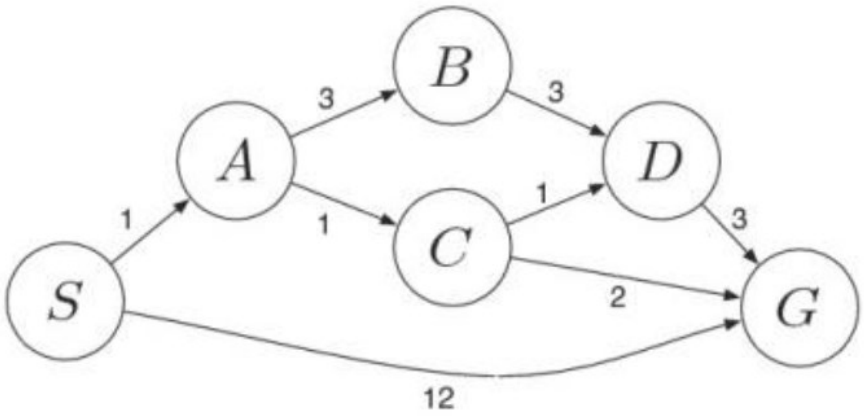
Introduction to Artificial Intelligence Lab

Lab 5: A* Algorithm

A* Search makes use of a priority queue just like Uniform Cost Search with the element stored being the path from the start state to a particular node, but the priority of an element is not the same. In Uniform Cost Search we used the actual cost of getting to a particular node from the start state as the priority. For A*, we use the cost of getting to a node plus the heuristic at that point as the priority. Let n be a particular node, then we define g(n) as the cost of getting to the node from the start state and h(n) as the heuristic at that node. The priority thus is f(n) = g(n) + h(n). The priority is maximum when the f(n) value is least.



Node	h1
S	4
A	2
B	6
C	2
D	3
G	0

Figure 1. Sample graph

Initialization: { [S , 4] }
Iteration1: { [S->A , 3] , [S->G , 12] }
Iteration2: { [S->A->C , 4] , [S->A->B , 10] , [S->G , 12] }
Iteration3: { [S->A->C->G , 4] , [S->A->C->D , 6] , [S->A->B , 10] , [S->G , 12] }
Iteration4 gives the final output as S->A->C->G.

- The algorithm returns the first path encountered. It does not search for all paths.
- The algorithm returns a path which is optimal in terms of cost, if an admissible heuristic is used (this can be proved).

The above example illustrates that A* Search gives the optimal path faster than Uniform Cost Search. This is, however, true only if the heuristic is admissible. In general, the efficiency of the algorithm depends on the quality of the heuristic. The nearer the heuristic is to the actual cost, the better is the speed of the algorithm. Trivially, the heuristic can be taken to be 0, which gives the Uniform Cost Search algorithm.

Exercises

1. Represent the graph in Figure 1 using C++, Java, or any programming language that you know. A node in this graph is represented with its parent node, g value, h value and f value. (You can use the graph data structure codes that you done in previous experiments.)
2. Write a program that implements A* algorithm which calculates the shortest path from a starting node “S” to a goal node “G”.