K. J. Somaiya College of Engineering, Mumbai-77
(A Constituent College of Somaiya Vidyavihar University)
Department of Computer Engineering

Batch: A1	**Roll No.: 16010123012**
Experiment No. 5	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

Title: Study, Implementation and Analysis of Single Source Shortest Path

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:

```
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  }
```

Code:

```cpp
#include <bits/stdc++.h>
#define endl '\n'
using namespace std;

int minDistance(vector<int> &dist, vector<bool> &S, int n)
{
    int min = INT_MAX, minIndex = -1;
    for (int i = 0; i < n; i++)
    {
        if (!S[i] && dist[i] < min)
        {
            min = dist[i];
            minIndex = i;
        }
    }
    return minIndex;
}

void dijkstra(vector<vector<int>> &W, int source, int n, vector<int> &dist)
{
    vector<bool> S(n, false);
    dist[source] = 0;
```

```
    for (int i = 0; i < n - 1; i++)
    {
        int u = minDistance(dist, S, n);
        if (u == -1)
        {
            break;
        }

        S[u] = true;
        for (int v = 0; v < n; v++)
        {
            if (!S[v] && W[u][v] != INT_MAX && dist[u] != INT_MAX &&
                dist[u] + W[u][v] < dist[v])
            {
                dist[v] = dist[u] + W[u][v];
            }
        }
    }
}

int main()
{
    int n;
    cout << "Enter the number of vertices: ";
    cin >> n;

    vector<vector<int>> W(n, vector<int>(n));
    cout << "Enter the adjacency matrix: " << endl;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            cin >> W[i][j];
        }
    }

    int source;
    cout << "Enter the source vertex: ";
    cin >> source;
    if (source < 0 || source >= n)
    {
        cout << "Invalid source vertex. Exiting...\n";
        return 1;
    }

    vector<int> dist(n, INT_MAX);
    dijkstra(W, source, n, dist);
```

```
    cout << "\nVertex\tDistance from Source" << endl;
    for (int i = 0; i < n; i++)
    {
        if (dist[i] == INT_MAX)
            cout << i << "\tNo Path" << endl;
        else
            cout << i + 1 << "\t" << dist[i] << endl;
    }
}
```

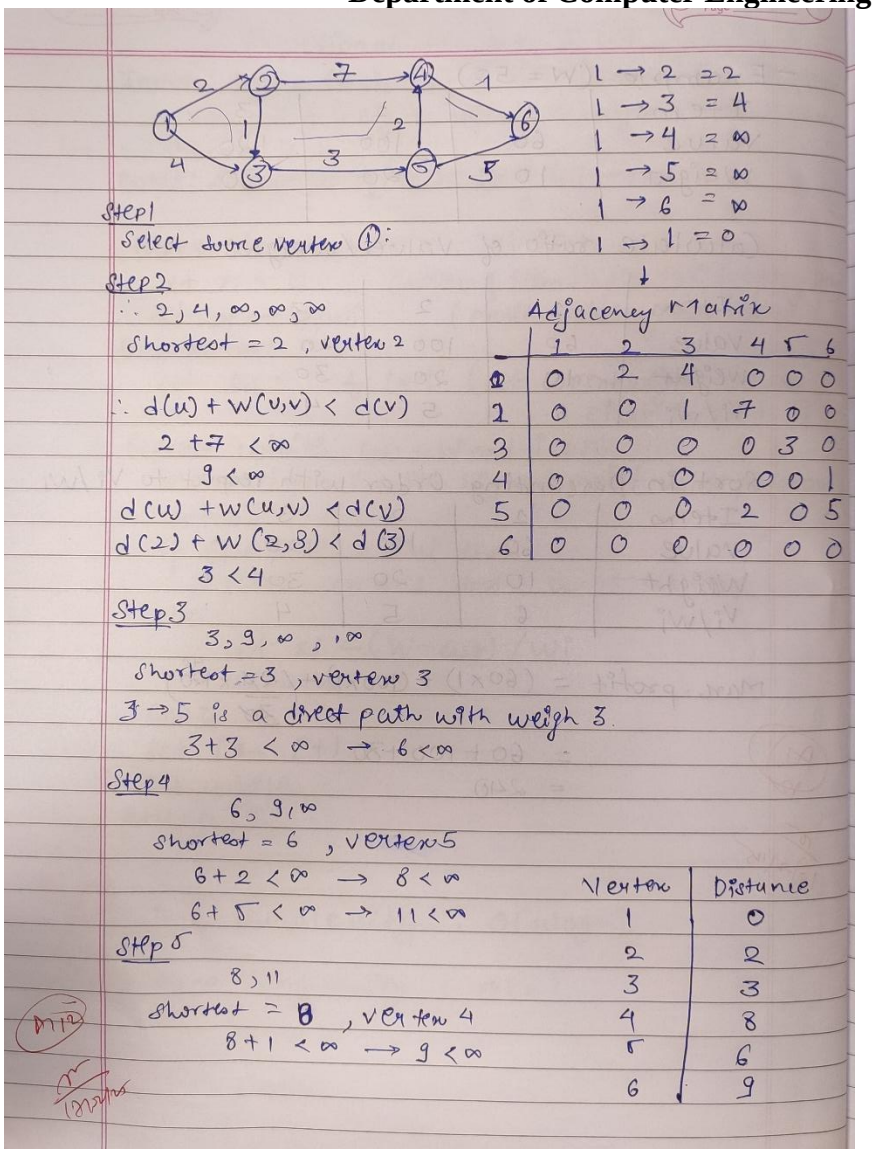
Output:

```
PS D:\KJSCE\BTech\SY\Sem IV\AOA\Code>
Code\" ; if ($?) { g++ dijastra.cpp -o dijastra } ; if ($?) { .\dijastra }
Enter the number of vertices: 6
Enter the adjacency matrix:
0 2 4 0 0 0
0 0 1 7 0 0
0 0 0 0 3 0
0 0 0 0 0 1
0 0 0 2 0 5
0 0 0 0 0 0
Enter the source vertex: 0

Vertex  Distance from Source
0       0
1       2
2       3
3       8
4       6
5       9
```

Time Complexity: $O(n^2)$
Space Complexity: $O(n^2)$

Example Graph:



Conclusion:

I have successfully implemented and analyzed the Single Source Shortest Path problem using Dijkstra's Algorithm. This experiment reinforced my understanding of the Greedy strategy, graph traversal, and algorithm complexity analysis. I observed that using an adjacency matrix results in $O(n^2)$ time complexity and space complexity, which can be optimized with a priority queue. This experiment enhanced my ability to apply greedy techniques to real-world problems.