**Batch: A1 Roll No.: 1911004**

**Experiment No 4b**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **Title:** Implementation ofSingle source shortest path by Greedy strategy |

**Objective:** To learn the Greedy strategy of solving the problems for different types of problems

**CO to be achieved:**

|  |  |
| --- | --- |
| SrNo | Objective |
| CO 1 | Analyze the asymptotic running time and space complexity of algorithms. |
| CO 2 | Describe various algorithm design strategies; solve different problems,analyze Complexity. |
| CO 3 | Develop string matching techniques |
| CO 4 | Describe the classes P, NP, and NP-Complete |

**Books/ Journals/ Websites referred:**

1. **1. Ellis horowitz, Sarataj Sahni, S.Rajsekaran,” Fundamentals of computer algorithm”, University Press**
2. **T.H.Cormen ,C.E.Leiserson,R.L.Rivest and C.Stein,” Introduction to algortihtms”,2nd Edition ,MIT press/McGraw Hill,2001**
3. **https://www.mpi-inf.mpg.de/~mehlhorn/ftp/ShortestPathSeparator.pdf**
4. **en.wikipedia.org/wiki/Shortest\_path\_problem**
5. **www.cs.princeton.edu/~rs/AlgsDS07/15ShortestPaths.pdf**

**Pre Lab/ Prior Concepts:**

Data structures, Concepts of algorithm analysis

**Historical Profile:**

Sometimes the problems have more than one solution. With the size of the problem, every time it’s not feasible to solve all the alternative solutions and choose a better one. The greedy algorithms aim at choosing a greedy strategy as solutioning method and proves how the greedy solution is better one.

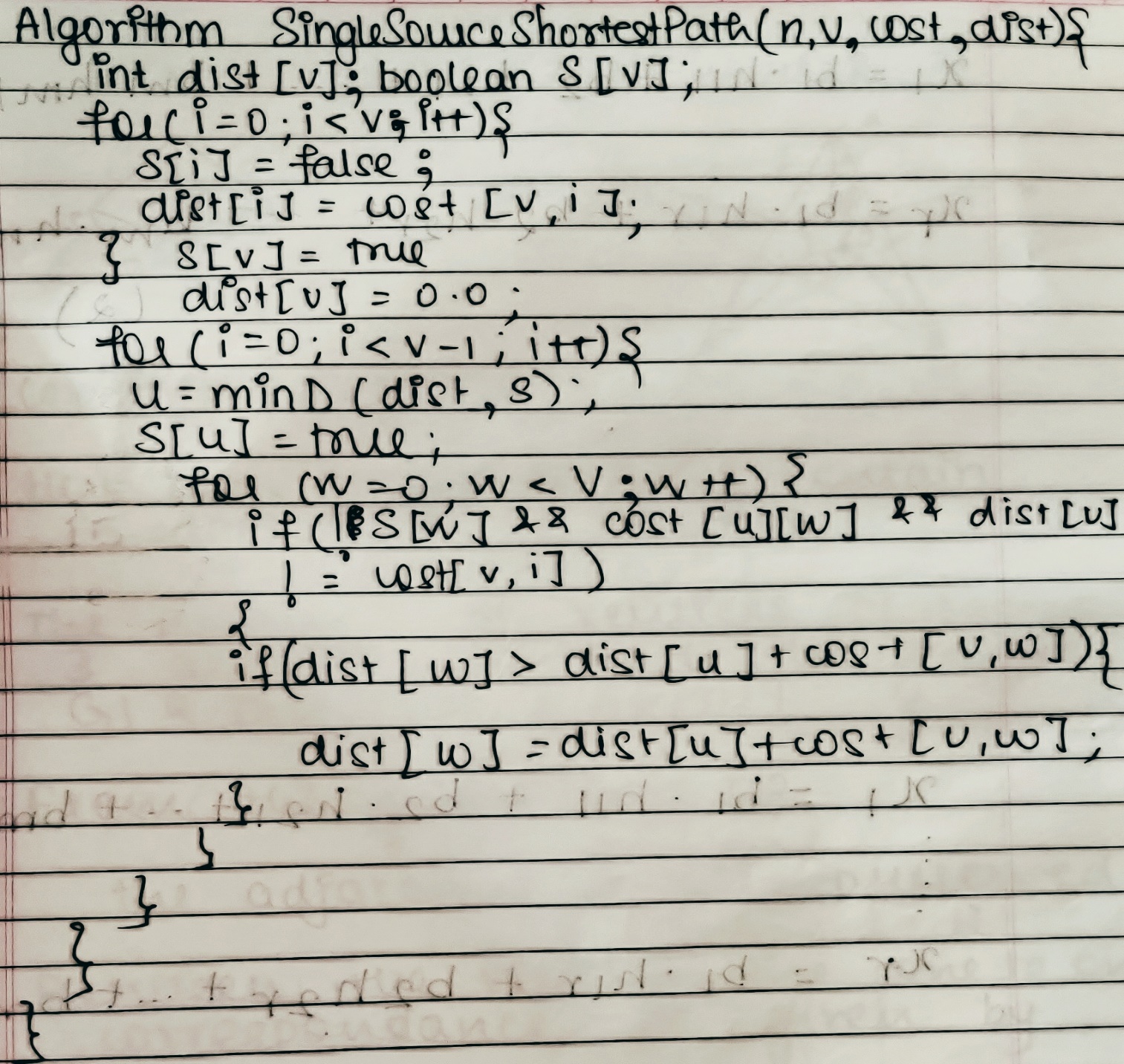
Though greedy algorithms do not guarantee optimal solution, they generally give a better and feasible solution.

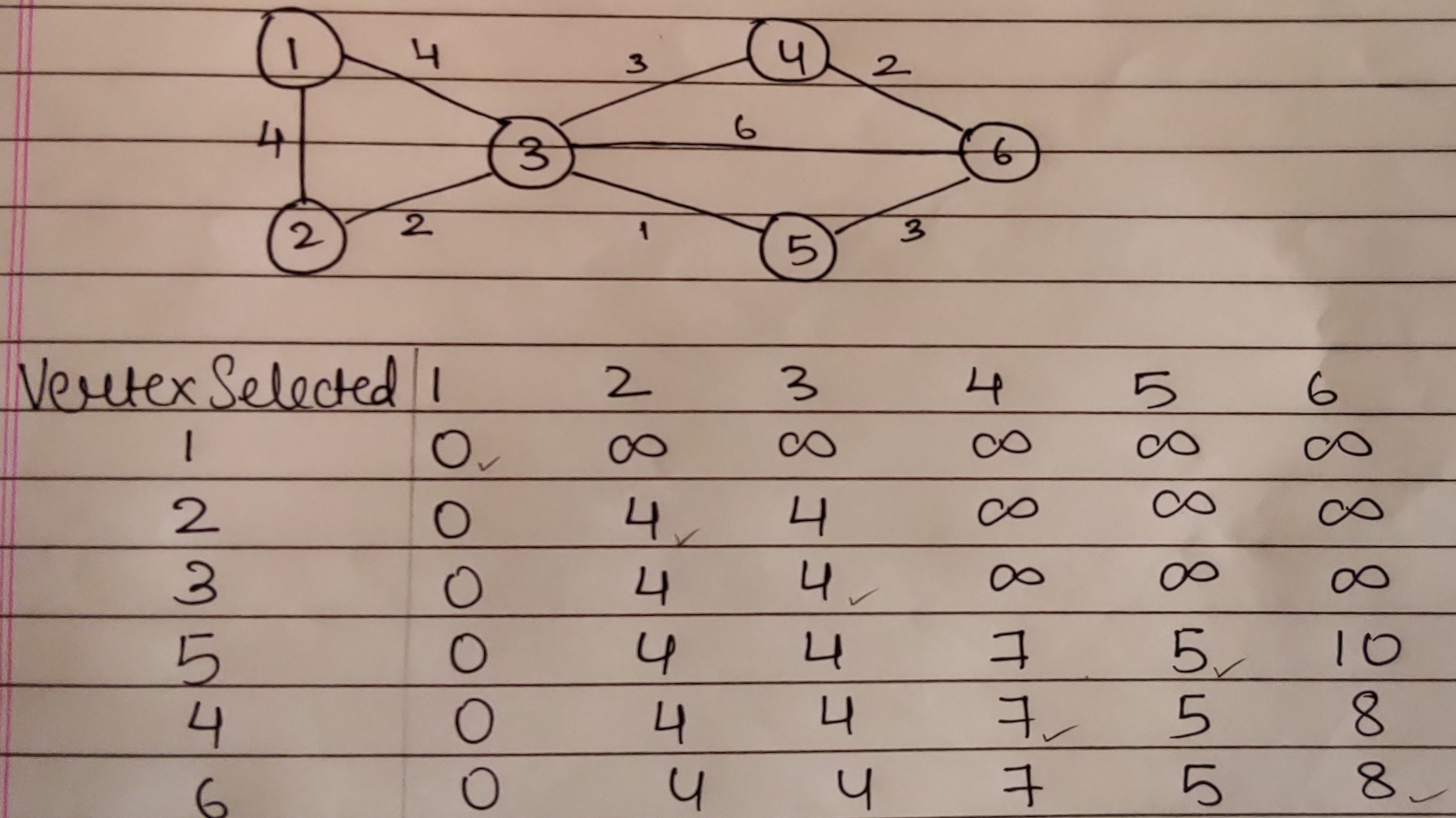
The path finding algorithms work on graphs as input and represent various problems in the real world.

**New Concepts to be learned:**

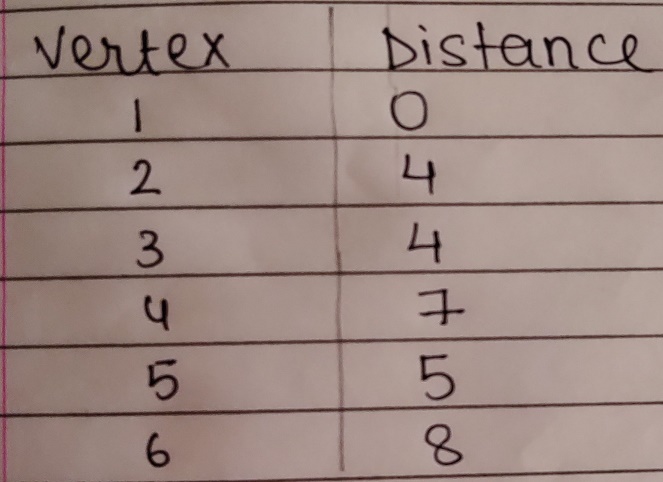
Application of algorithmic design strategy to any problem, Greedy method of problem solving Vs other methods of problem solving, optimality of the solution

**Algorithm**:



**Example Graph:** 

**Solution:**



**Implemented Code**

import java.util.\*;

class Main{

public static void main(String[] args){

Scanner ob=new Scanner(System.in);

System.out.println("Dijkstra Algorithm");

System.out.print("Enter No of Nodes ");

int i,j,n=ob.nextInt();

int cost[][]=new int[n][n];

int dist[]=new int[n];

int calc[][]=new int[100][101];

Path path[]= new Path[n];

boolean visited[]=new boolean[n];

System.out.print("Enter No of Edges ");

int e=ob.nextInt();

for(i=0;i<n;i++){

for(j=0;j<n;j++)

cost[i][j]=-1;

dist[i]=9999;

visited[i]=false;

}

System.out.print("Enter Start End Cost\n");

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

cost[ob.nextInt()-1][ob.nextInt()-1]=ob.nextInt();

System.out.print("Enter Source Vertex ");

int src=ob.nextInt();

dist[src-1]=0;

shortest(cost,dist,visited,src,calc,0);

System.out.print("\nVertex selected ");

for(i=1;i<=n;i++)

System.out.print("{"+i+"} ");

System.out.println("");

for(i=0;i<n;i++){

System.out.print(String.format("%-16d ",calc[i][0]));

for(j=1;j<n+1;j++)

System.out.print(String.format("%-5d ",calc[i][j]));

System.out.println("");

}

System.out.println("Final distances :");

System.out.println("Vertex "+" Distance");

for(i=0;i<n;i++){

System.out.println(" "+(i+1)+"\t "+dist[i]);

path[i]= new Path(dist[i],i+1);

}

System.out.println("Path ");

ptrPath(path);

}

static void shortest(int cost[][],int dist[],boolean visited[],int start,int calc[][],int j){

int s=start-1,min\_d=9999,min\_v=s;

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

if(cost[s][i]!=-1 && dist[s]+cost[s][i]<dist[i] && !visited[i]){

dist[i]=dist[s]+cost[s][i];

if(min\_d>dist[i]){

min\_d=dist[i];

min\_v=i+1;

}

}

}

calc[j][0]=start;

for(int i=1;i<dist.length+1;i++)

calc[j][i]=dist[i-1];

visited[s]=true;

if(isVisited(visited))

return;

else

shortest(cost,dist,visited,min\_v,calc,++j);

}

static boolean isVisited(boolean a[]){

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

if(a[i]==false)

return false;

return true;

}

public static void ptrPath(Path path[]){

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

for(int j=0;j<path.length-1;j++){

if(path[j].d<path[j+1].d){

Path pt = path[j];

path[j]=path[j+1];

path[j+1]=pt;

}

}

}

for(int i=path.length-1;i>=0;i--){

if(i!=0)

System.out.print(path[i].v+" - ");

else

System.out.print(path[i].v);

}

System.out.println("\n");

}

static void print(int a[]){

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

System.out.print(a[i]+" ");

System.out.println("");

}

}

class Path{

int d,v;

Path(int d,int v)

{

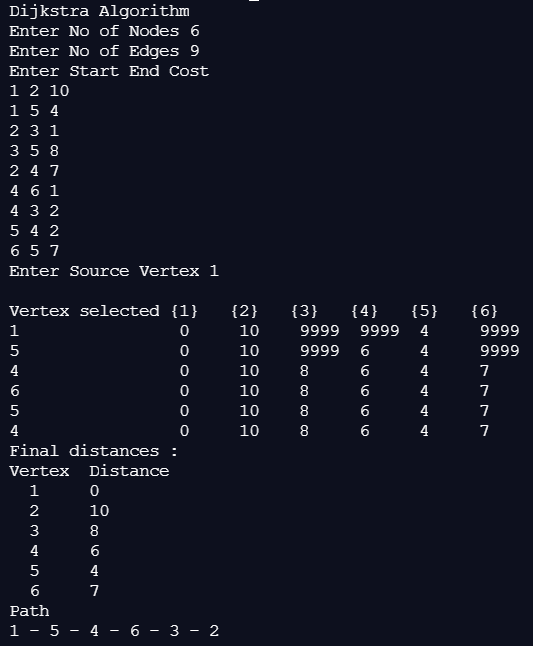
this.d=d;

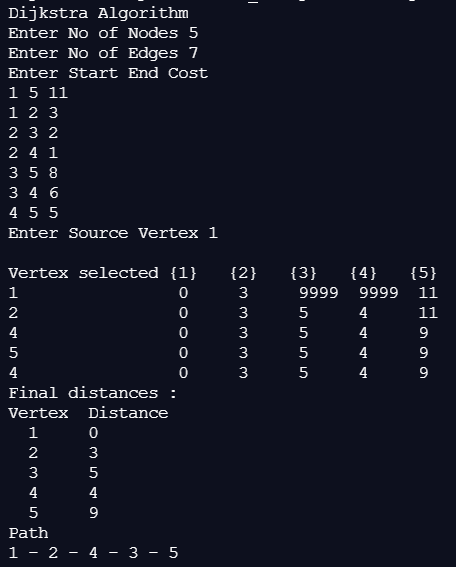
this.v=v;

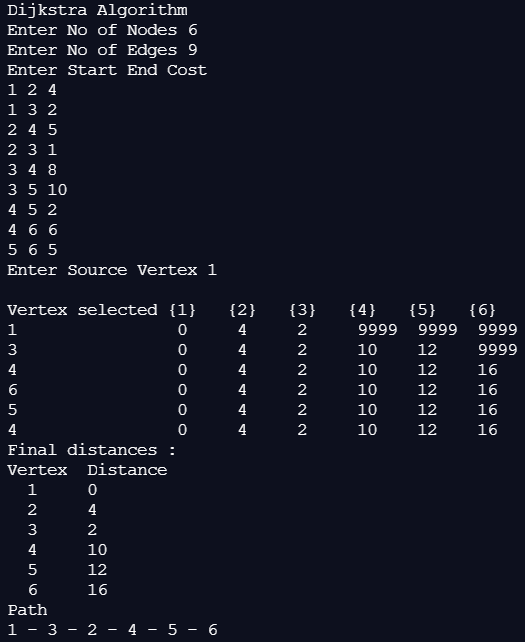
}

}

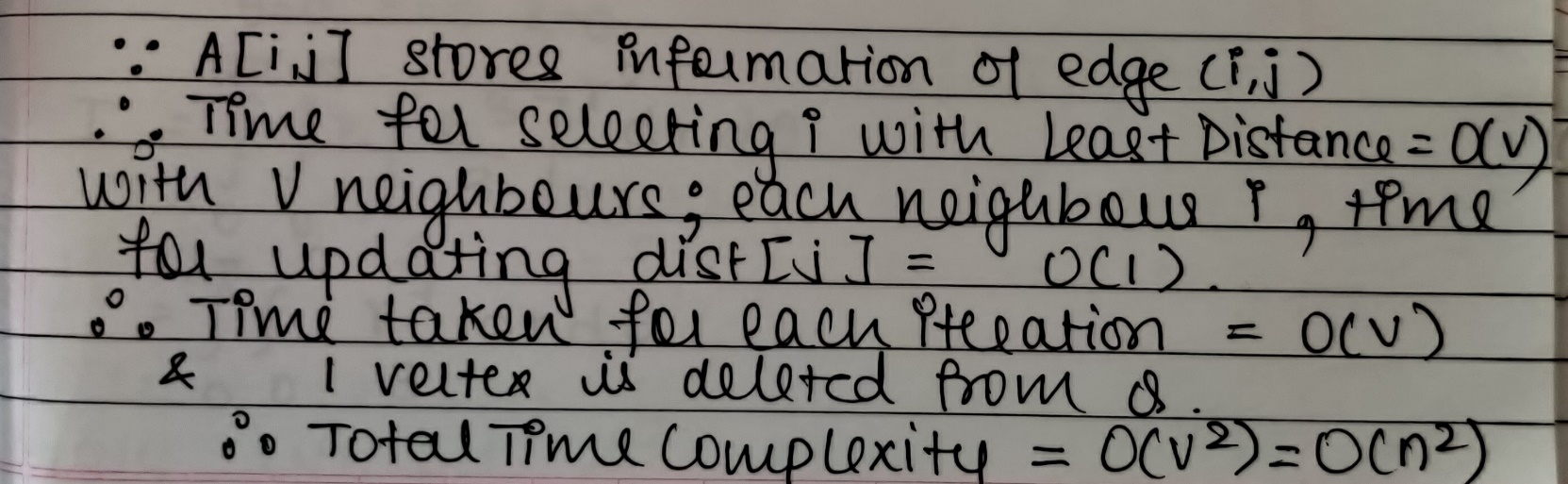
**Output**







**Time Complexity for single source shortest path**



Worst Case : O(V2)

Best Case : O(V)

**CONCLUSION:**

We successfully implemented the concept of Dijkstra’s Algorithm for finding Single Source Shortest Path in a given graph in Java ; thus obtaining correct necessary output.