CS 345: Algorithms II

Submitted By:
Anirudh Kumar (Y9088)
Chandra Prakash (Y9181)

Assignment 3

1 Solution of 1

$\delta^0(i,j) = \infty$ if there is no edge between the two nodes
$\delta^0(i,j) = weight(i,j)$if there is an edge between the two nodes
$\delta^n(i,j)$ gives the distance between the nodes when all the nodes from 1 to n have been considered

1.1

We propose the dynamic programming algorithm for the given problem:
$\delta^k(i,j) = min\{\delta^{k-1}(i,j), \delta^{k-1}(i,k) + \delta^{k-1}(k,j)\}$

Pseudo Code:

```
function SHORTESTDISTANCE(M)
    M(i,j)=$\infty$ if there is no edge between i and j
    M(i,j)=weight(i,j) if there is an edge between i and j
    for k $\leftarrow$ 1 to $n$ do
        for i $\leftarrow$ 1 to $n$ do
            for j $\leftarrow$ 1 to $n$ do
                M(i,j) $\leftarrow$ min{M(i,k)+M(k,j),M(i,j)}
            end for
        end for
    end for
    for i $\leftarrow$ 1 to $n$ do
        for j $\leftarrow$ 1 to $n$ do
            M(i,j) $\leftarrow$ min{M(i,n)+M(n,j),M(i,j)}
        end for
    end for
end function
```

Proof by induction:

Base Case:When n=2, the distance between all the pair is same as in their weight Matrix

Induction Step:Let this be true for all values less than or equal to k.Then for the $(k+1)^{th}$ iteration, either $\delta^k(i,k)$ and $\delta^k(k,j)$ has already been evaluated or atleast one of them has not been evaluated.Consider the case when both are evaluated.Then,both these values are accessible in O(1) time from the matrix and the new value $\delta^k(i,j)$ is atmost the previous value in the table.So,the distance between the two is shorter than the previous value.Now consider the case when atleast one of these values has not been updated then we can compute the value of $\delta^k(i,k)$ has in this iteration and update the value of $\delta^k(i,j)$ in the next iteration.This is the reason why there is an update value at the end of iteration on k.Thus the time complexity is also $O(n^3)$

1.2

Let j be reachable from i through nodes $k_1, k_2, k_3, ..k_t$.Now, we create a two dimensional array R such that R[i,j]=k_1,R[k_1,j]=k_2 and so on and finally R[k_t,j]=0.
Justification:
Suppose we want to report the shortest path from i to j.Then we follow the array entry staring from R[i,j] until we get a zero.So,this is done in optimal time that is the number of nodes in the path.The change in algorithm is as under

```
function shortestDistance(M)
    M(i,j)=$\infty$ if there is no edge between i and j
    M(i,j)=weight(i,j) if there is an edge between i and j
    for k $\leftarrow$ 1 to $n$ do
        for i $\leftarrow$ 1 to $n$ do
            for j $\leftarrow$ 1 to $n$ do
                if M(i,k)+M(k,j) < M(i,j) then
                    if (R[i,j]$\neq$ 0 or j=k) and i$\neq$k then
                        R[i,j] = R[i,k]
                    else if i=k then
                        R[i,j]=R[i,j]
                    else
                        R[i,j]=k
                    end if
                end if
                M(i,j) $\leftarrow$ min{M(i,k)+M(k,j),M(i,j)}
            end for
        end for
    end for
    for i $\leftarrow$ 1 to $n$ do
        for j $\leftarrow$ 1 to $n$ do
            if M(i,n)+M(n,j) < M(i,j) then
                if (R[i,j]$\neq$ 0 or j=n) and i$\neq$k then
                    R[i,j] = R[i,n]
                else if i=n then
                    R[i,j]=R[i,j]
                else
                    R[i,j]=n
                end if
            end if
            M(i,j) $\leftarrow$ min{M(i,n)+M(n,j),M(i,j)}
        end for
    end for
end function
```

Thus the total space requirement is 2kn^2 ie $O(n^2)$ This algorithm also reports the shortest path ie entry i,$k_1, k_2, k_3, ..., k_t$,j by following the entries in the R array.

2 Solution of 2

2.1 Algorithm

2.1.1 Algorithm for question 2

```
for each $(i, j)$ do
    $wt(i, j) \leftarrow -log(r_{ij})$
end for
for each $v \epsilon V$ do
    if negCycle($G$, $v$) != -1 then
        Report the sequence of vertices in negCycle($G$, $v$) as an opportunity
    end if
end for
```

2.1.2 Function negCycle

```
function negCycle($G$, $s$)
    $d[s] \leftarrow 0$;
    for each $v \ \epsilon \ V/\{s\}$ do$d[v] \leftarrow \infty$;
    end for
    for each $i = 1$ to $n - 1$ do
        for each $(x, v) \ \epsilon \ E$ do
            if $d[x] + wt(x, v) < d[v]$ then
                $d[v] \leftarrow d[x] + wt(x, v)$; $pred[v] \leftarrow x$;
            end if
        end for
    end for
    $flag \leftarrow False$;
    for each $(x, v) \epsilon E$ do
        if $d[x] + wt(x, v) < d[v]$ then
            $flag \leftarrow True$; $cycleVertex \leftarrow v$;
        end if
    end for
    if $flag$ then
        $v \leftarrow pred[cycleVertex]$;
        while $v! = cycleVertex$ do
            append $v$ to $cycle$; $v \leftarrow pred[v]$;
        end while
        append $v$ to $cycle$;
        return $cycle$;
    else
        return -1;
    end if
end function
```

2.2 Proof of correctness

Algorithm 2 is a slightly modified 'SSSP with negative edges' algorithm discussed in class, so that it returns the negative cycle itself.
We find a negative cycle in the new graph, which is created after taking the negative log of the edge weights of original graph.

$$\sum_{v \epsilon Cycle} wt(i,j) < 0$$
$$\sum_{v \epsilon Cycle} -log(r_{ij}) < 0$$
$$-log(\prod_{v \epsilon Cycle} r_{ij}) < 0$$
$$\prod_{v \epsilon Cycle} r_{ij} > 1$$

Therefore a negative cycle in a modified graph correponds to an opportunity cycle in original graph.

2.3 Time Complexity

The time complexity of negCycle is O(mn), due to the second 'for' loop. Since it is run n times the overall time complexity is O(mn^2).