CS583_Assignment_4 **Mithilaesh Jayakumar(G01206238)**

23.1-11

In order to add the new edge after decreasing the weight to the given tree T, we need to create a cycle.
In order to remove any one of those newly added edges from the cycle, a spanning tree is required.
We will examine all the weights in this cycle which was formed by adding all the newly formed edges after decreasing the weights. The edge with the maximum weight is removed from the cycle.
As we can only add a single edge whose weight has been reduced and then change the graph back to the tree such that the total weight of the tree is minimized, it is exactly how we wanted the flow to be.

24.3-6

We find a path which has the maximized product of the probabilities in it. Assume a source s and a terminal t. The path p from s to t consists of {v0,...........vk} where v0 is the source and vk is the terminal.

Making w(u,v) = - log r(u,v) on each and every edge (u,v) helps to transform this problem into a single source shortest path problem as the log maintains the monotonicity and the negative log maximizes the problem.

Therefore the path p now becomes

$$p = \arg\max \prod_{i=0}^{k} r(v_{i-1}, v_i)$$

The graph now obtained can be solved by using Dijkstra's algorithm with O(E) for the initialization of the weights. The algorithm will run in O((V + E) log V + E)+O(E) = O(E log V) provided the graph has all the vertices reachable from the source and is sparse.

24.3

Let us assume that each currency represents a node and an exchange between any two currencies represents an edge with the weight as the exchange rate. Let us consider

$$
\begin{aligned}
x_1 \cdot x_2 \cdots x_k &> 1 && (1)\\
\Leftrightarrow \frac{1}{x_1} \cdot \frac{1}{x_2} \cdots \frac{1}{x_k} &< 1 && (2)\\
\Leftrightarrow \ln\left(\frac{1}{x_1} \cdot \frac{1}{x_2} \cdots \frac{1}{x_k}\right) &< \ln(1) && (3)\\
\Leftrightarrow \ln\left(\frac{1}{x_1}\right) + \ln\left(\frac{1}{x_2}\right) + \cdots + \ln\left(\frac{1}{x_n}\right) &< 0 && (4)
\end{aligned}
$$

a) A negative cycle occurs when we replace the weight of the edges by ln(1/x). Therefore in order to find the existence of a negative cycle we follow the below algorithm.

```
Alg CheckNegativeCycle(vertices, edges, cost)
n = total count of vertices
distance[n][n]
for i = 0 to n-1
{
    for j = 0 to n-1
    {
        if ((i, j) = edge)
        {
            distance[i][j] = cost(i,j)
        }
        else
        {
            distance[i][j] = infinity
        }
    }
}
for k = 0 to n-1
{
    for i = 0 to n-1
    {
        for j = 0 to n-1
{
if (distance[i][j] > distance[i][k] + distance[k,j])
{
distance[i][j] = distance[i][k] + distance[k][j]
}
}
}
}
for i = 0 to n-1
{
if (distance[i][i] < 0)
{
return true
}
}
return false
```

The running time of this algorithm is O(n^3) as it has three nested loops.

b) We modify the above algorithm mentioned in a) to find out the nodes in the negative cycle.

```
Alg CheckNegativeCycle(vertex, edges, cost)
n = total count of vertices
distance[n][n]
nodetovisit[n-1][n-1]
for i = 0 to n-1
{
for j = 0 to n-1
{
if ((i, j) = edge)
{
distance[i][j] = cost(i,j)
nodetovisit[i][j] = j
}
else
{
distance[i][j] = infinity
nodetovisit[i][j] = no node
}
}
}
for k = 0 to n-1
{
for i = 0 to n-1
{
for j = 0 to n-1
{
if (distance[i][j] > distance[i][k] + distance[k,j])
{
distance[i][j] = distance[i][k] + distance[k][j]
nodetovisit[i][j] = nodetovisit[i][k]
}
}
}
}
result <list of nodelist> = null
for i = 0 to n-1
{
if (distance[i][i] < 0)
{
negativecycle <nodelist> = < i >
current = i
do
{
current = nodetovisit[current][i]
add current to negativecycle
}
while (current == i);
add negativecycle to result
}
}
return result
```

Still the running time is O(n^3) for this algorithm as it has three loops.