

K. J. Somaiya College of Engineering, Mumbai-77

(Autonomous College Affiliated to University of Mumbai)

Batch: B1

Roll No.: 1711072

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:

Sr. No	Objective
CO 1	Compare and demonstrate the efficiency of algorithms using asymptotic complexity notations.
CO 2	Analyze and solve problems for divide and conquer strategy, greedy method, dynamic programming approach and backtracking and branch & bound policies.
CO 3	Analyze and solve problems for different string matching algorithms.

Books/ Journals/ Websites referred:

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 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.

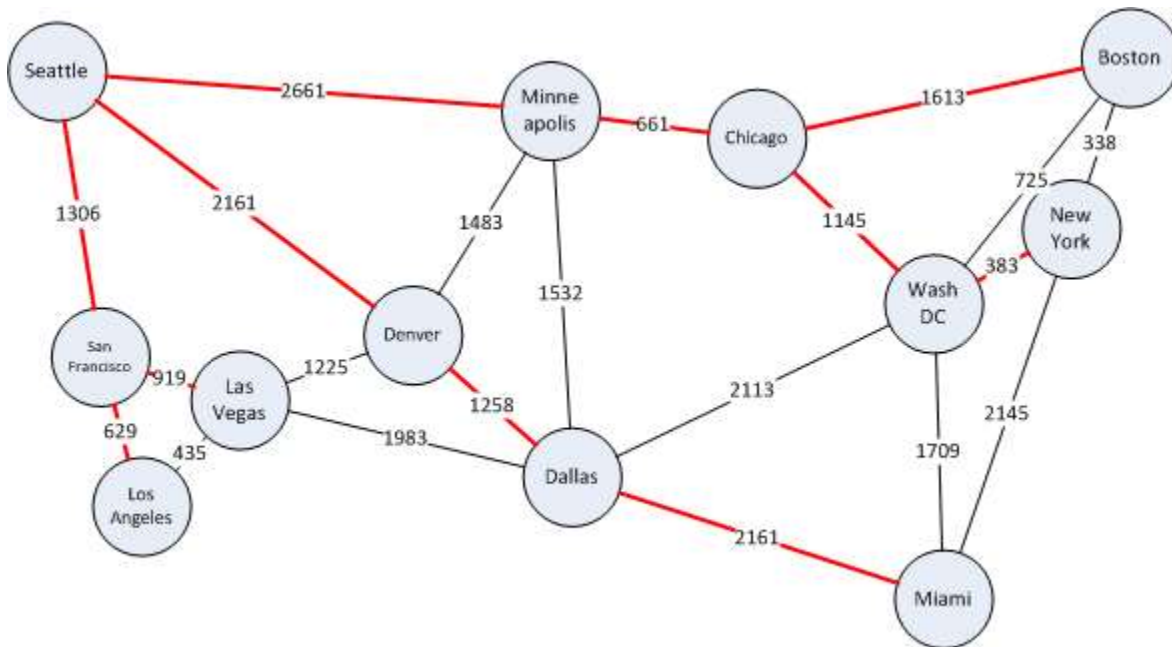
New Concepts to be learned:

Application of algorithmic design strategy to any problem, Greedy method of problem solving
Vs other methods of problem solving, optimality of the solution

Algorithm:

```
1  Algorithm ShortestPaths(v, cost, dist, n)
2  // dist[j],  $1 \leq j \leq n$ , 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:



Solution (Implementation in Python):

```
class Graph():
    def __init__(self, vertices):
        self.V=vertices
        self.graph=[[0 for col in range(vertices)] for row in range(vertices)]

    def printSolution(self, dist):
        print ("Vertex \tDistance from Source")
        for node in range(self.V):
            print (node, "\t\t\t", dist[node])

    def minDistance(self, dist, SPT):
        min = float('inf')
        min_index=0
        for v in range(self.V):
            if dist[v] < min and SPT[v] == False:
                min = dist[v]
                min_index = v

        return min_index

    def dijkstra(self,source):
        dist= [float('inf')] * self.V
```

```

dist[source]=0
SPT=[False] * self.V

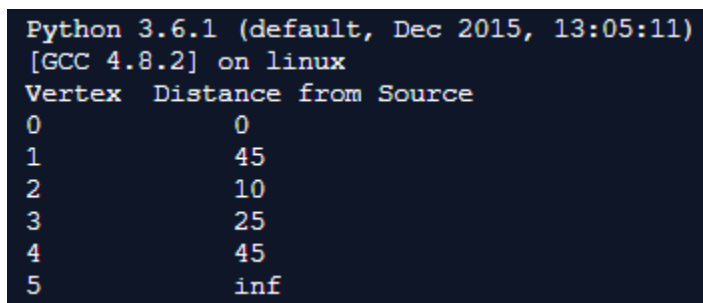
for _ in range(self.V):
    u=self.minDistance(dist, SPT)
    SPT[u]=True
    for v in range(self.V):
        if self.graph[u][v] > 0 and SPT[v] == False and dist[v] > dist[u] +
self.graph[u][v]:
            dist[v] = dist[u] + self.graph[u][v]
    self.printSolution(dist)

g=Graph(6)
g.graph=[[0,50,10,0,45,0],[0,0,15,0,10,0],[20,0,0,15,0,0],[0,20,0,0,35,0],[0,0,0,
30,0,0],[0,0,0,3,0,0]]

g.dijkstra(0)

```

Screenshot:



```

Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
Vertex  Distance from Source
0          0
1         45
2         10
3         25
4         45
5        inf

```

Time Complexity for single source shortest path:

The main logic of single source shortest path is defined in the function `dijkstra(self, source)`. The main logic consists of nested for loops which both run for total number of vertices. If number of vertices is $|V|$, then the Time complexity of this algorithm is given by:

$$T(n)=O(|V|^2)$$

The complexity of the algorithm can be improved by using Fibonacci heaps, Binary heaps, adjacency list or Disjoint data structure. If number of edges is “E” and number of vertices is $|V|$, then the time complexity on using any these data structures will be approximately:

$$T(n)=O(E*\log(|V|))$$

CONCLUSION:

The program for Dijkstra’s shortest path algorithm was successfully implemented in Python and works for all test cases.