CS170–Fall 2022 — Homework 4 Solutions

Deming Chen `cdm@pku.edu.cn`

Last Modified: October 22, 2022

Collaborators: NONE

2 Updating a MST

(a) $e \in E'$ and $\hat{w}(e) < w(e)$

The algorithm is very simple: T is still a minimum spanning tree. So in this scenario, updating the minimum spanning tree takes time $\mathcal{O}(1)$.

Proof of correctness: assume that the new minimum spanning tree is not T but T'. No matter whether $e \in T'$ or $e \notin T'$, T' is lighter than T, both before and after modifying the weight of e, which contradict the fact that T is the minimum spanning tree initially.

(b) $e \notin E'$ and $\hat{w}(e) < w(e)$

Adding e in T, then T contains a cycle. Remove the heaviest edge in the cycle and we get a new tree T_1, which is a new minimum spanning tree. The algorithm for finding this cycle is DFS in $T \cup \{e\}$. When searching $T \cup \{e\}$ starting at e, we save the previous vertex of every vertex along the path. Once e is met twice, we find the cycle. This algorithm takes time $\mathcal{O}(|V| + |E'|) = \mathcal{O}(|V|)$.

Proof of correctness: assume that the new minimum spanning tree is not $T_1 = (V, E_1)$ but $T_2 = (V, E_2)$. There are three cases.

i) $e \notin E_2$. In this case, T_2 is lighter than T_1 and T_1 is not heavier than T. So, T_2 is lighter than T, which contradicts the fact that T is the minimum spanning tree initially.

ii) $e \in E_2$, and $\hat{w}(e)$ is still the largest in that cycle. Removing e from T_2 we get two disconnected trees. Besides e, there must be another edge e' in that cycle which connects these two trees. In this case, $w(e') < \hat{w}(e)$. So, e' and these two trees form a spanning tree in G, which is lighter than T_2. It contradicts the fact that T_2 is the minimum spanning tree.

iii) $e \in E_2$, and $\hat{w}(e)$ is not the largest in that cycle. Assume the heaviest edge is e' (not necessarily equals to e' in case ii). Remove e from T_2 and add e' in it. If we get a new tree (call it T_3), we have (all evaluated after modifying the weight of e):

$$w(T_2) - w(T_3) = w(T_1) - w(T). \tag{1}$$

Since $w(T_2) < w(T_1)$, $w(T_3) < w(T)$, which contradicts the fact that T is the minimum spanning tree before modifying the weight of edge e. So, the result cannot be a tree, which implies some edges in T_2 and e' form a cycle C. e' is the heaviest edge in that cycle, otherwise we can replace the heaviest edge with e' in T_2 so that T_2 becomes lighter (this cannot happen since T_2 is the minimum spanning tree).

Note $e' \in T$. Removing e' from T we get two disconnected trees. Besides e', there must be another edge e'' in cycle C which connects these two trees. $w(e'') < w(e')$, so e'' and these two trees form a spanning tree in G, which is lighter than T. It contradicts the fact that T is the minimum spanning tree before modifying the weight of edge e.

(c) $e \in E'$ and $\hat{w}(e) > w(e)$

Removing e from T we get two disconnected trees T_1 and T_2 (also a cut of graph G). Add the lightest edge across the cut, we get a minimum spanning tree.

Vertices in T_1 and all edges whose endpoints are both in T_1 form a subgraph of G (call it G_1). Also, vertices in T_2 and all edges whose endpoints are both in T_2 form a subgraph of G (call it G_2). Every edge in E is either across the cut, in G_1, or in G_2. After DFS in G_1 and G_2 from a endpoint of edge e we know the edges across G_1 and G_2. The algorithm takes time $\mathcal{O}(|V| + |E|)$.

Proof of correctness: let $\hat{T} = (\hat{V}, \hat{E})$ be the minimum spanning tree after increasing the weight of e. From (a) and (b) we know that, if $e \in \hat{E}$, $\hat{T} = T$; otherwise, $\hat{T}$ and T differ by only one edge, so there must be another edge in $\hat{T}$ which crosses T_1 and T_2. In both cases, there exists only one edge in $\hat{E}$ across T_1 and T_2, and it is just the lightest one.

(d) $e \notin E'$ and $\hat{w}(e) > w(e)$

The algorithm is very also simple: T is still a minimum spanning tree. So in this scenario, updating the minimum spanning tree takes time $\mathcal{O}(1)$.

The proof of correctness is the same as that in (a).

3 Rigged Tournament

Define a weighted directed graph $G = (V, E)$ based on the points in the games. n vertices represent n teams. $(u, v) \in E$ means that if teams u and v plays, u will win, and the weight of the edge is the points scored in that game. Our goal is to find the spanning arborescence A rooted at i^* of maximum weight. Since the number of edges in A is fixed, the algorithm for finding the maximum spanning arborescence is similar to that for finding the minimum spanning arborescence.

Unluckily, it is challenging and I could not come up with an algorithm myself. It is a classic problem, and I found the wikipedia page of an algorithm.

4 Arbitrage

Construct a weighted directed graph $G = (V, E)$, where each vertex represent a currency and the weights of edge $w(u, v) = -\log r_{u,v}$.

(a) Converting currency a into currency b

Assume we successively converting a to currencies $c_{i_2}, c_{i_3}, \cdots, c_{i_{k-1}}, b$. One unit of currency a can be converted into $r_{a,i_1} r_{i_1,i_2} \cdots r_{i_{k-1},b}$ units of currency b. Maximize the product of rates is equivalent to minimize

$$\begin{aligned} & -\log(r_{a,i_1} \cdot r_{i_1,i_2} \cdot \cdots \cdot r_{i_{k-1},b}) \\ = & -(\log r_{a,i_1} + \log r_{i_1,i_2} + \cdots + \log r_{i_{k-1},b}) \\ = & w(a, i_1) + w(i_1, i_2) + \cdots + w(i_{k-1}, b). \end{aligned}$$

Our goal is to find a shortest path in the graph from a to b. Since some rates is larger than 1, weights of some edges in the graph G is less than 0. We can use Bellman-Ford algorithm, and its run-time is $\mathcal{O}(|V| \cdot |E|) = \mathcal{O}(n^3)$.

(b) there is a sequence of currencies $c_{i_1}, c_{i_2}, \cdots, c_{i_k}$ such that $r_{i_1,i_2} \cdot r_{i_2,i_3} \cdot \cdots \cdot r_{i_k,i_1} > 1$ is equivalent to $w(i_1, i_2) + w(i_2, i_3) + \cdots + w(i_k, i_1) < 0$.

Choose a vertex i in G randomly. We can also use Bellman-Ford algorithm update to the `dist` value from i of all vertex n times. If some `dist` value is reduced during the final round, there is a negative cycle in graph G, which implies the possibility of arbitrage. It takes time $\mathcal{O}(|V| \cdot |E|) = \mathcal{O}(n^3)$.

5 Bounded Bellman-Ford

Besides recording `dist` value of each vertex, we record each vertex's `num` value which equals to the number of edges of the shortest path from source s to that vertex we have found so far. In the beginning, $\texttt{num}(s) = 0$ and for $u \neq s, \texttt{num}(u) = \infty$. The *update* operation, then, becomes

```
procedure update((u, v) ∈ E)
if dist(u) + l(u, v) < dist(v) and num(u) ⩽ k − 1:
    dist(v) = dist(u) + l(u, v);
    num(v) = num(u) + 1;
```

After updating all edges $k-1$ times, a vertex's `dist` value is length of the shortest path from s to that vertex with the restriction that the path must have at most k edges.