CSCE 629 - 602 Analysis of Algorithms

October 19, 2016

Homework VI
Prof. Anxiao (Andrew) Jiang

Rishabh Yadav
UIN: 425009824

1A. 23-3 Solution

Say that the Minimum Spanning Tree for graph, G is T. The largest weight edge in T is $E(u, v) = a$. We shall try to prove this by contradiction. We assume $E(u, v) = a$ is not part of the bottleneck spanning tree B of G.

Case 1
The largest edge $E'(u'v') = b$, of the bottleneck spanning tree B is larger than edge $E(u, v) = a$ of T, i.e. $b > a$.

In this case, clearly the edge $E'(u'v') = b$, can not be the value of the bottleneck spanning tree B of graph G. Since there exists another spanning tree that provides a smaller largest edge. It is a conflict from the definition of bottleneck spanning tree.

Case 2
The largest edge $E'(u', v') = b$, of the bottleneck spanning tree B is smaller than edge $E(u, v) = a$ of T, i.e. $b < a$.

In this case, we consider that the edge $E(u, v)$, in T separates the vertices into two parts, left sub-tree T_l and right sub-tree T_r. Since it is a Tree (removing) any edge divided the tree into a forest with two sub-trees.

Let us say the bottleneck spanning tree B does not include edge, $E(u, v)$. From our assumption, in this bottleneck spanning tree B, the largest weight edge is $E'(u', v')$, occurs somewhere else and its smaller than $E(u, v)$, i.e. $b < a$.

If we delete $E(u, v)$, from T, then we should pick another edge to connect T_l and T_r. Say, a new edge $E''(p, q) = c$ is picked to get the bottleneck spanning tree B.

From the bottleneck spanning tree definition, weight of $E''(p, q) = c$ should be less than weight of $E'(u', v') = b$, i.e $c \leq b$ and,

From the minimum spanning tree definition, weight of $E''(p,q) = c$ should be greater than weight of $E(u,v) = a$, i.e $a < c$
Therefore $a < b$ (since $c \leq b$ and $a < c$). This is contradictory to our assumption of this case.

Therefore edge $E(u,v)$, is also in the bottleneck spanning tree and is the largest edge of it. Hence minimum spanning tree T, is also a bottleneck spanning tree B.

2A. 24.3 Solution

Part a

Idea

The idea is to make node for each currency c_i. We will make a graph out of this with each edge between nodes (currencies) c_i and c_j as a function of exchange rate between currency c_i c_j. To decide what function to use we show the following mathematical idea. If we were to use a function which is direct linear function of exchange rates then, we have to find a cycle with weight product greater than 1. That is,

$R[i,j].R[j,k].....R[m,l].R[l,i] > 1$ for some currencies $c_i, c_j, c_k, ..., c_m, c_l$

$$\frac{1}{R[i,j]}.\frac{1}{R[j,k]}.....\frac{1}{R[m,l]}.\frac{1}{R[l,i]} < 1$$

So far we have algorithms which find the sum of weights on path so we will try to convert product finding to sum finding. As we know from basic mathematics taking logarithm on both side should do the trick.

$$ln(\frac{1}{R[i,j]}.\frac{1}{R[j,k]}.....\frac{1}{R[m,l]}.\frac{1}{R[l,i]}) < ln(1)$$

Which is equivalent to

$$ln(\frac{1}{R[i,j]} + ln(\frac{1}{R[j,k]}) + ... + ln(\frac{1}{R[m,l]}) + ln(\frac{1}{R[l,i]}) < 0$$

The function is as follows:

$$f(c_i, c_j) = ln(\frac{1}{R[i,j]}).$$

Our problem is now reduced to finding a negative weight cycle (< 0) in the graph constructed in the above mentioned manner. We use Bellman-Ford algorithm for detecting a negative weight cycle.

Pseudo code

We add a new vertex s and connect it with every vertex c_i, the edge weight between $W(s, c_i)$ is 0. This is our source vertex.

Arbitrage (G)

1. convert the given currency and exchange rates into a graph as described. That is create a node for every currency c_i and created edge between every c_i c_j with weight $ln(\frac{1}{R[i,j]})$ where $R[i,j]$ is the exchange rate as given in matrix form for currency c_i and c_j

2. add a new vertex s and connect it with every vertex c_i, the edge weight between $W(s, c_i)$ is 0. This is our source vertex.
3. *d(s)* $= 0$
4. *d(v)* $= \infty$ $\forall$ vertex $v \epsilon V$ - $\{s\}$.
5. for $i = 1 \rightarrow |V - 1|$
   (a) for every edge $(u, v) \epsilon V$
      i. if $d(u) + w(u, v) < d(v)$
         A. $d(v) = d(u) + w(u, v)$
6. for every edge $(u, v) \epsilon E$
   (a) if $d(u) + w(u, v) < d(v)$
      i. return *Sequence-exists*
   (b) return *No-such-sequence*

Proof of correctness

By adding the new vertex s we are making sure that every vertex in the graph is reachable from the source vertex. Also by introducing this new vertex we are sure of introducing NO new cycle(s) since s has no incoming edge and hence no new cycle, Also we ensure that it doesn't effect the weight of cycles (which already exists) by having zero weights from s to c_i. Thus by introducing this new vertex s we make sure that we will find a negative weight cycle if it exists. From the idea described above we have already shown that negative weight cycle is basically the solution we were looking for. Hence the relaxation in step 4 will lead us to a negative cycle if it exists.

Time complexity

Time complexity for the Bellman-Ford algorithm is $O(VE)$. Here $|V| = n$ and $|E| = n^2$. Hence time complexity id $O(n^3)$

Time Complexity $= O(n^3)$

Part b

Idea

The idea is to make node for each currency c_i. We will make a graph out of this with each edge between nodes (currencies) c_i and c_j as a function of exchange rate between currency c_i c_j. To decide what function to use we show the following mathematical idea. If we were to use a function which is direct linear function of exchange rates then, we have to find a cycle with weight product greater than 1. That is,

$R[i,j].R[j,k].....R[m,l].R[l,i] > 1$ for some currencies $c_i, c_j, c_k, ..., c_m, c_l$

$$\frac{1}{R[i,j]}.\frac{1}{R[j,k]}.....\frac{1}{R[m,l]}.\frac{1}{R[l,i]} < 1$$

So far we have algorithms which find the sum of weights on path so we will try to convert product finding to sum finding. As we know from basic mathematics taking logarithm on both side should do the trick.

$$ln(\frac{1}{R[i,j]}.\frac{1}{R[j,k]}.....\frac{1}{R[m,l]}.\frac{1}{R[l,i]}) < ln(1)$$

Which is equivalent to

$$ln(\frac{1}{R[i,j]} + ln(\frac{1}{R[j,k]}) + ... + ln(\frac{1}{R[m,l]}) + ln(\frac{1}{R[l,i]}) < 0$$

The function is as follows:

$$f(c_i, c_j) = ln(\frac{1}{R[i,j]}).$$

Our problem is now reduced to finding a negative weight cycle (< 0) in the graph constructed in the above mentioned manner. We use Bellman-Ford algorithm for detecting a negative weight cycle.

In addition to the this we will also keep track of parent node $\pi(C_i)$, in order to print the sequence.

Pseudo code

We add a new vertex s and connect it with every vertex c_i, the edge weight between $W(s, c_i)$ is 0. This is our source vertex.

We use $\pi(v)$ to store the parent vertex.

Arbitrage (G)

1. convert the given currency and exchange rates into a graph as described. That is create a node for every currency c_i and created edge between every c_i c_j with weight

$ln(\frac{1}{R[i,j]})$ where $R[i,j]$ is the exchange rate as given in matrix form for currency c_i and c_j

2. add a new vertex s and connect it with every vertex c_i, the edge weight between $W(s, c_i)$ is 0. This is our source vertex.
3. $d(s) = 0$
4. $d(v) = \infty$ $\forall$ vertex $v \epsilon V$ - $\{s\}$.
5. $\pi(v) = null$ $\forall$ vertex $v \epsilon V$
6. for $i = 1 \rightarrow |V - 1|$
   (a) for every edge $(u, v) \epsilon V$
      i. if $d(u) + w(u, v) < d(v)$
         A. $d(v) = d(u) + w(u, v)$
         B. $\pi(v) = u$
7. for every edge $(u, v) \epsilon E$
   (a) if $d(u) + w(u, v) < d(v)$
      i. sequenceStack = ϕ
      ii. current vertex = $\pi(v)$
      iii. while current vertex != v
         A. sequenceStack.push(current vertex)
         B. current vertex = $\pi(currentvertex)$
      iv. while sequenceStack:
         A. vertex = sequenceStack.pop()
         B. print vertex
      v. return *Sequence-exists*
   (b) return *No-such-sequence*

Proof of correctness

By adding the new vertex s we are making sure that every vertex in the graph is reachable from the source vertex. Also by introducing this new vertex we are sure of introducing NO new cycle(s) since s has no incoming edge and hence no new cycle, Also we ensure that it doesn't effect the weight of cycles (which already exists) by having zero weights from s to c_i. Thus by introducing this new vertex s, we make sure that we will find a negative weight cycle if it exists. From the idea described above we have already shown that negative weight cycle is basically the solution we were looking for. Hence the relaxation in step 4 will lead us to a negative cycle if it exists.

Time complexity

Time complexity for the Bellman-Ford algorithm is $O(VE)$. Here $|V| = n$ and $|E| = n^2$. Hence time complexity id $O(n^3)$

Time Complexity $= O(n^3)$

3A. 26.1-7 Solution

The problem can be reduced to the standard maximum flow problem, by splitting every vertex v into two vertices v_{in} and v_{out}, adding one edge (v_{in}, v_{out}) of capacity $l(v)$, and then converting every edge (u, v) to an edge (u, v_{in}) and every edge (v, w) to an edge (v_{out}, w). Solving the (standard) maximum flow problem on the new network is equivalent to solving the maximum flow with vertex capacity constraints in the original network. Since the vertex capacity $l(v)$ is the maximum flow that can pass through v. Now represented by maximum flow that can pass through edge (v_{in}, v_{out}).

If the original graph is $G(V, E)$ and the transformed graph is $G'(V', E')$ then,

$|V'| = 2 * |V|$, and,
$|E'| = |V| + |E|$

References

Introduction to Algorithms by T. Cormen, C. Leiserson, R. Rivest, C. Stein