CS170–Spring 2019 — Homework 5 Solutions

Ran Liao, SID 3034504227

February 21, 2019

Collaborators:Jingyi Xu, Renee Pu

1 Study Group

Name	SID
Ran Liao	3034504227
Jingyi Xu	3032003885
Renee Pu	3032083302

2 Updating Labels

(a) **Main Idea**

Run DFS from root node r and update label whenever do post operation to a vertex. Suppose vertex v is currently being operated and $l(v) = k$. The kth ancestor of v is located on the kth element in stack counting from top (this value can be access in $O(1)$ time, if we use an array and a pointer to simulate a stack). Donate this vertex as w, then update $l(v) = l(w)$.

(b) **Proof of Correctness**

Labels are updated from leaves r to root nodes. Therefore the ancestor of a given vertex v is updated after v. This property can guarantee the correctness of algorithm.

(c) **Runtime Analysis**

DFS will cost $O(|V| + |E|)$.

3 Count Four Cycle

(a) **Main Idea**

Suppose A is the adjacency matrix of graph G. $A_{i,j} = 1$ iff there's an edge between vertex i and j. Compute A^2 and then subtract 1 from $A^2_{i,i}$ for $\forall A_{i,j} = 1$. Then compute $A^3 = A^2A$ and subtract 1 from $A^3_{i,j}$ for $\forall A_{j,k} = 1$. Lastly, compute $A^4 = A^3A$. There's a four cycle iff $\exists A^4_{i,i} > 0$.

(b) **Proof of Correctness**

A represents the number of path between every pair of vertices with length 1. A^2 represents the number of path between every pair vertices with length 2, and so on. Therefore check $A^4 i, i$ can reveal the existence of four cycles. All subtraction made previous is for eliminating invalid cycles.

(c) **Runtime Analysis**

The trivial matrix product will cost $O(|V|^3)$ time, and all subtraction can be finished within $O(|V|^2)$ time. Therefore, the overall runtime is $O(|V|^3)$.

4 Constrained Dijkstra

(a) **Main Idea**

Run Dijkstra algorithm on vertex v_0 and record shortest path in array p. Then reverse all edges in G, denote the new graph as G_M. Run Dijkstra algorithm on vertex v_0 again in new graph and record shortest path in array p_M. The shortest path between u and v can be reconstruct by combining path from v_0 to u in G_M and path from v_0 to v in G.

(b) **Proof of Correctness**

The path from u to v can be divided into two parts, namely, path from u to v_0 and path from v_0 to v. The shortest path from u to v_0 can be found in G_M. The shortest path from v_0 to v can be found in G.

(c) **Runtime Analysis**

Dijkstra algorithm will cost $O((|V| + |E|)\log|V|)$

5 Arbitrage

(a) (i) **Main Idea**

Construct a graph G where v_i represents currency c_i. The weight of edge between v_i and v_j will be $\frac{1}{r_{i,j}}$. Run a modified Bellman-Ford on this graph start with vertex s. The update role is changed to $\text{dist}(v) = \min(\text{dist}(v), \text{dist}(u) \times l(u, v))$. Initially, $\text{dist}(s) = 1$.

(ii) **Proof of Correctness**

The multiplication and addition has similar associative and commutative property. All deduction for Bellman-Ford algorithm will still holds true for the modified version. In modified version, edge with weight less than 1 will be consider as a "negative" edge.

(iii) **Runtime Analysis**

The modification will not change Bellman-Ford algorithm's runtime. Therefore it is $O(|V||E|)$.

(b) (i) **Main Idea**

Use the same graph defined in part (a) and add additional iteration to the outer loop. If some vertices is updated in the final iteration, arbitrage situation exists.

(ii) **Proof of Correctness**

If there's an arbitrage situation, there must exist a loop where weights' product is less than 1. This is similar to have a negative loop in the original version of Bellman-Ford algorithm. This "negative" loop will cause the shortest path never stop updating.

(iii) **Runtime Analysis**

The modification will not change Bellman-Ford algorithm's runtime. Therefore it is $O(|V||E|)$.

6 Bounded Bellman-Ford

(a) **Pseudocode**

Modified-Bellman-Ford($G = (V, E)$){

for $\forall u \in V$:

dist(u) $= \infty$

prev(u) $=$ nil

dist(s) $= 0$

for i from 1 to k:

for $\forall e = (u, v) \in E$:

if(dist(u) $+ l(u, v) <$ dist(v))

dist(v) $=$ min(dist(v), dist(u) $+ l(u, v)$)

prev(v) $= u$

}

(b) **Proof of Correctness**

Each iteration in outer loop will try to consider the shortest path with 1 more edges. Therefore, after k iterations, the dist will contain information about the shortest path with no more than k edges.

(c) **Runtime Analysis**

The outer loop will run k times, therefore the runtime is $O(k|E|)$.