**Batch: B3 Roll No.: 121**

**Experiment No.\_\_\_4\_\_\_**

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

**Signature of the Staff In-charge with date**

|  |
| --- |
| **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.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**
6. **https://www.scaler.com/topics/data-structures/dijkstra-algorithm/**
7. **https://www.gatevidyalay.com/dijkstras-algorithm-shortest-path-algorithm/**

**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);

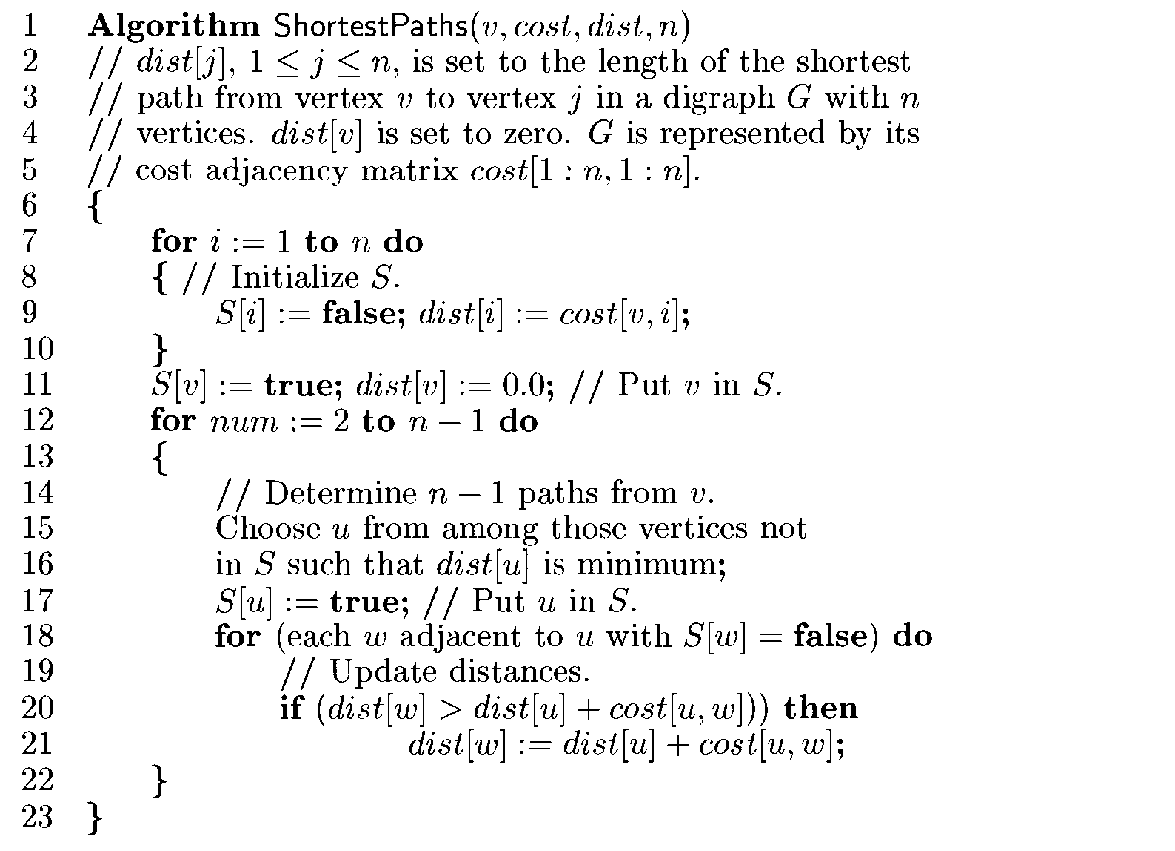
If Feasible (solution, x)

Solution = Union (solution, x) ;

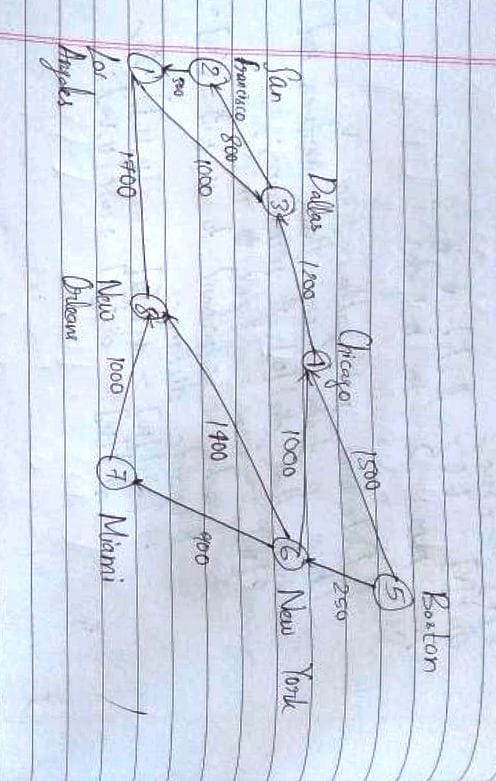
}

return solution;

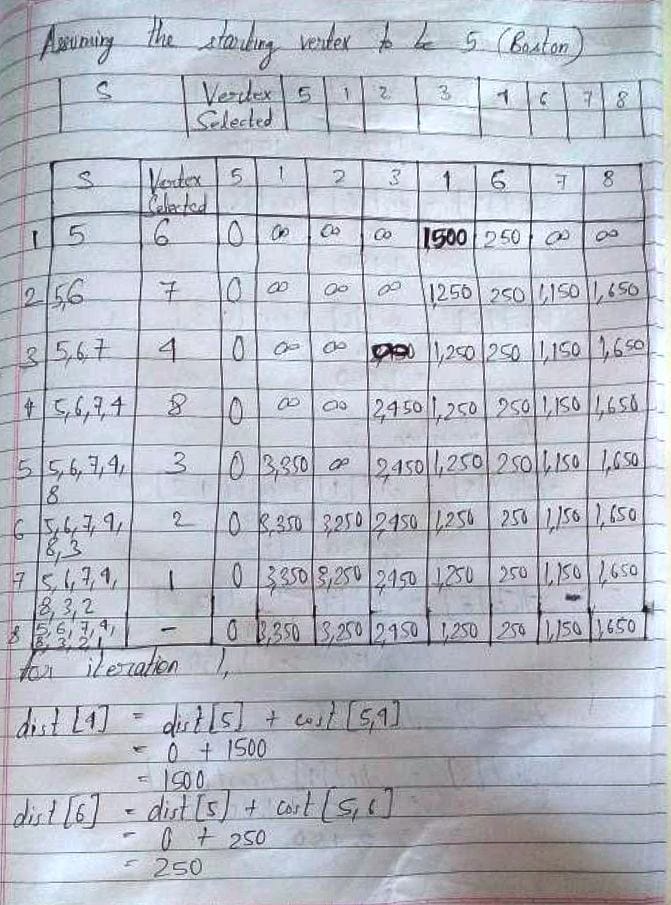
}

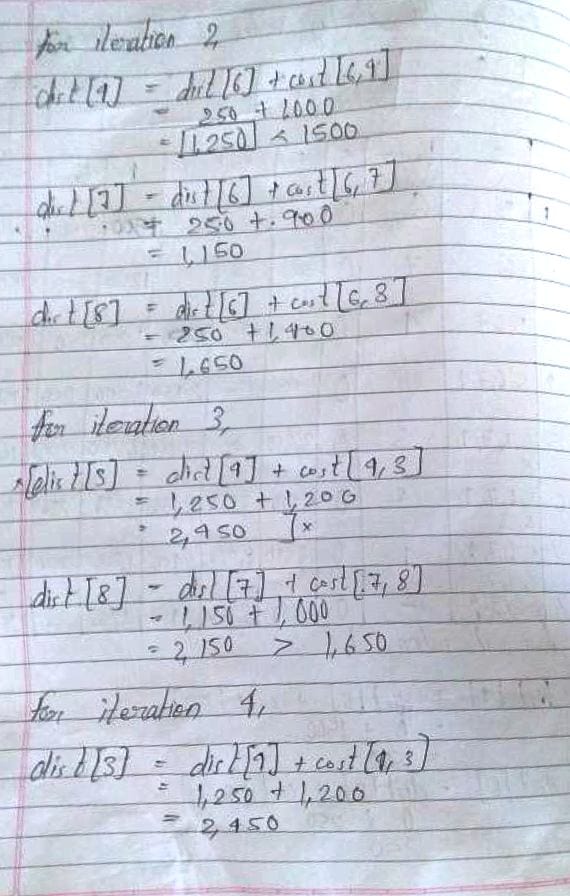
**Algorithm**: 

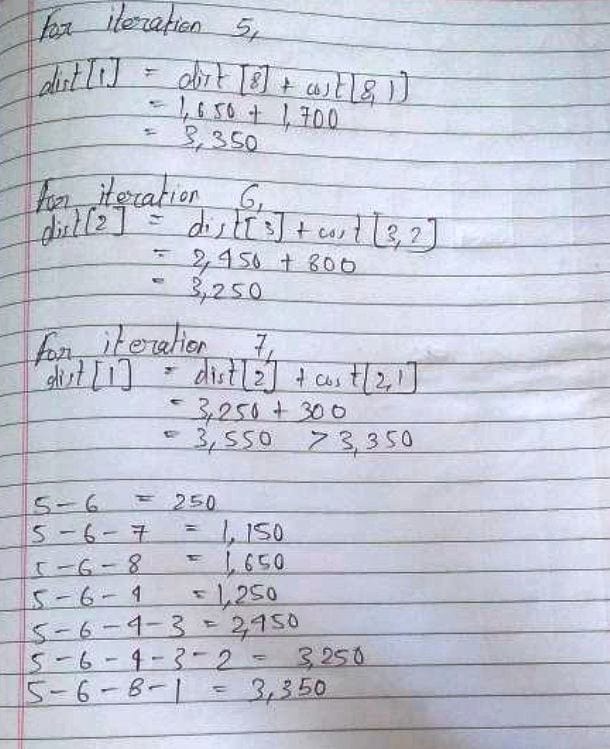
**Example Graph:**



**Solution:**

****

****

****

**Time Complexity for single source shortest path**

**Case 1:**

This case is valid when:

1. The given graph G is represented as an adjacency matrix
2. Priority queue Q is represented as an unordered list

Here,

1. A[i,j] stores the information about edge (i,j).
2. Time taken for selecting i with the smallest distance is O(V).
3. For each neighbour of i, time taken for updating dist[j] is O(1) and there will be maximum V neighbours.
4. Time taken for each iteration of the loop is O(V) and one vertex is deleted from Q.
5. Thus, total time complexity becomes O(V2).

**Case 2:**

This case is valid when:

1. The given graph G is represented as an adjacency list.
2. Priority queue Q is represented as a binary heap.

Here,

1. With adjacency list representation, all vertices of the graph can be traversed using BFS in O(V+E) time.
2. In min heap, operations like extract-min and decrease-key value takes O(logV) time.
3. So, overall time complexity becomes O(E+V)×O(logV) which is O((E+V)logV)

**Conclusion:**

Thus, in this experiment, the concept of Single Source Shortest Path (Dijkstra's Algorithm) has been learnt and implemented. The implementation performed belongs to Case-1, which has time complexity of O(V2). Although Dijkstra's algorithm proves to be simple to understand, it may not necessarily give an optimal solution. Further, it cannot be used when edges have negative values. So, practically, it does have applications, but the applications are limited.