Assignment: Implement A star algorithm
Batch:A1
Roll no:3330

```java
import java.util.Arrays;
import java.util.Scanner;
public class AStar {

       public static double aStar(int[][] graph, double[] heuristic, int start, int
goal) {

             //distance from start node to all other nodes
       int[] distance = new int[graph.length];
       Arrays.fill(distance, Integer.MAX_VALUE);

       //distance from start node to itself is zero
       distance[start] = 0;


       double[] f = new double[graph.length];
       Arrays.fill(f, Integer.MAX_VALUE);

       f[start] = heuristic[start];
       boolean[] visited = new boolean[graph.length];

       int [] path=new int[graph.length];
       int p=0;
       int lowestPriorityIndex=-1;

       //while there are nodes left to visit
       while (true) {

        //find the node with the currently lowest f value
           double lowestPriority = Integer.MAX_VALUE;
           lowestPriorityIndex = -1;

           for (int i = 0; i < pq.length; i++) {

                if (f[i] < lowestPriority && !visited[i]) {

                      lowestPriority = f[i];
                   lowestPriorityIndex = i;
                }
             }

           if (lowestPriorityIndex == -1) {
             return -1;
           }
           else if (lowestPriorityIndex == goal) {
             System.out.println("Goal node found!");
             break;
            }
         //for all adjacent nodes that have not yet been visited
           for (int i = 0; i < graph[lowestPriorityIndex].length; i++) {
                if (graph[lowestPriorityIndex][i] != 0 && !visited[i]) {

                       if (distance[lowestPriorityIndex] +
graph[lowestPriorityIndex][i] < distance[i]) {

                             //if path over this edge is shorter save this path as
the new shortest path

                             distance[i] = distance[lowestPriorityIndex] +
graph[lowestPriorityIndex][i];
                       f[i] = distance[i] + heuristic[i];
                  }
               }
           }//close for

           visited[lowestPriorityIndex] = true;
           path[p]=lowestPriorityIndex;
           p=p+1;
          }  //close while loop
        path[p]=goal;
        int i=0;

        for(i=0;i<p;i++) {
            System.out.print(path[i]+" --> ");
        }
        System.out.print(path[i]);
        System.out.println();
         return distance[lowestPriorityIndex];
}
public static void main(String [] args) {
      Scanner sc = new Scanner(System.in);

      System.out.println("Enter number of node: ");
      int n = sc.nextInt();

      System.out.println("Enter number of edges: ");
      int e = sc.nextInt();

      int [][] graph = new int [n][e];

      for(int i=0;i<e;i++) {

       System.out.println("Enter starting vertex: ");
       int x = sc.nextInt();
       System.out.println("Enter ending vertex: ");
       int y = sc.nextInt();

       System.out.println("Enter weight of the edge:");
       int w = sc.nextInt();

       graph[x][y]=w;
     }

      double [] heuristic= new double[n];
      for(int i=0;i<n;i++) {
           System.out.println("Enter heuristic cost for node "+i+": ");
           heuristic[i] = sc.nextInt();
      }

      System.out.println("Enter start node: ");
      int start = sc.nextInt();
      System.out.println("Enter end node: ");
      int end = sc.nextInt();

      double ans=aStar(graph, heuristic,start,end);

      System.out.println("Cost of optimal path: "+ans);
      sc.close();
}
}
```

OUTPUT

```
Enter number of node:
6
Enter number of edges:
8
Enter starting vertex:
0
Enter ending vertex:
1
Enter weight of the edge:
1
Enter starting vertex:
0
Enter ending vertex:
5
Enter weight of the edge:
12
Enter starting vertex:
1
Enter ending vertex:
4
Enter weight of the edge:
1
Enter starting vertex:
1
Enter ending vertex:
2
Enter weight of the edge:
3
Enter starting vertex:
2
Enter ending vertex:
3
Enter weight of the edge:
3
Enter starting vertex:
4
```

```
Enter ending vertex:
3
Enter weight of the edge:
1
Enter starting vertex:
4
Enter ending vertex:
5
Enter weight of the edge:
2
Enter starting vertex:
3
Enter ending vertex:
5
Enter weight of the edge:
3
Enter heuristic cost for node 0:
4
Enter heuristic cost for node 1:
2
Enter heuristic cost for node 2:
6
Enter heuristic cost for node 3:
3
Enter heuristic cost for node 4:
2
Enter heuristic cost for node 5:
0
Enter start node:
0
Enter end node:
5
Goal node found!
0 --> 1 --> 4 --> 5
Cost of optimal path: 4.0
```