University of Washington
Department of Computer Science and Engineering
CSE 417, Winter 2020

March 13, 2020

Homework 9, Problem 1

Problem 1 (10 points):

(Page 324, Exercise 13.) The problem of searching for cycles in graphs arises naturally in financial trading applications. Consider a firm that trades shares in n different companies. For each pair $i \neq j$, they maintain a trade ratio r_{ij}, meaning that one share of i trades for r_{ij} shares of j. Here we allow the rate r to be fractional; that is, $r_{ij} = \frac{2}{3}$ means that you can trade three shares of i to get two shares of j.

A *trading cycle* for a sequence of shares $i_1, i_2, \ldots, i_k$ consists of successively trading shares in company i_1 for shares in company i_2, then shares in company i_2 for shares i_3, and so on, finally trading shares in i_k back to shares in company i_1. After such a sequence of trades, one ends up with shares in the same company i_1 that one starts with. Trading around a cycle is usually a bad idea, as you tend to end up with fewer shares than you started with. But occasionally, for short periods of time, there are opportunities to increase shares. We will call such a cycle an *opportunity* cycle, if trading along the cycle increases the number of shares. This happens exactly if the product of the ratios along the cycle is above 1. In analyzing the state of the market, a firm engaged in trading would like to know if there are any opportunity cycles.

Give a polynomial-time algorithm that finds such an opportunity cycle, if one exists.

Answer:

algorithm:

```
# build a graph G with each edge length equal to -log(r_ij)
# for each pair of share i,j
for each pair of share i and j:
    cost[i, j] = -log(r[i,j])
# add a source node s to every node in graph G with zero edge cost
for each share:
    cost[s, j] = 0
n = number of nodes of G
 maintain a array m with length of n
# keep incoming node (predecessor) of each node
maintain a array p with length of n
# for the graph G, use Bellman-Ford algorithm to find negative cycle
for each w:
    set m[w] = infinity
    p[v] = null
```

```
set m[0] = 0
for i = 1 to n-1
    for each w:
        for each x:
            if (m[w] > m[x] + cost[x, w])
                P[w] = x
                m[w] = m[w] + cost[x, w]

# traceback the negative cycle if exist
maintain a list ans to keep result
for each w:
    for each x:
        if (m[w] > m[x] + cost[x, w]):
            cur = p[x]
            while cur != x:
                cur = p[cur]
                ans.append(cur)
```

proof:
Since the sum of logarithms equal to the multiplication of base number, hence by setting edge cost equal to $-\log r_{ij}$, the opportunity cycle ($\prod_{ij} r_{ij} > 1$) can be found from solving equilent problem of $\sum_{ij} -\log r_{ij} < 0$. By using a adjusted version of Bellman Ford algorithm, the algorithm can solve the problem of finding a negative cycle with $O(mn)$, and since the preproccess steps with linear time complexity (traverse graph once) therefore it is a polynomial time algorithm to find an opportunity cycle.