K. J. Somaiya College of Engineering

(A Constituent College of Somaiya Vidyavihar University)

Batch: A1	**Roll No.:16010120015**
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.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:

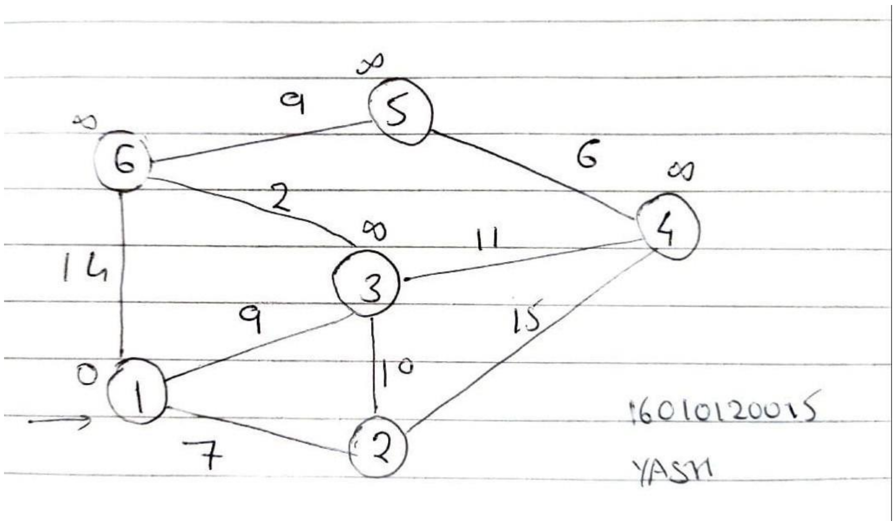
```
SolType Greedy (Type s [ ], int n)

// 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:

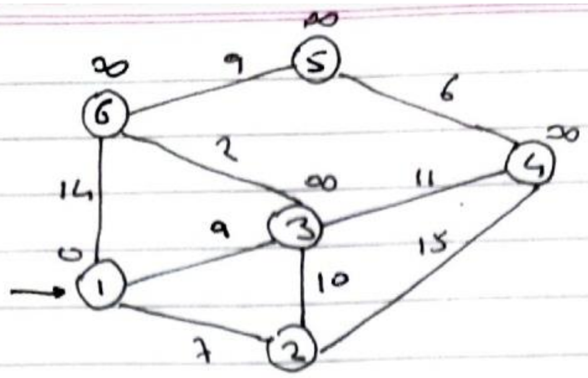
```
Algorithm ShortestPaths($v, cost, dist, n$)
// $dist[j]$, $1 \le j \le n$, is set to the length of the shortest
// path from vertex $v$ to vertex $j$ in a digraph $G$ with $n$
// vertices. $dist[v]$ is set to zero. $G$ is represented by its
// cost adjacency matrix $cost[1:n, 1:n]$.
{
    for $i := 1$ to $n$ do
    { // Initialize $S$.
        $S[i]$ := false; $dist[i] := cost[v, i]$;
    }
    $S[v]$ := true; $dist[v] := 0.0$; // Put $v$ in $S$.
    for $num := 2$ to $n - 1$ do
    {
        // Determine $n - 1$ paths from $v$.
        Choose $u$ from among those vertices not
        in $S$ such that $dist[u]$ is minimum;
        $S[u]$ := true; // Put $u$ in $S$.
        for (each $w$ adjacent to $u$ with $S[w]$ = false) do
            // Update distances.
            if ($dist[w] > dist[u] + cost[u, w]$)) then
                    $dist[w] := dist[u] + cost[u, w]$;
    }
}
```

Example Graph:

Solution:



Q

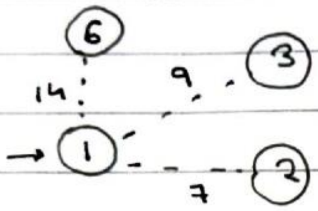


160101200015

→ Starting node = 1

Iteration 1:

N = {1}



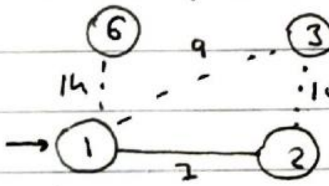
$\boxed{D_{12} = 7}$

$D_{13} = 9$

$D_{14} = 14$

Iteration 2:

N = {1, 2}

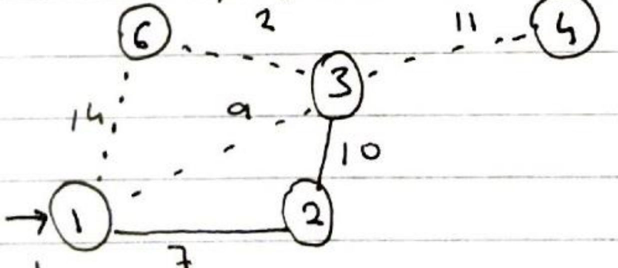


$\boxed{D_{23} = ~~3~~\ 10}$

$D_{24} = 15$

Iteration 3:

N = {1, 2, 3}

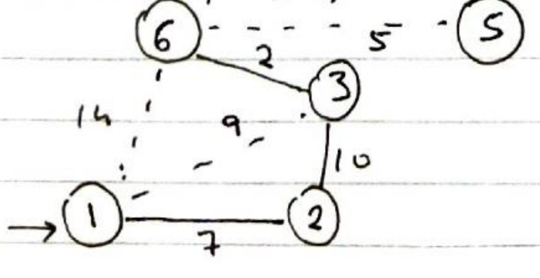


$D_{34} = 11$

$\boxed{D_{36} = 2}$

Iteration 4:

N = {1, 2, 3, 6}



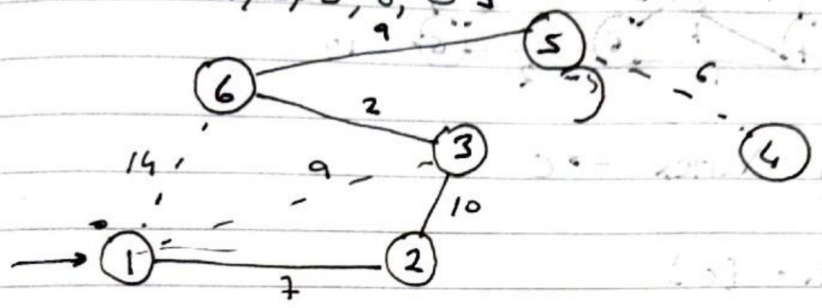
$\boxed{D_{65} = 9}$

D

Iteration 5 :

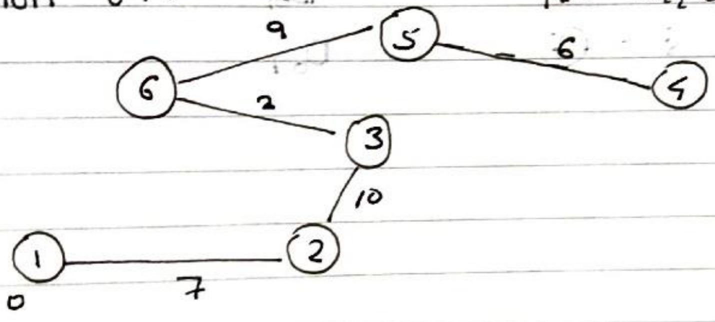
N = {1, 2, 3, 6, 5}

$\boxed{D_{54} = 6}$



Iteration 6 :

N = {1, 2, 3, 6, 5, 4}

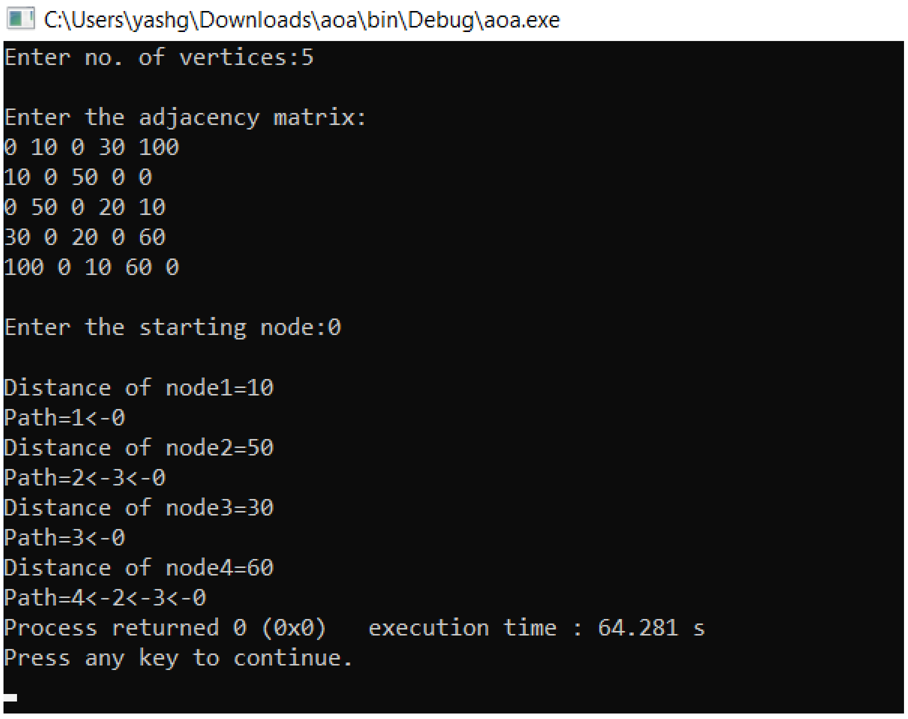


Code:

```c
#include<stdio.h>
#define INFINITY 9999
#define MAX 10

void dijkstra(int G[MAX][MAX],int n,int startnode);

int main()
{
    int G[MAX][MAX],i,j,n,u;
    printf("Enter no. of vertices:");
    scanf("%d",&n);
    printf("\nEnter the adjacency matrix:\n");

    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
            scanf("%d",&G[i][j]);

    printf("\nEnter the starting node:");
    scanf("%d",&u);
    dijkstra(G,n,u);

    return 0;
```

```c
}

void dijkstraa(int G[MAX][MAX],int n,int startnode)
{

    int cost[MAX][MAX],distance[MAX],pred[MAX];
    int visited[MAX],count,mindistance,nextnode,i,j;

    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
            if(G[i][j]==0)
                cost[i][j]=INFINITY;
            else
                cost[i][j]=G[i][j];
    for(i=0;i<n;i++)
    {
        distance[i]=cost[startnode][i];
        pred[i]=startnode;
        visited[i]=0;
    }

    distance[startnode]=0;
    visited[startnode]=1;
    count=1;

    while(count<n-1)
    {
        mindistance=INFINITY;

        for(i=0;i<n;i++)
            if(distance[i]<mindistance&&!visited[i])
            {
                mindistance=distance[i];
                nextnode=i;
            }

            visited[nextnode]=1;
            for(i=0;i<n;i++)
                if(!visited[i])
                    if(mindistance+cost[nextnode][i]<distance[i])
                    {
                                                   distance[i]=mindistance+cost[nextno
                                                   de][i]; pred[i]=nextnode;
                                               }
                    count++;
                }


    for(i=0;i<n;i++)
        if(i!=startnode)
        {
            printf("\nDistance of node%d=%d",i,distance[i]);
            printf("\nPath=%d",i);

            j=i;
            do
            {
                j=pred[j];
                printf("<-%d",j);
            }while(j!=startnode);
      }
}
    count=1;

    while(count<n-1)
    {
        mindistance=INFINITY;

        for(i=0;i<n;i++)
            if(distance[i]<mindistance&&!visited[i])
            {
                mindistance=distance[i];
                nextnode=i;
            }
            visited[nextnode]=1;
            for(i=0;i<n;i++)
                                    if(!visited[i])
                    if(mindistance+cost[nextnode][i]<distance[i])
                                        {
                        distance[i]=mindistance+cost[nextnode][i];
                        pred[i]=nextnode;
                                        }
        count++;
        }

            for(i=0;i<n
        ;i++)
        if(i!=startnode)
                {
            printf("\nDistance of node%d=%d",i,distance[i]);
            printf("\nPath=%d",i);

            j=i;
            do
                        {
                j=pred[j];
                printf("<-%d",j);
            }while(j!=startnode);
            }
```

Output:


C:\Users\yashg\Downloads\aoa\bin\Debug\aoa.exe

```
Enter no. of vertices:5

Enter the adjacency matrix:
0 10 0 30 100
10 0 50 0 0
0 50 0 20 10
30 0 20 0 60
100 0 10 60 0

Enter the starting node:0

Distance of node1=10
Path=1<-0
Distance of node2=50
Path=2<-3<-0
Distance of node3=30
Path=3<-0
Distance of node4=60
Path=4<-2<-3<-0
Process returned 0 (0x0)   execution time : 64.281 s
Press any key to continue.
```

Time Complexity for single source shortest path

```
for(int k = 1; k <= n; k++){
    for(int i = 1; i <= n; i++){
        for(int j = 1; j <= n; j++){
            dist[i][j] = min( dist[i][j], dist[i][k] + dist[k][j] );
        }
    }
}
```

Time Complexity for Single Source Shortest Path: The time complexity for Dijkstra’s algorithm is O(V2) where “V” is the number of vertices of the graph. This is due to the fact that we have used two loops nested together and each of them iterates over all the nodes.

The space complexity is O(E) where “E” is the number of edges of the graph because we are appending path with the edges.

Conclusion:

we discussed Dijkstra’s algorithm, a popular shortest path algorithm on the graph.

i.e : an algorithm that is used for finding the shortest distance, or path, from starting node to target node in a weighted graph .

- Dijkstra's algorithm makes use of weights of the edges for finding the path that minimizes the total distance from other nodes. This algorithm is also known as the single-source shortest path algorithm.
- It uses a priority queue to greedily choose the nearest node that has not been visited yet and executes the relaxation process on all of its edges.