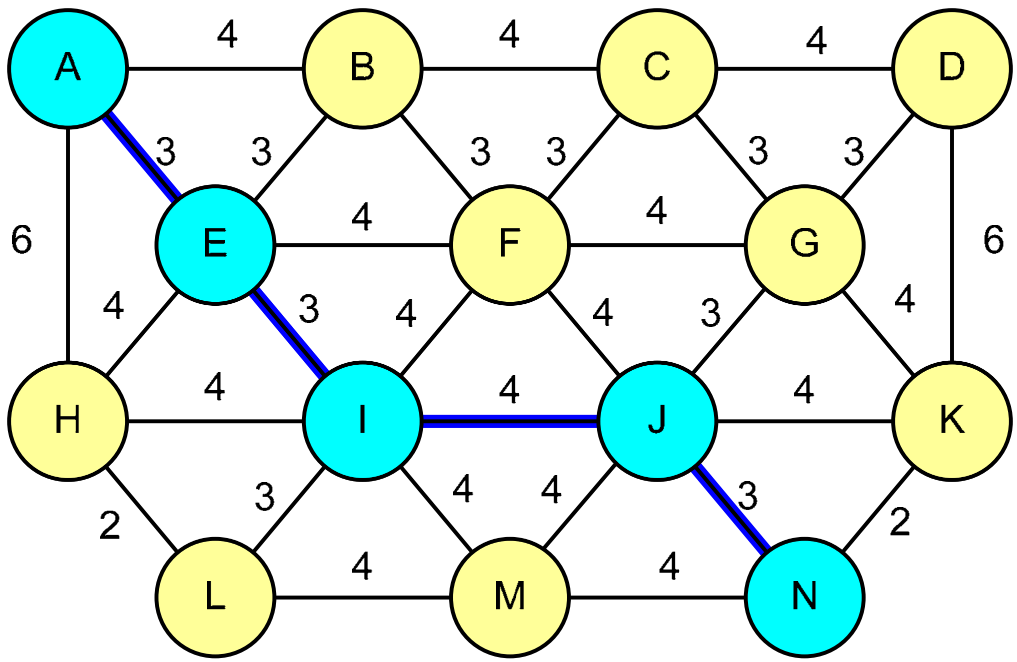




Heuristics
A = 14, B = 10, C = 8, D = 6, E = 8, F = 7, G = 6
H = 8, I = 5, J = 2, K = 2, L = 6, M = 2, N = 0

Since the path to N from M is greater than that from J, **the optimal path to N is the one traversed from J**

A*: Example Result



Generate the path from the goal node back to the start node through the back-pointer attribute