1. **Program JAVA BEST (BEST FIRST SEARCH).**

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;

}

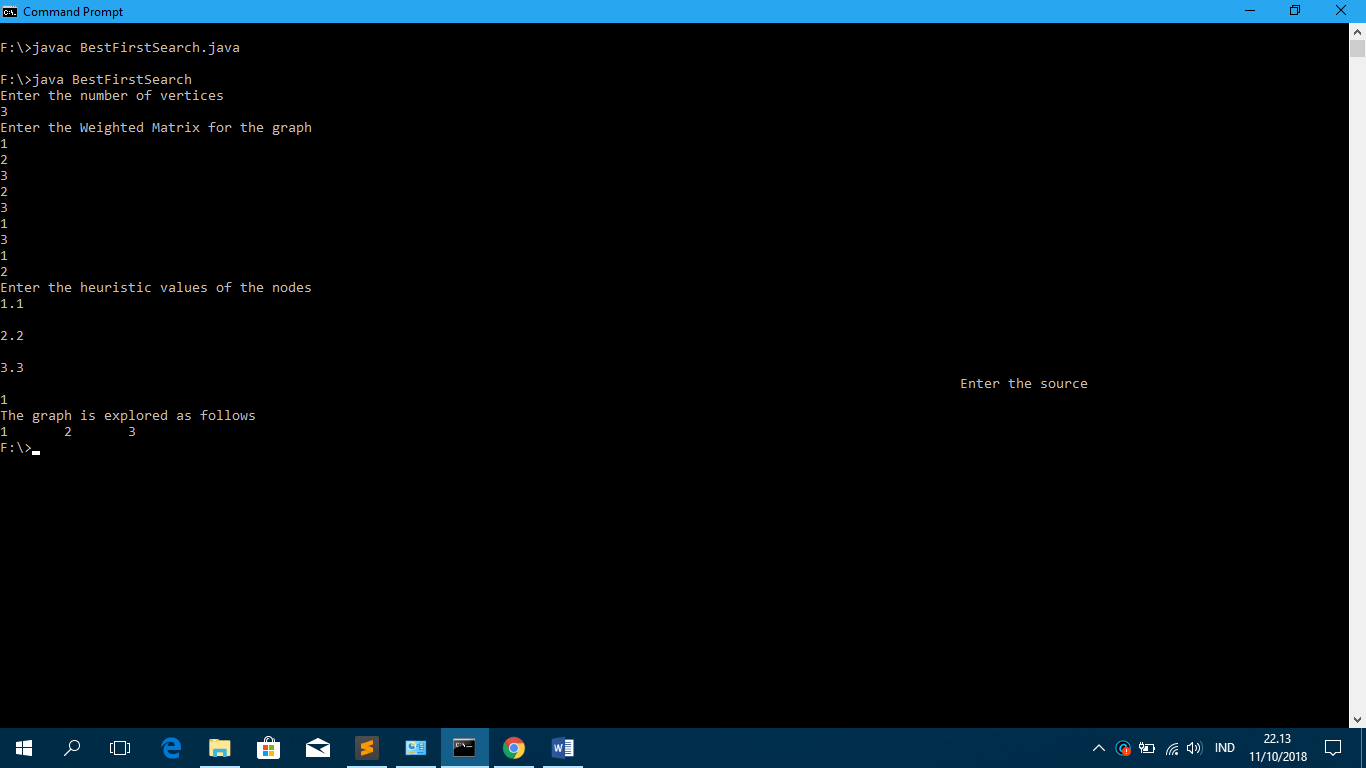
}

return false;

}

}

1. **Dokumentasi**



1. **Penjelasan Kode Program dan Waktu Eksekusi**

Program Java ini, mengimplementasikan Pencarian Pertama-Pertama. Pencarian pertama adalah algoritma pencarian yang mengeksplorasi grafik dengan memperluas simpul yang paling menjanjikan yang dipilih sesuai dengan aturan yang ditentukan.   
Judea Pearl mendeskripsikan pencarian pertama terbaik sebagai memperkirakan janji node n oleh "fungsi evaluasi heuristik yang, secara umum, mungkin bergantung pada deskripsi n, deskripsi tujuan, informasi yang dikumpulkan oleh pencarian hingga titik itu, dan yang paling penting, tentang pengetahuan tambahan tentang domain masalah.

Dengan waktu eksekusi 0,000123 second.