AI LAB ASSIGNMENT 5

NAME: PRATHAPANI SATWIKA

REG NO: 20BCD7160

Search list of items by using Best First Search

CODE:

```
import java.util.Comparator;

import java.util.InputMismatchException;

import java.util.PriorityQueue;

import java.util.Scanner;


public class BestFirstSearch

{

    private PriorityQueue<Vertex> priorityQueue;

    private int heuristicvalues[];

    private int numberOfNodes;


    public static final int MAX_VALUE = 999;


    public BestFirstSearch(int numberOfNodes)

    {

        this.numberOfNodes = numberOfNodes;
        this.priorityQueue = new PriorityQueue<Vertex>(this.numberOfNodes,
        new Vertex());
    }

    public void bestFirstSearch(int adjacencyMatrix[][], int[] heuristicvalues,int
source)
    {
        int evaluationNode;
        int destinationNode;
        int visited[] = new int [numberOfNodes + 1];
        this.heuristicvalues = heuristicvalues;

        priorityQueue.add(new Vertex(source, this.heuristicvalues[source]));
        visited[source] = 1;

        while (!priorityQueue.isEmpty())
        {
          evaluationNode = getNodeWithMinimumHeuristicValue();
          destinationNode = 1;

          System.out.print(evaluationNode + "\t");
          while (destinationNode <= numberOfNodes)
          {
            Vertex vertex = new
Vertex(destinationNode,this.heuristicvalues[destinationNode]);
```

```
                if ((adjacencyMatrix[evaluationNode][destinationNode] != MAX_VALUE

                      && evaluationNode != destinationNode)&& visited[destinationNode]
== 0)
                {
                    priorityQueue.add(vertex);
                    visited[destinationNode] = 1;
                }
                destinationNode++;
            }
        }
    }

    private int getNodeWithMinimumHeuristicValue()
    {
        Vertex vertex = priorityQueue.remove();
        return vertex.node;
    }

    public static void main(String... arg)
    {
        int adjacency_matrix[][];
        int number_of_vertices;
        int source = 0;
        int heuristicvalues[];
        Scanner scan = new Scanner(System.in);
        try
        {
            System.out.println("Enter the number of vertices");
            number_of_vertices = scan.nextInt();
            adjacency_matrix = new int[number_of_vertices + 1][number_of_vertices
+ 1];
            heuristicvalues = new int[number_of_vertices + 1];

            System.out.println("Enter the Weighted Matrix for the graph");
            for (int i = 1; i <= number_of_vertices; i++)
            {
                for (int j = 1; j <= number_of_vertices; j++)
                {
                    adjacency_matrix[i][j] = scan.nextInt();
                    if (i == j)
                    {
                        adjacency_matrix[i][j] = 0;
                        continue;
                    }
                    if (adjacency_matrix[i][j] == 0)
                    {
                        adjacency_matrix[i][j] = MAX_VALUE;
                    }
```

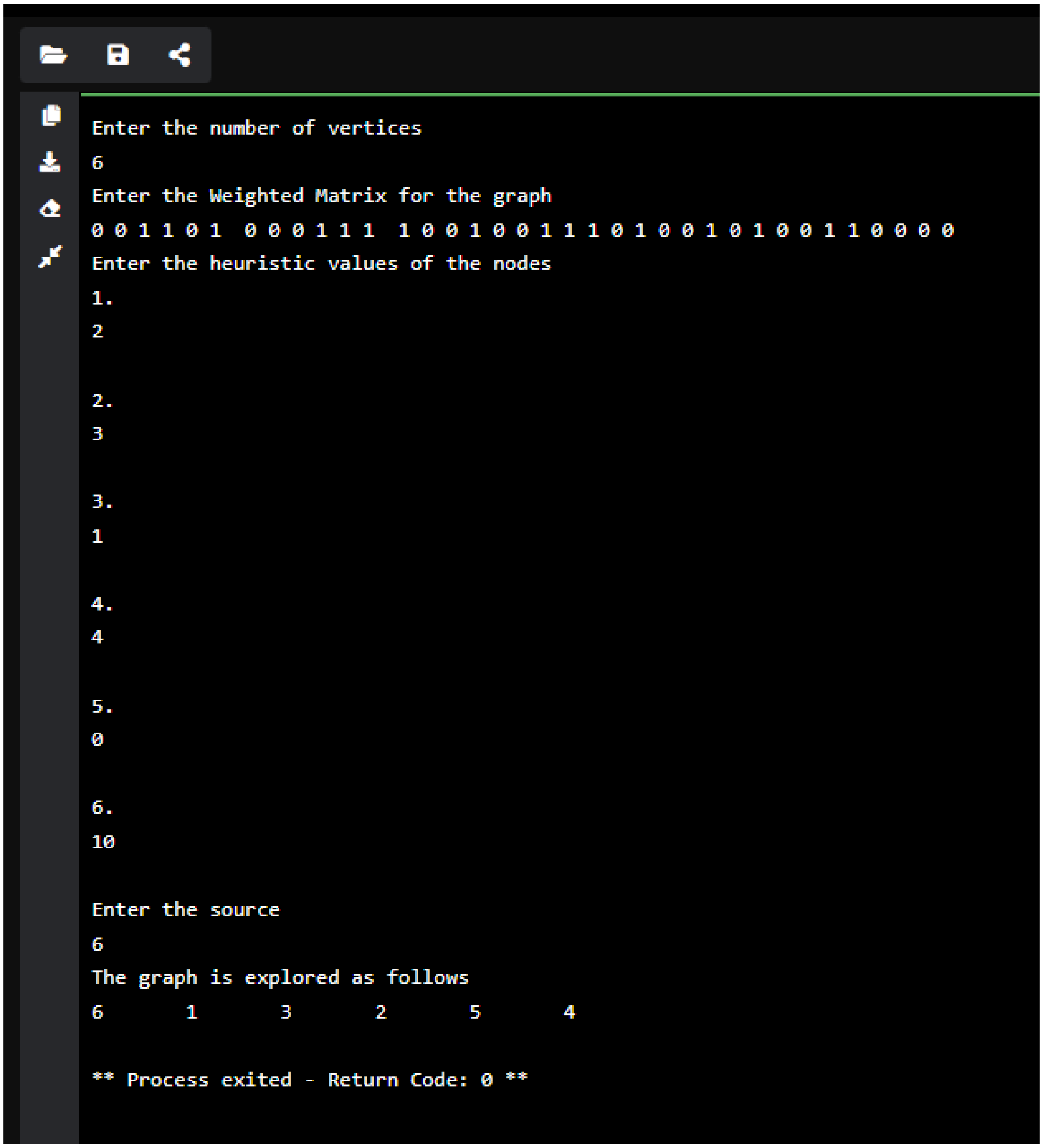
```
    }
}
for (int i = 1; i <= number_of_vertices; i++)
{
    for (int j = 1; j <= number_of_vertices; j++)
    {
        if (adjacency_matrix[i][j] == 1 && adjacency_matrix[j][i] == 0)
        {
            adjacency_matrix[j][i] = 1;
        }
    }
}

System.out.println("Enter the heuristic values of the nodes");
for (int vertex = 1; vertex <= number_of_vertices; vertex++)
{
    System.out.print(vertex + ".");
    heuristicvalues[vertex] = scan.nextInt();
    System.out.println();
}

System.out.println("Enter the source ");
source = scan.nextInt();
```

```
            System.out.println("The graph is explored as follows");
            BestFirstSearch bestFirstSearch = new
BestFirstSearch(number_of_vertices);
            bestFirstSearch.bestFirstSearch(adjacency_matrix, heuristicvalues,source);

        } catch (InputMismatchException inputMismatch)
        {
            System.out.println("Wrong Input Format");
        }
        scan.close();
    }
}

class Vertex implements Comparator<Vertex>
{
    public int heuristicvalue;
    public int node;

    public Vertex(int node, int heuristicvalue)
    {
        this.heuristicvalue = heuristicvalue;
        this.node = node;
    }

    public Vertex()
```

```
{

}

@Override
public int compare(Vertex vertex1, Vertex vertex2)
{
   if (vertex1.heuristicvalue < vertex2.heuristicvalue)
      return -1;
   if (vertex1.heuristicvalue > vertex2.heuristicvalue)
      return 1;
   return 0;
}

@Override
public boolean equals(Object obj)
{
   if (obj instanceof Vertex)
   {
      Vertex node = (Vertex) obj;
      if (this.node == node.node)
      {
         return true;
      }
```

```
        }
        return false;
    }
}
```

OUTPUT:



```
Enter the number of vertices
6
Enter the Weighted Matrix for the graph
0 0 1 1 0 1  0 0 0 1 1 1  1 0 0 1 0 0 1 1 1 0 1 0 0 1 0 1 0 0 1 1 0 0 0 0
Enter the heuristic values of the nodes
1.
2

2.
3

3.
1

4.
4

5.
0

6.
10

Enter the source
6
The graph is explored as follows
6       1       3       2       5       4

** Process exited - Return Code: 0 **
```