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

Batch: B2	**Roll No.: 16010122151**
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**.

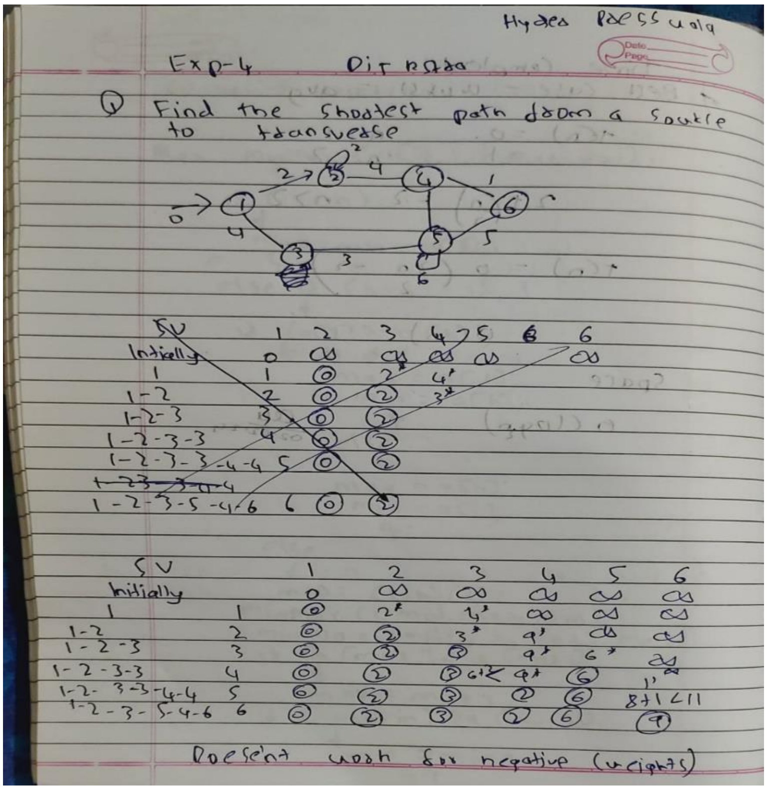
Control Abstraction:

```
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 ≤ j ≤ 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:



Algo - dijkstra (Graph, Source)
dist[source] = 0
Create empty set S
For each vertex in Graph
dist[v] = infinity
while SptSet does not include all vertices
u = vertex in Graph not in SptSet
with min dist[u].
add u to SptSet

for each vertex adjacent to u:
if (distance[u] + wt(u,v) < dist[v])
dist[v] = dist[u] + weight (u,v)

return dist[]

~~Best case: O(V+E)~~

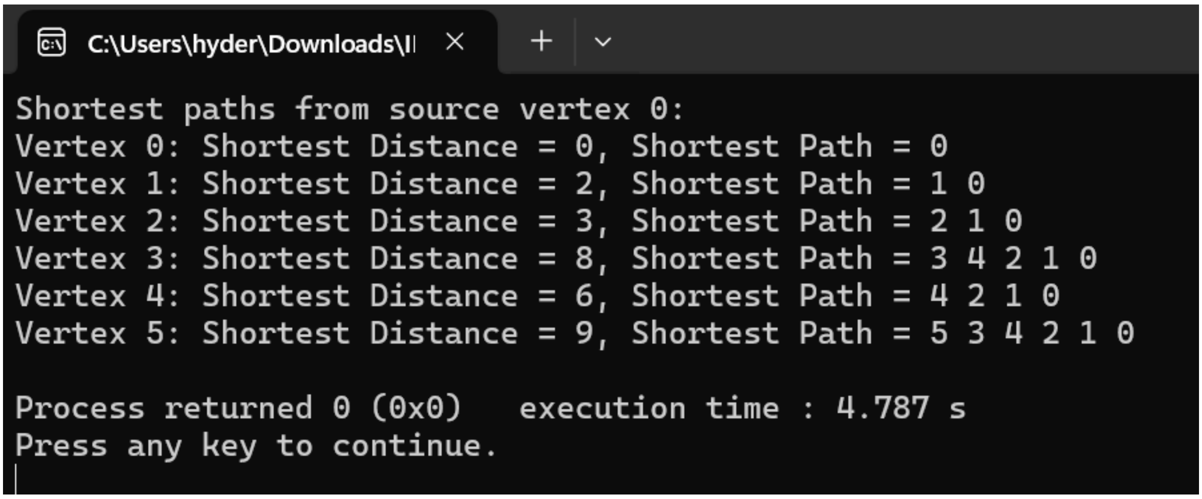
15.03.24

CODE:-

```
#include <stdio.h>
#include <stdbool.h>
#include <limits.h>

#define V 6 // Number of vertices

void dijkstra_shortest_path(int graph[V][V], int source) {
    int dist[V]; // The output array. dist[i] will hold the shortest distance from
source to i
    bool S[V]; // S[i] will be true if vertex i is included in shortest path tree
or shortest distance from source to i is finalized
    int pred[V]; // Array to store predecessors in shortest path

    // Initialize all distances as infinite and S[] as false
    for (int i = 0; i < V; i++) {
        dist[i] = INT_MAX;
        S[i] = false;
        pred[i] = -1;
    }

    // Distance of source vertex from itself is always 0
    dist[source] = 0;

    // Find shortest path for all vertices
    for (int count = 0; count < V - 1; count++) {
        // Pick the minimum distance vertex from the set of vertices not yet
processed.
        // u is always equal to source in the first iteration.
        int u, min_dist = INT_MAX;
        for (int v = 0; v < V; v++) {
            if (!S[v] && dist[v] < min_dist) {
                u = v;
                min_dist = dist[v];
            }
        }

        // Mark the picked vertex as processed
        S[u] = true;

        // Update dist value of the adjacent vertices of the picked vertex.
        for (int v = 0; v < V; v++) {
            if (!S[v] && graph[u][v] && dist[u] + graph[u][v] < dist[v]) {
                dist[v] = dist[u] + graph[u][v];
                pred[v] = u;
            }
```

```c
        }
    }

    // Print the shortest paths
    printf("Shortest paths from source vertex %d:\n", source);
    for (int i = 0; i < V; i++) {
        printf("Vertex %d: Shortest Distance = %d, Shortest Path = ", i, dist[i]);
        int current = i;
        while (current != -1) {
            printf("%d ", current);
            current = pred[current];
        }
        printf("\n");
    }
}

int main() {
    // Example adjacency matrix representation of a graph
    int graph[V][V] = {
        {0, 2, 4, 0, 0, 0},
        {2, 0, 1, 7, 0, 0},
        {4, 1, 0, 0, 3, 0},
        {0, 7, 0, 0, 2, 1},
        {0, 0, 3, 2, 0, 5},
        {0, 0, 0, 1, 5, 0}
    };

    int source = 0; // Source vertex

    dijkstra_shortest_path(graph, source);

    return 0;
}
```

OUTPUT:-



Conclusion:
We successfully have calculated the single source shortest path on paper and via code and obtained the same correct output. The time complexity of this algorithm is O(n^2).