CS345

Algorithms -II

Indian Institute of Technology, Kanpur

Anshit Arya (190158), Saket Dhakar (190739)

Assignment 2

Question 1

We have already seen Bellman Ford algorithm which solves single source shortest paths (SSSP) problem in O(mn) time for directed graphs with potentially negative weighted edges but no negative cycle..................

Solution

We have to consider the following
1. There can be negative weight edge (but no negative weight cycle).
2. The vertices are labelled form 1 to n.(1, 2, ..., n)

Ans a

$\rightarrow$ δ^0(i,j) - there is no 0 vertex so if there is edge then value will be w_{ij} otherwise value will be infinite. (where w_{ij} is the weight of the edge from i to j

$\rightarrow$ From the optimal sub-path property

δ^k(i,j) = min(δ^{k-1}(i,j) , δ^{k-1}(i,k) +δ^{k-1}(k,j))

$\Rightarrow$ When k = n , the smallest path between any two vertex will be p^n (i, j) which have weight δ^n (i, j) .

Algorithm (a) ::: All Pair Shortest Path length

Let G be the Adjacency List of the Directed Graph .
$n \equiv$ No. of nodes

$\rightarrow$ DP[][] will be the output matrix that will finally have the shortest distances between every pair of vertices

```
func APSP(G, n)
        DP[n+1][n+1] = G → Initialize the solution matrix same
                           as input graph matrix(i.e.
                           the initial values of shortest distances
                           are based on considering
                           no intermediate vertex. )
        for k=0 to n
           for i=0 to n
             for j=0 to n
               if DP[i][k] != inf and DP[k][j] != inf
                 DP[i][j] = min(DP[i][j] , DP[i][k] + DP[k][j] ) → If vertex k is on the
                                                   shortest path from i to j, then update
                                                   the value of DP[i][j]
               endif
             endfor
           endfor
        endfor
        Return DP
endfunc
```

Ans B

$\rightarrow$ now we get shortest path between 2 vertices
$\rightarrow$ In the k^{th} iteration if the value of DP array is changed we can say that k is in between the path from i to j
$\rightarrow$ We will now recursively check for vertices in between i to k and k to j
$\rightarrow$ For optimal answer run it for only the number of vertices in the path .

Algorithm b ::::Pre-Computation for finding path in Optimal Time

Let G be the Adjacency List of the Directed Graph .
$n \equiv$ No. of nodes

```
func APSP2(G, n )
        DP1[n+1][n+1] = G → Initialize all values to the values of Graph
        DP2[n+1][n+1] → If the path exists between two nodes then DP2[i][j] = j
                        (that is the direct edge from i -> j)else DP2[i][j] = -1
          for k=0 to n
            for i=0 to n
              for j=0 to n
                if DP1[i][k] != inf and DP1[k][j] != inf → We cannot travel through
                                                           edge that doesn't exist
                  if DP1[i][j] > DP1[i][k] + DP1[k][j]
                    DP1[i][j] = DP1[i][k] + DP1[k][j]
                    DP2[i][j] = DP2[i][k] → we found the shortest path between
                                            i, j through an intermediate node k).
                endif
              endif
            endfor
          endfor
         endfor
endfunc
```

Algorithm c ::: Finding path

Let G be the Adjacency List of the Directed Graph .
$n \equiv$ No. of nodes

```
DP ≡ APSP2(G, n)
Path ≡ []
func smallest-path(i, j)
    if i == j then
        Path = [i]
    else
        if Path is empty
            Path = [i, j]
        else if
        if DP2[i][j] != 0
            Path.insert(DP2[i][j])
            Smallest-path(i, k)
            Smallest-path(k, j)
        endif
    endif
endfunc
```

Path is our required path from i to j.