CSCE 411 Design and Analysis of Algorithms

HW5: Solutions

Q - 23.1-9

Suppose that T' is not a minimum weight spanning tree in graph G' and S' is a minimum weight spanning tree in G'. Then, if we joined the subset of edges $T \backslash T'$ to S', then we would obtain a spanning tree S in the graph G. The weight of S would be smaller than the weight of T and this contradicts the condition that T is a minimum weight spanning tree. Thus, our assumption is false and T' is a minimum weight spanning tree in the graph G'.

Q - 22.3

(a)

We can use the Bellman-Ford algorithm on a suitable weighted, directed graph $G = (V, E)$, which we form as follows. There is one vertex in V for each currency, and for each pair of currency c_i and c_j, there is directed edges (v_i, v_j) and (v_j, v_i). (Thus, $|V| = n$ and $|E| = \binom{n}{2}$)
To determine edge weights, we start by observing that

$$R[i_1, i_2] \cdot R[i_2, i_3] \cdots R[i_{k-1}, i_k] \cdot R[i_k, i_1] > 1$$

if and only if

$$\frac{1}{R[i_1, i_2]} \cdot \frac{1}{R[i_2, i_3]} \cdots \frac{1}{R[i_{k-1}, i_k]} \cdot \frac{1}{R[i_k, i_1]} < 1$$

Taking logs of both sides of the inequality above, we express this condition as

$$ln \frac{1}{R[i_1, i_2]} + ln \frac{1}{R[i_2, i_3]} + \cdots + ln \frac{1}{R[i_{k-1}, i_k]} + ln \frac{1}{R[i_k, i_1]} < 0$$

Therefore, if we define the weight of edge (v_i, v_j) as

$$w(v_i, v_j) = ln \frac{1}{R[i, j]}$$

$$= -ln R[i, j]$$

then we want to find whether there exists a negative-weight cycle in G with these edge weights.

We can determine whether there exists a negative-weight cycle in G by adding an extra vertex v_0 with 0-weight edges (v_0, v_i) for all $v_i \in V$, running BELLMAN-FORD from v_0, and using the boolean result of BELLMAN-FORD (which is TRUE if there are no negative-weight cycles and FALSE if there is a negative-weight cycle) to guide our answer. That is,we invert the boolean resul tof BELLMAN-FORD.

This method works because adding the new vertex v_0 with 0-weight edges from v_0 to all other vertices cannot introduce any new cycles,yet it ensures that all negative-weight cycles are reachable from v_0.

It takes $\theta(n^2)$ time to create G, which has $\theta(n^2)$ edges.Then it takes $\theta(n^3)$ time to run BELLMAN-FORD. Thus, the total time is $\theta(n^3)$.

Another way to determine whether a negative-weight cycle exists is to create G and,without adding v_0 and its incident edges, run either of the all-pairs shortest-paths algorithms. If the resulting shortest-path distance matrix has any negative values on the diagonal, then there is a negative-weight cycle.

Algorithm 1 Algorithm for (a)

```
procedure hasNegCyc((V,E,c) : WeightedGraph) : boolean
    n = card(V)
    distance : Array[0, ..., n][0, ..., n]ofReal

    for i = 0 to n − 1 do
        for j = 0 to n − 1 do
            if (i, j) ∈ E then
                distance[i][j] = c(i, j)
            else
                distance[i][j] = +∞

    for k = 0 to n − 1 do
        for i = 0 to n − 1 do
            for j = 0 to n − 1 do
                if distance[i][j] > distance[i][k] + distance[k][j] then
                    distance[i][j] = distance[i][k] + distance[k][j]

    for i = 0 to n − 1 do
        if distance[i][i] < 0 then return true
    return false
```

(b)

We ran BELLMAN-FORD to solve part(a), we only need to find the vertices of a negative-weight cycle. We can do so as follows. First,relax all the edges once more. Since there is a negative-weight cycle,the d value of some vertex u will change. We just need to repeatedly follow the π values until we get back to u. In other words, above routine has to be modified such that it memorizes the shortest path.

The running time of this algorithm is still $O(n^3)$, because the $nextNodeloop$ loops n times at maximum.

Algorithm 2 Algorithm for (b)

```
procedure hasNegCyc((V,E,c) : WeightedGraph) : List of List of Node
    n = card(V)
    distance : Array[0, ..., n][0, ..., n]ofReal
    nextNode = Array[0, ..., n − 1][0, ..., n − 1]ofNode

    for i = 0 to n − 1 do
        for j = 0 to n − 1 do
            if (i, j) ∈ E then
                distance[i][j] = c(i, j)
                nextNode[i][j] = j
            else
                distance[i][j] = +∞
                nextNode[i][j] = nil

    for k = 0 to n − 1 do
        for i = 0 to n − 1 do
            for j = 0 to n − 1 do
                if distance[i][j] > distance[i][k] + distance[k][j] then
                    distance[i][j] = distance[i][k] + distance[k][j]
                    nextNode[i][j] = nextNode[i][k]

    result : ListofListofNode = φ
    for i = 0 to n − 1 do
        if distance[i][i] < 0 then
            negcyc : ListofNode =< i >
            runnode = i
            repeat
                runnode = nextNode[runnode][i]
                negcyc.pushBack(runnode)
            until runnode == i
            result.pushFront(negcyc)
    return result
```