

| **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:**

| CO 2 | Describe various algorithm design strategies to solve different problems and analyse Complexity. |
| --- | --- |

**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

**Topic: GREEDY METHOD**

**Theory:** The greedy method suggests that one can devise an algorithm that work in stages, considering one input at a time. At each stage, a decision is made regarding whether a particular input is in an optimal solution. This is done by considering the inputs in an order determined by some selection procedure. If the inclusion of the next input into the partially constructed optimal solution will result in an infeasible solution, then this input is not added to the partial solution. Otherwise, it is added. The selection procedure itself is based on some optimization measures may be plausible for a given problem. Most of these, however, will result in algorithms that generate suboptimal solutions. This version of the greedy technique is called the **subset paradigm**.

**Control Abstraction**:

SolType Greedy (Type s [], int n)

// a[1:n] contains the n inputs.

{SolType solution = EMPTY;

// Initialize the solution.

For (int i=1; I<=n; i++) {

Type x = Select (a);

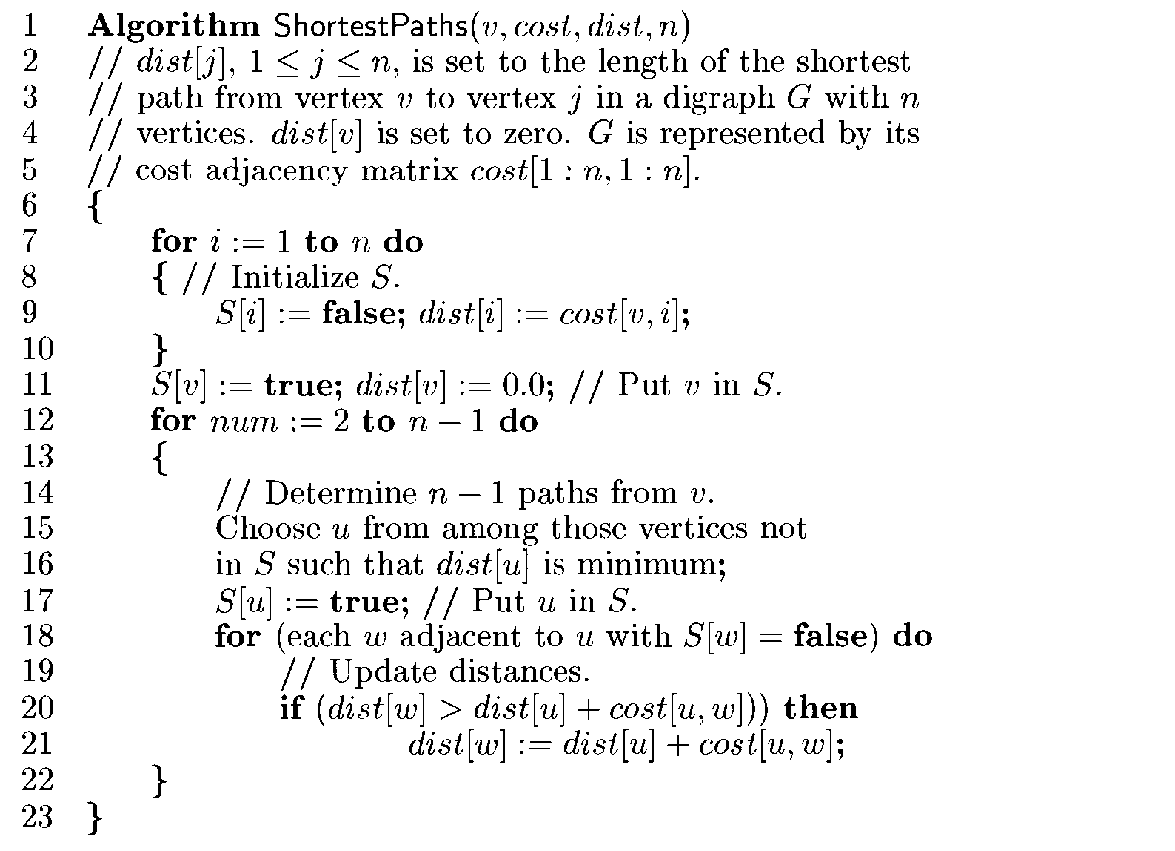
If Feasible (solution, x)

Solution = Union (solution, x) ;

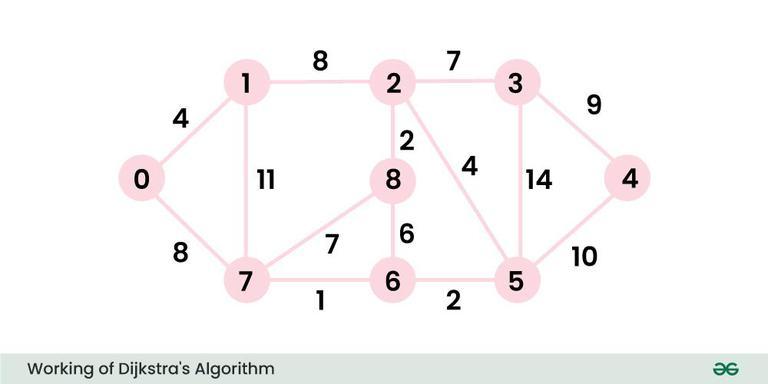
}

return solution;

}

**Algorithm**: 

**Example Graph:**



**Solution:**

import java.util.\*;

class ShortestPath {

// A utility function to find the vertex with minimum distance value,

// from the set of vertices not yet included in shortest path tree

static final int V = 9;

int minDistance(int dist[], Boolean sptSet[])

{

// Initialize min value

int min = Integer.MAX\_VALUE, min\_index = -1;

for (int v = 0; v < V; v++)

if (sptSet[v] == false && dist[v] <= min) {

min = dist[v];

min\_index = v;

}

return min\_index;

}

void printSolution(int dist[])

{

System.out.println("Vertex \t\t Distance from Source");

for (int i = 0; i < V; i++)

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

}

void dijkstra(int graph[][], int src)

{

int dist[] = new int[V];

Boolean sptSet[] = new Boolean[V];

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

dist[i] = Integer.MAX\_VALUE;

sptSet[i] = false;

}

dist[src] = 0;

for (int count = 0; count < V - 1; count++) {

int u = minDistance(dist, sptSet);

sptSet[u] = true;

for (int v = 0; v < V; v++)

if (!sptSet[v] && graph[u][v] != 0 && dist[u] !=

Integer.MAX\_VALUE && dist[u] + graph[u][v] < dist[v])

dist[v] = dist[u] + graph[u][v];

}

printSolution(dist);

}

public static void main(String[] args)

{

// System.out.println("Enter the vertices of the graph");

// Scanner sc=new Scanner(System.in);

// int v=sc.nextInt();

// int graph[][] = new int[v][v];

// int value;

// System.out.println("enter the weights of the graph");

// for (int i=0;i<v;i++){

// for(int j=0;j<v;j++){

// value=sc.nextInt();

// graph[i][j]=value;

// }

// Let us create the example graph discussed above \*/

int graph[][] = new int[][] { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },

{ 4, 0, 8, 0, 0, 0, 0, 11, 0 },

{ 0, 8, 0, 7, 0, 4, 0, 0, 2 },

{ 0, 0, 7, 0, 9, 14, 0, 0, 0 },

{ 0, 0, 0, 9, 0, 10, 0, 0, 0 },

{ 0, 0, 4, 14, 10, 0, 2, 0, 0 },

{ 0, 0, 0, 0, 0, 2, 0, 1, 6 },

{ 8, 11, 0, 0, 0, 0, 1, 0, 7 },

{ 0, 0, 2, 0, 0, 0, 6, 7, 0 } };

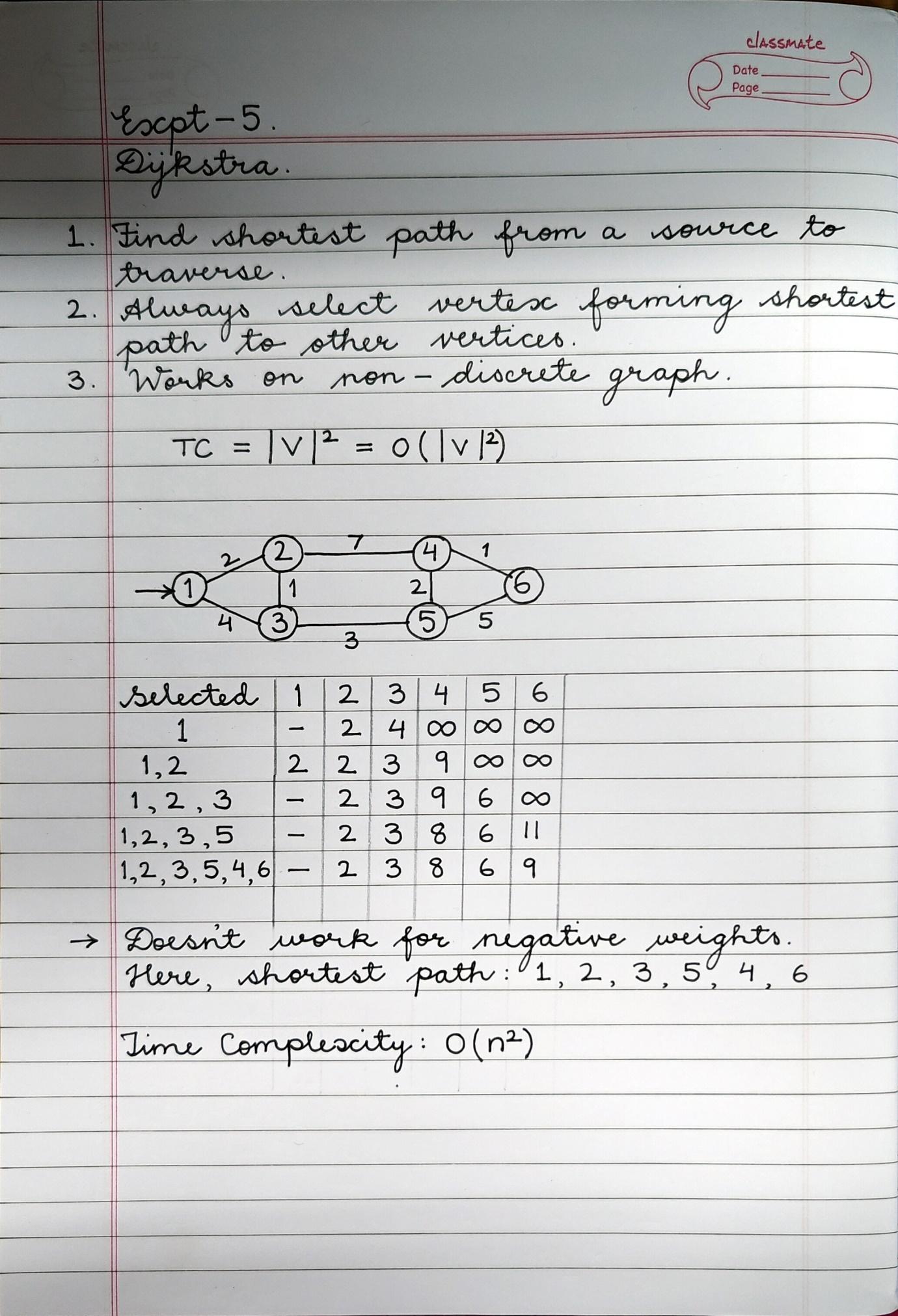
ShortestPath t = new ShortestPath();

t.dijkstra(graph, 0);

}

}  
  


**Time Complexity for single source shortest path**



**Conclusion:**

Through this experiment we learnt about find the single source shortest path in a graph. We

used Dijkstra’s algorithm to find the shortest path in our graph.