Batch:	**B2**	**Roll No.:**	**16010121110**

Experiment No.__4_____

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

Signature of the Staff In-charge with date

Title: Implementation of Single 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.Rajasekaran," Fundamentals of computer algorithm", University Press**
2. **T.H.Cormen ,C.E.Leiserson,R.L.Rivest and C.Stein," Introduction to algorithms",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 a solutioning method and proves how the greedy solution is better one.

Though greedy algorithms do not guarantee optimal solutions, 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 that 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**.

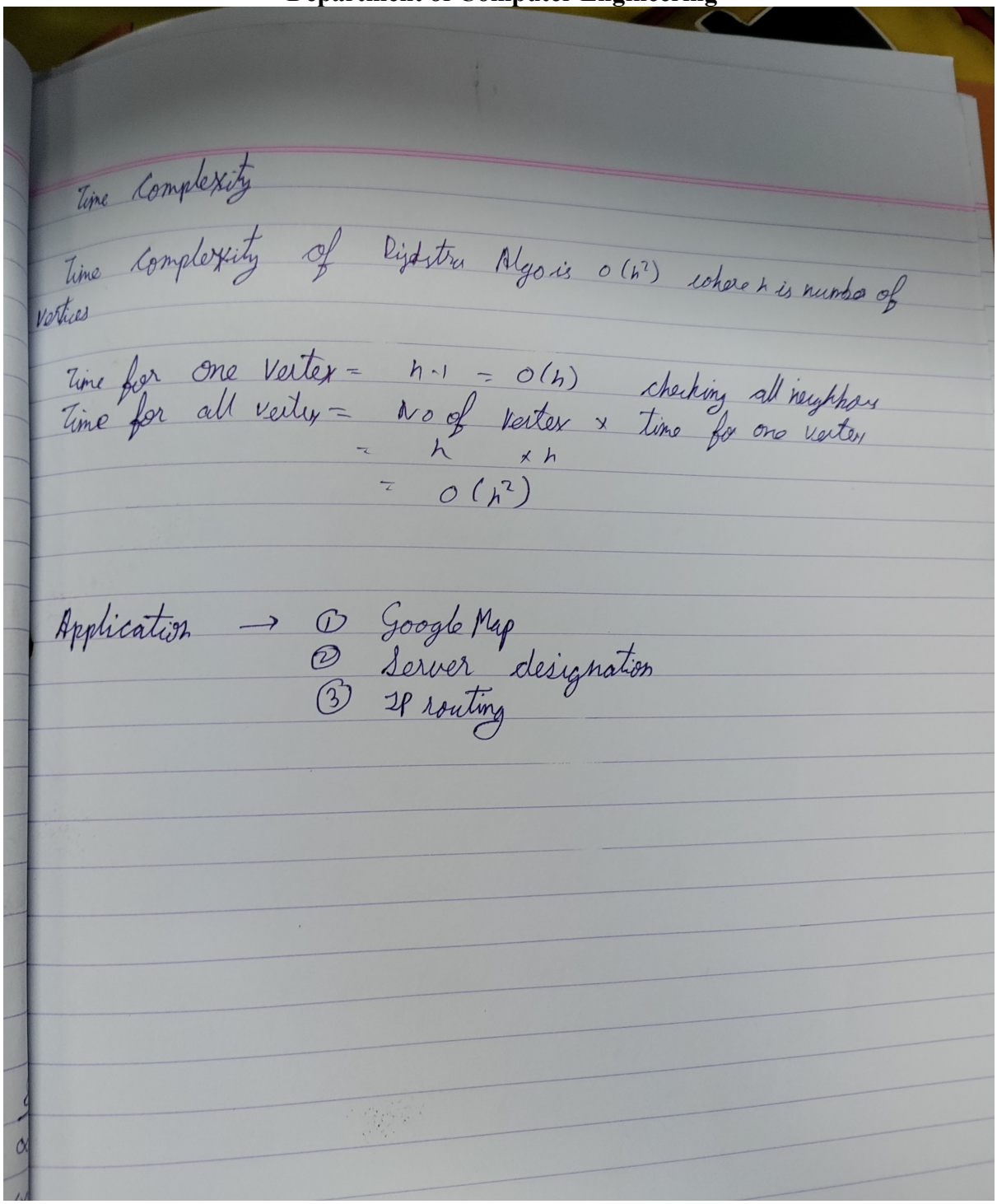
Control Abstraction:

```
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:

```
1   Algorithm ShortestPaths(v, cost, dist, n)
2   // dist[j], 1 ≤ j ≤ n, is set to the length of the shortest
3   // path from vertex v to vertex j in a digraph G with n
4   // vertices. dist[v] is set to zero. G is represented by its
5   // cost adjacency matrix cost[1 : n, 1 : n].
6   {
7       for i := 1 to n do
8       { // Initialize S.
9           S[i] := false; dist[i] := cost[v, i];
10      }
11      S[v] := true; dist[v] := 0.0; // Put v in S.
12      for num := 2 to n − 1 do
13      {
14          // Determine n − 1 paths from v.
15          Choose u from among those vertices not
16          in S such that dist[u] is minimum;
17          S[u] := true; // Put u in S.
18          for (each w adjacent to u with S[w] = false) do
19              // Update distances.
20              if (dist[w] > dist[u] + cost[u, w])) then
21                  dist[w] := dist[u] + cost[u, w];
22      }
23  }
```

Example Graph:



Time Complexity

Time Complexity of Dijkstra Algo is $O(n^2)$ where n is number of vertices

Time for one vertex = $n \cdot 1 = O(n)$ checking all neighbors
Time for all vertex = No of vertex x time for one vertex
$= n \times n$
$= O(n^2)$

Application → ① Google Map
② Server designation
③ IP routing

⑧ Single Source Shortest Path

finding path from one source to all possible destinations

step 1. find paths to adjusant vertexes from chosen vertex
step 2. If path length is smaller, update table

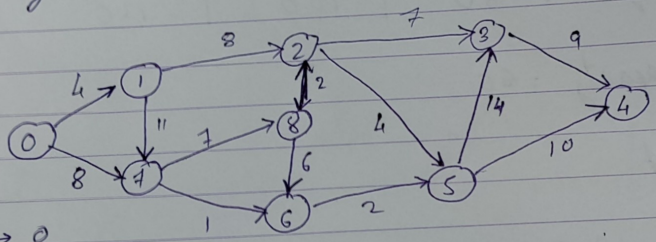
$d(j,i) = d(i) + d(i,j)$

cost to adjusant vertex ↑ cost to chosen vertex ↑ path length

step 3. Repeat 1 & 2 till all vertices are chosen
step 4. Backtrack to find shortest path.

Time Complexity $= O(n^2)$

Example



Source given → 0

Node / distance	0	1	2	3	4	5	6	7	
	0	4	∞	∞	∞	∞	∞	8	
ⓐ	0	0	4	∞	∞	∞	∞	8	
ⓑ	1	0	4	12	∞	∞	∞	8	
ⓒ	7	0	4	12	∞	∞	9	8	
ⓓ	6	0	4	12	∞	∞	11	9	8
ⓔ	5	0	4	12	25	21	11	9	8
ⓕ	2	0	4	12	19	21	11	9	8
ⓖ	8	0	4	12	19	21	11	9	8
ⓗ	3	0	4	12	19	21	11	9	8
ⓘ	4	0	4	12	19	21	11	9	8
S.P.		0	01	012	07653	07653	0765	076	07

Solution:

```
import java.util.*;
public class main
{
	public static void main(String[] args) {
		System.out.println("Hello World");
		int[][] adjustancy={ { 1000, 4, 1000, 1000, 1000, 1000, 1000, 8, 1000 },
	 { 4, 1000, 8, 1000, 1000, 1000, 1000, 11, 1000 },
	 { 1000, 8, 1000, 7, 1000, 4, 1000, 1000, 2 },
	 { 1000, 1000, 7, 1000, 9, 14, 1000, 1000, 1000 },
	 { 1000, 1000, 1000, 9, 1000, 11000, 1000, 1000, 1000 },
	 { 1000, 1000, 4, 14, 10, 1000, 2, 1000, 1000 },
	 { 1000, 1000, 1000, 1000, 1000, 2, 1000, 1, 6 },
	 { 8, 11, 1000, 1000, 1000, 1000, 1, 1000, 7 },
	 { 1000, 1000, 2, 1000, 1000, 1000, 6, 7, 1000 } };

		int[] distance= new int[adjustancy[0].length];
			for(int i=0;i<=distance.length-1;i++){
		    distance[i]=1000;//initialize to Infinity
		}
		ArrayList<Integer> covered= new ArrayList<Integer>();
		path(0,0,adjustancy,distance,covered);
		for(int i=0;i<=distance.length-1;i++){
		    System.out.print(distance[i]+",");
		}
	}
	public static void path(int initialCost, int nodeId,int[][] adjustancy,int []
distance,ArrayList<Integer> covered){
	      covered.add(nodeId);
	        System.out.println("visited="+nodeId);

	    for(int i=0;i<adjustancy[nodeId].length;i++){
		if(i==nodeId){
		    continue; // dont check for same element
		}
		int cost=initialCost+adjustancy[nodeId][i];
		System.out.println("cost="+cost);
		if(distance[i]>cost){
		    distance[i]=cost;
```

```
                System.out.println("updated "+cost);
            }


        }

        if(covered.size()==distance.length){
               System.out.println("all nodes covered");
            return;
        }
      int max=-1;
      for(int i=0;i<distance.length;i++){
          if(i==nodeId){
              continue;
          }
          if(!covered.contains(i)){
              if(max==-1){
                  max=i;
              }
              else

              if(distance[i]<distance[max] ){

                 max=i;
                  System.out.println("max is  "+max);

             }
      }
        }

        path(distance[max],max,adjustancy,distance,covered);
     return;
    }
}
```

Time Complexity for single source shortest path
O(n^2)

CONCLUSION: Thus we have understood how to implement SSSP algorithm using greedy technique. Single source shortest path algorithm is used to find the distance between one node and all other nodes.