Department of
Computer Science and Engineering

Title: Implementation of Heuristic A* Search Algorithm

Artificial Intelligence Lab

CSE 404

Green University of Bangladesh

1 Objective(s)

- To understand how A* search works.
- To understand how informed search finds the optimal solution.

2 Problem analysis

A* is a searching technique that is used to detect the shortest path within an initial and a final point. In the implementation, A* always uses weighted graphs. A* algorithm uses a heuristic function to help decide which path to follow next. Heuristic functions are used to provide 'good enough' solutions to very complex problems where finding a perfect solution would take too much time. In A* the heuristic function provides an estimate of the minimum cost between a given node and the target node. Let's say for an example graph, currently, we have reached node n starting from the source node, and we have to reach the goal node. The algorithm will combine the actual cost from the start node - referred to as $G(n)$, with the estimated cost to the target or goal node - referred to as $H(n)$, and uses the result to select the next node to evaluate. So, A* takes the evaluation function as: $F(n) = G(n) + H(n)$

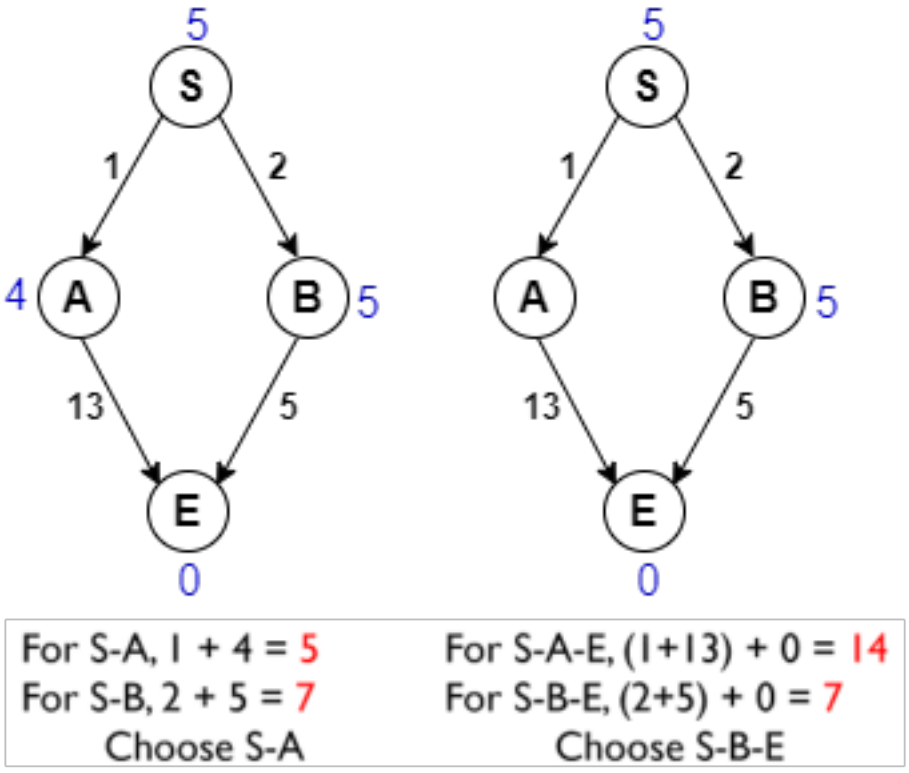


Figure 1: A* algorithm simulation based on heuristic function.

3 Algorithm of A*

The algorithm for the A* search algorithm implemented in this code can be outlined as follows:

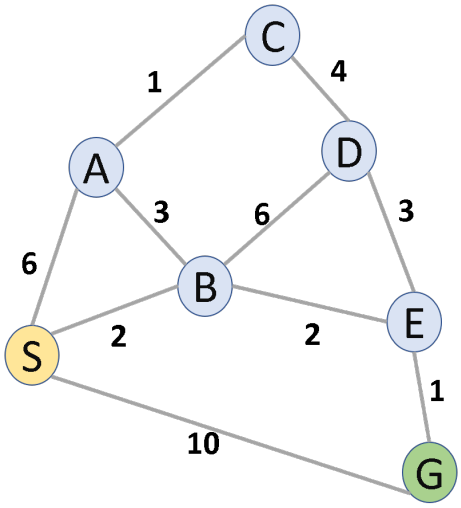
Algorithm 1: A* Algorithm

1. Read the graph from the input file and store it in an adjacency matrix.
2. Read the heuristic values for each node in the graph.
3. Set the start and destination nodes.
4. Create an empty priority queue and add the start node to it.
5. While the priority queue is not empty:
   (a) Extract the node $\boldsymbol{u}$ with the lowest F value from the priority queue.
   (b) If the node $\boldsymbol{u}$ is the destination node, output goal found information and cost for reaching the goal.
   (c) For each neighbor of node $\boldsymbol{u}$ node:
      i. Calculate the G value of the neighbor (the sum of the G value of the removed node and the cost to reach the neighbor).
      ii. Calculate the H value of the neighbor (the heuristic value of the neighbor).
      iii. Calculate the F value of the neighbor (the sum of the G value and the H value).
      iv. Add the neighbor to the priority queue.

Note that the A* search algorithm uses both the G value (the cost to reach the current node from the start node) and the H value (the heuristic estimate of the cost from the current node to the destination node) to calculate the F value (the estimated total cost from the start node to the destination node via the current node). The priority queue is sorted based on the F value, so the nodes with the lowest F value (the most promising nodes) are explored first.

4 Example

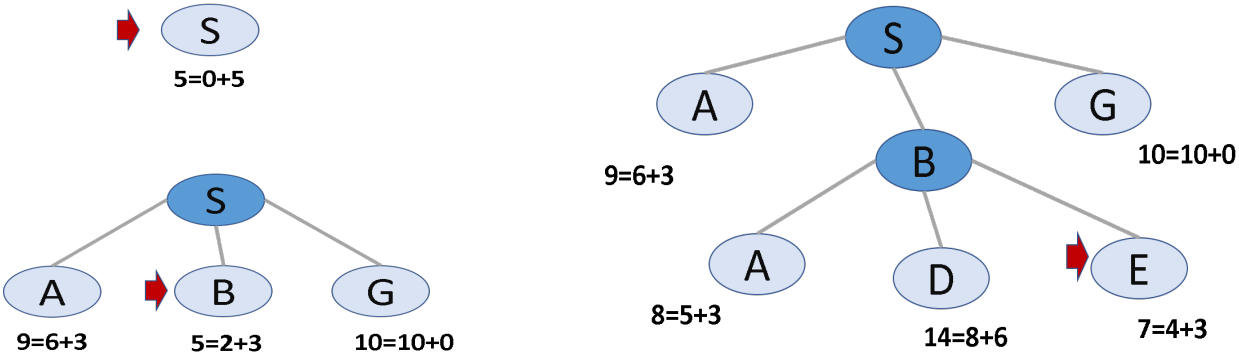
In this case, we'll navigate the given graph, (Fig.2) utilizing the A* algorithm. The heuristic value of all states is given within the underneath table (Fig.2) so we'll calculate the $F(n)$ of each state utilizing the equation $F(n) = G(n) + H(n)$



State	h(n)
S	5
A	3
B	3
C	2
D	6
E	3
G	0

Figure 2: A sample graph and Heuristic table

A* algorithm returns the path which occurred first, and it does not search for all remaining paths. It expands all nodes which satisfy the condition of lowest $F(n)$ value and finally provides the optimal path. At each step, the algorithm selects a node based on the value $F(n)$, a parameter equal to the sum of the other two parameters $G(n)$ and $H(n)$. At each step, the node with the smallest $F(n)$ is selected, and that node is processed.



(a) (b)

Figure 3: Iteration 1 and 2

Starting from the S we have to reach goal node G. Here current node n = S, then the $G(n)$ is 0 and $H(n)$ is 5 as we get from the heuristic table.
Therefore at node S, $F(n) = G(n) + H(n) = 0 + 5 = 5$.
Now, it expands all the nodes from S and finds the lowest F(n) from those nodes for expanding the next nodes. It will continue the steps before getting the final goal with the optimal path having the lowest cost. Considering (Fig.2)the sample graph and Heuristic table, the iteration steps (Fig.3) are described below.

Iteration-1:

S → A, n = A => F(n)= 6 + 3 = 9,
S → B, n = B => F(n) = 2 + 3 = 5,
S → G, n = G => F(n)= 10 + 0 = 10.

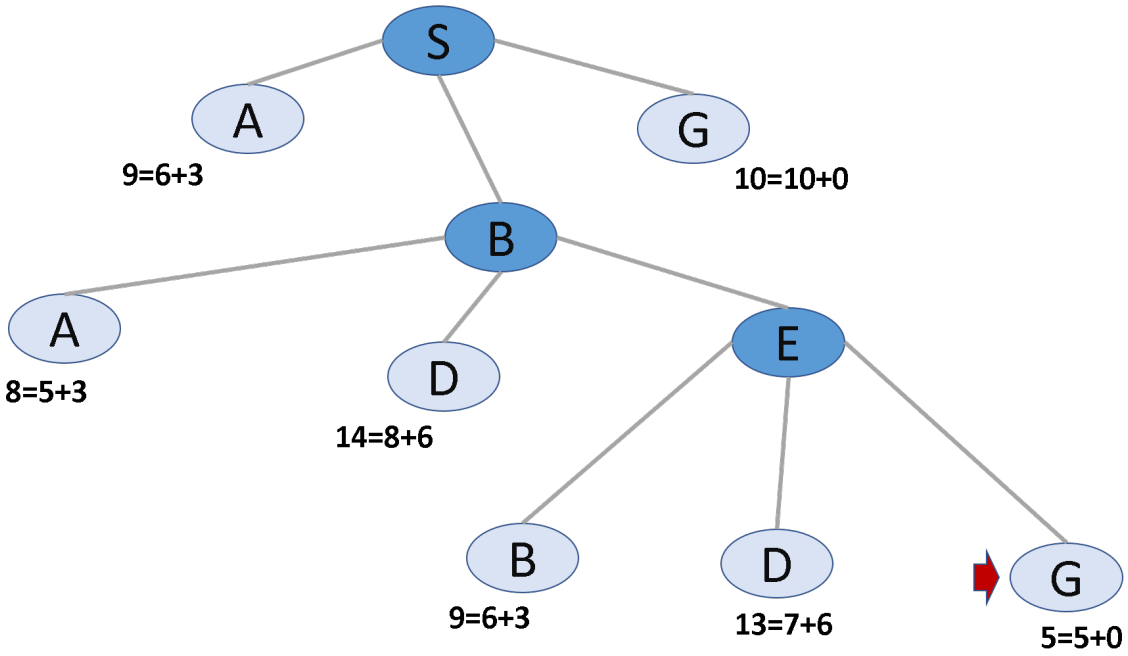


Figure 4: Final state of A* search

As node B holds the lowest F(n) value, we will start traversal from this node.

Iteration-2:

B → A, n = B => F(n)=5 + 3 = 8,
B → D, n = D => F(n) = 8 + 6 = 14,
B → E, n = E => F(n) = 4 + 3 = 7.
Using a similar strategy we will now expand node E, as it has the lowest F(n) value. The calculation is

omitted here for your exercise. Fig. 4 is the final search tree starting from node S and reaching goal node G. The optimal cost for reaching the goal node is 5 and the path is S → B → E → G.

5 Implementation in Java

```
1  import java.io.FileNotFoundException;
2  import java.util.Comparator;
3  import java.util.PriorityQueue;
4  import java.util.Scanner;
5
6  /**
7   *
8   * @author Jargis
9   */
10 class Node {
11
12     int u, GofN, HofN, FofN;
13
14     Node(int x, int val1, int val2) {
15         u = x;
16         GofN = val1;
17         HofN = val2;
18         FofN = GofN + HofN;
19     }
20 }
21
22 class NodeComparator implements Comparator<Node> {
23
24     public int compare(Node n1, Node n2) {
25         return n1.FofN - n2.FofN;
26     }
27 }
28
29 public class A_star_search {
30
31     public static void main(String[] args) throws FileNotFoundException {
32         // TODO code application logic here
33         int N = 7;      //Number of Nodes
34         //graph is here.
35         int[][] arr = {
36             {0, 6, 2, 0, 0, 0, 10},
37             {6, 0, 3, 1, 0, 0, 0},
38             {2, 3, 0, 0, 6, 2, 0},
39             {0, 1, 0, 0, 4, 0, 0},
40             {0, 0, 6, 4, 0, 3, 0},
41             {0, 0, 2, 0, 3, 0, 1},
42             {10, 0, 0, 0, 0, 1, 0}
43         };
44         int heuristic[] = {5,3,3,2,6,3,0};      //Heuristic values
45
46         int start = 0;      //Starting node
47         int dest = 6;       //Destination node
48         PriorityQueue<Node> pq = new PriorityQueue<Node>(20, new NodeComparator
               ());
49
50         //Insert source node to queue
51         pq.add(new Node(0, 0, heuristic[start]));
```

```
52          boolean flag = false;
53
54          //Simulating the main logic
55          while (!pq.isEmpty()) {
56              Node parent = pq.poll();
57              if (parent.u == dest) {
58                  System.out.println("Goal Found and cost = " + parent.GofN);
59                  flag = true;
60                  break;
61              }
62              for (int i = 0; i < N; i++) {
63                  if (arr[parent.u][i] != 0) {
64                      Node child = new Node(i, parent.GofN + arr[parent.u][i],
                            heuristic[i]);
65                      pq.add(child);
66                  }
67              }
68          }
69          if (!flag) {
70              System.out.println("Not possible to reach goal node");
71          }
72      }
73  }
```

6 Sample Input/Output (Compilation, Debugging & Testing)

Output:
Goal Found and cost = 5

7 Lab Task (Please implement yourself and show the output to the instructor)

- Write a program to perform the A* Algorithm on the following figure. Explicitly Print the queue at each step.

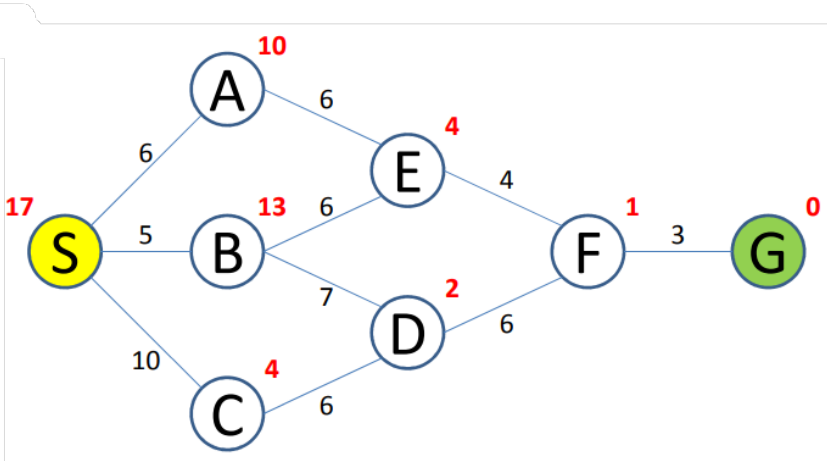


Figure 5: A sample graph of A*

8 Lab Exercise (Submit as a report)

- Scenario: You are a robot that needs to navigate a grid-based maze to reach a goal location. The maze is represented as a 2D grid, where each cell can be either empty or blocked. The robot can move up, down, left, or right to adjacent cells in the grid. The cost of moving from one cell to another is one. The robot has a limited battery life, and its movement consumes the battery. The robot starts at the top-left corner of the maze, and the goal is at the bottom-right corner. The battery life of the robot is equal to the Manhattan distance between the start and goal positions.

- Problem: Draw the graph and write an A* search algorithm to find the shortest path from the robot's starting position to the goal position while minimizing battery consumption. The algorithm should take into account the battery life of the robot and avoid paths that will consume more battery life.

  To solve this problem, we can represent each cell in the grid as a node in a graph. The edges between nodes correspond to valid movements from one cell to an adjacent cell in the grid. The cost of moving between cells is one, and the heuristic value represents the estimated proportion of battery life remaining to reach the goal from the current node. The A* search algorithm can utilize this heuristic to guide its search, favoring a path that will require a minimum battery life.

  Sample 2D-grid:

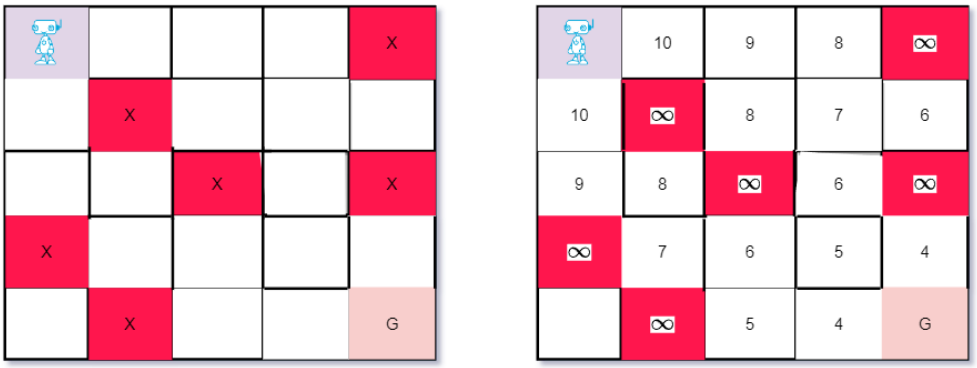


Figure 6: Sample 2D-grid of A*

9 Policy

Copying from internet, classmate, seniors, or from any other source is strongly prohibited. 100% marks will be *deducted* if any such copying is detected.