CSORW4231 HOMEWORK 7

Due Thu, May 11 Jun Hu

(jh3846)

Problem 1. Exercise 23.1-3 and 23.1-8 on Page 629.

Solution.

23.1-3 Denote A be the set containing the minimum spanning tree edges. By removing the edge $(u, v) \in A$ we obtain the set A'. Considering a cut determines $(S, V - S)$ s.t. $u \in S$ and $v \in V - S$ and no cut edges in A', then (u, v) must be the light edge among these cut edges.

Because in order to form the minimum spanning tree edges A, the light edge is needed to be selected from the cut edges. If (u, v) is not the light edge, there must be another edge (u', v') s.t. $A = A' \cup \{(u', v')\}$. However, by our assumption, $A = A' \cup \{(u, v)\}$, which means $(u', v') = (u, v)$. To sum up, (u, v) has to be the light edge among the cut edges.

23.1-8 Denote L' be the sorted list of T', edge $e \in T$ and $e' \in T'$, $\exists$:

$w(e_1) \leq w(e_2) \leq \cdots \leq w(e_k) \leq \cdots \leq w(e_i) \leq \cdots \leq w(e_j) \leq \cdots \leq w(e_n) \in L$

$w(e'_1) \leq w(e'_2) \leq \cdots \leq w(e'_k) \leq \cdots \leq w(e'_i) \leq \cdots \leq w(e'_j) \leq \cdots \leq w(e'_n) \in L'$

Both L and L' contain n elements, suppose $L' \neq L$, assume the first different occurred at $w(e_i) \neq w(e'_i)$, without loss of generality, assume $w(e_i) > w(e'_i)$:

If T contains e'_i, because weights before e_i in T are the same with weights before e'_i in T', so $w(e'_i)$ in L should at least emerge after $w(e_i)$, assume $e'_i = e_j$ in T, that is $w(e'_i) = w(e_j) \geq w(e_i)$, which contradicts that the assumption $w(e_i) > w(e'_i)$ at the beginning.

If T doesn't contain e'_i, by adding e_i in T will form a cycle, in this cycle, there must be an edge e_j doesn't in T', s.t. $\exists\ e_j \geq e'_i$, weights before $w(e)$ and $w(e')$ are in the same order, so $e_j \geq e_i$. However, due to the minimum spanning tree properties, other edges in this cycle in T must have weights smaller than e'_i, e_j is also one of these edges, s.t. $w(e_j) \leq w(e'_i)$, that is $w(e) \leq w(e_j) \leq w(e'_i)$ which contradicts the assumption $w(e_i) > w(e'_i)$ at the beginning.

Consequently, there is no such different weights existing in the L or L', L and L' must be in the same order of edge weights.

□

Problem 2. Problem 23-4 on Page 641: Alternative minimum-spanning-tree algorithms. For each of the three algorithms, either give a counterexample or prove that it always outputs a minimum spanning tree. Make sure your proof is written clearly and concisely. Also there is no need to describe efficient implementations of these algorithms.

Solution.

a. the algorithm always returns a minimum spanning tree correctly.

Correctness: Because the algorithm will delete all the edges as long as the remain part is still connected, all cycles in G will be broken, the returned edge set T must be a tree. Denote T^* be a minimum spanning tree of G, $|T| = |T^*|$ for each edge $e \notin T$(removed by the algorithm), $\exists\ e \in T^*$ or $e \notin T^*$.

If $e \notin T^*$, which is trivial, the removal is correct.

If $e \in T^*$. First of all, e must lie in a cycle, otherwise e can not be removed because of causing disconnection. Secondly, e will be deleted only if any other edge is not larger than $w(e)$, otherwise it would be discovered before e and deleted. There also must be an edge $v \notin T^*, v \in T$ in this cycle was kept by the algorithm instead of e to hold all connected property. $\exists\ w(v) \leq w(e)$. Moreover, because $e \in T^*$, it can be even speculated that only exists $w(v) = w(e)$, the removal of e is still correct.

b. the algorithm may not return a minimum spanning tree. Counterexample: As **Figure 1**. If the arbitrary order of the edges is: $\{(v, w) : 3, (u, w) : 2, (u, v) : 1\}$. The algorithm will return $T = \{(v, w) : 3, (u, w) : 2\}$, with a summation of weights of 5, but the minimum spanning tree is $\{(u, v) : 1, (u, w) : 2\}$, with the minimum summation of weights of 3.

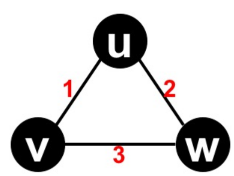

Figure 1: Graph of **b**

c. the algorithm always returns a minimum spanning tree correctly.

Correctness: For each edge $e \in G$, it will be first adding to T and only one edge be removed to prevent from forming a cycle, typically this will add all edges if there is no cycle at all. So in the end, T will contain no cycle but all connected by edges as a tree. Denote T^* be a minimum spanning tree of G, $|T| = |T^*|$ for each edge $e \notin T$(removed by the algorithm), $\exists\ e \in T^*$ or $e \notin T^*$.

If $e \notin T^*$, which is trivial, the removal is correct.

If $e \in T^*$. First of all, e lie in the new forming cycle. Secondly, e will be deleted only if any other edge is not larger than $w(e)$, otherwise the larger edge in the cycle will be deleted. There also must be an edge $v \notin T^*, v \in T$ in this cycle was kept by the algorithm instead of e to hold all connected property. $\exists\ w(v) \leq w(e)$. Moreover, because $e \in T^*$, it can be even speculated that only exists $w(v) = w(e)$, the removal of e is still correct.

$\square$

Problem 3. Problem 24-4 on Page 679: Gabows scaling algorithm for single-source shortest paths.

Solution.

a. Since weights of the graph $G = (V, E)$ are nonnegative, we can implement Dijkstra's algorithm to find the single-source shortest paths.

Especially, $\delta(s, v) \leq |E|$, the shortest path distances is bounded by E, and they are integers, which means we can maintain an array of linked list $L = [0, 1, 2, \cdots, i, \cdots, |E|]$, s.t. i can represent any $\delta(s, v)$, and $L[i]$ contains such vertices that $v.d = i$. To construct the L takes $O(V)$. For DECREASE-KEY on the vertex v, simply remove v from $L[v.d]$ and decrease its key to i by adding it to the $L[i]$ list, each call takes $O(1)$, totally $O(E)$. For EXTRACT-MIN, no need searching all elements in L, the smallest i s.t. $L[i]$ is non-empty returns the minimum. Because we always extract vetices with the non-decreasing shortest path distance i, so there will not be any vertex having shortest path distance less than i, each EXTRACT-MIN takes $O(1)$, and for the whole array of $|E|$ linked lists, with the total elements of the lists are $|V|$, and no backtracking, so it takes $O(E+V)$. Because $|E| > |V|-1$, The overall running time is $O(E)$.

b. Because w_1 uses only the first first significant bit of the actual edge weights, which means $\forall (u, v) \in E$ s.t. $w_1(u, v) \in \{0, 1\}$. The maximum shortest path is at the most $|V| - 1$ for all weights to be 1. That is $\delta_1(s, v) \leq |V| - 1 \leq |E|$. use the conclusion in **a**, it takes $O(E)$ to compute $\delta_1(s, v)$

c. By definition, w_i is the i most significant bits of w, consequently can be obtained by shifting w_{i-1} to left by 1 space, which is calculated by doubling w_{i-1} plus the *ith* significant bit – that is either 0 or 1. As a result, $w_i(u, v) = 2w_{i-1}(u, v)$ or $w_i(u, v) = 2w_{i-1}(u, v) + 1$.

The equations implies $2w_{i-1}(u, v) \leq w_i(u, v) \leq 2w_{i-1}(u, v) + 1$.

Let P be the shortest path from s to v, $\forall v \in V, \exists$:

$$\delta_i(s, v) = \min \sum_{(u,w) \in P} w_i(u, w)$$

$$\min \sum_{(u,w)\in P} 2w_{i-1}(u, w) \leq \min \sum_{(u,w)\in P} w_i(u, w) \leq \min \sum_{(u,w)\in P} (2w_{i-1}(u, w) + 1)$$

$$2 \cdot \min \sum_{(u,w)\in P} w_{i-1}(u, w) \leq \delta_i(s, v) \leq \min(2 \sum_{(u,w)\in P} w_{i-1}(u, w) + \sum_{(u,w)\in P})$$

$$2\delta_{i-1}(s, v) \leq \delta_i(s, v) \leq 2\delta_{i-1}(s, v) + |V| - 1$$

d. By definition and **c**

$$\widehat{w}_i(u, v) = w_i(u, v) + 2\delta_{i-1}(s, u) - 2\delta_{i-1}(s, v) \geq 2w_{i-1}(u, v) + 2\delta_{i-1}(s, u) - 2\delta_{i-1}(s, v)$$

By triangle inequality:

$$w_{i-1}(u, v) + \delta_{i-1}(s, u) \geq \delta_{i-1}(s, v)$$

s.t.

$$\widehat{w}_i(u, v) \geq 0$$

e. Let $P =< s, u_1, u_2, \cdots, u_n, v >$ be the shortest path from s to v:

$$\begin{aligned}\widehat{\delta}_i(s,v) &= \min \sum_{e \in P} \widehat{w}_i(e) \\ &= \min \big(\widehat{w}_i(s,u_1) + \widehat{w}_i(u_1,u_2) + \cdots + \widehat{w}_i(u_n,v)\big)\end{aligned}$$

Using $\widehat{w}_i(u,v) = w_i(u,v) + 2\delta_{i-1}(s,u) - 2\delta_{i-1}(s,v)$ to expand the equation s.t.:

$$\begin{aligned}\widehat{\delta}_i(s,v) &= \min \left(2\delta_{i-1}(s,s) - 2\delta_{i-1}(s,v) + \sum_{e \in P} w_i(e)\right) \\ &= -2\delta_{i-1}(s,v) + \min \left(\sum_{e \in P} w_i(e)\right) \\ &= -2\delta_{i-1}(s,v) + \delta_i(s,v)\end{aligned}$$

Which is:

$$\delta_i(s,v) = \widehat{\delta}_i(s,v) + 2\delta_{i-1}(s,v)$$

And because we've proved in **c**

$$\delta_i(s,v) \leq \delta_i(s,v) + |V| - 1$$

s.t.:

$$\widehat{w}_i(u,v) \leq |V| - 1 \leq |E|$$

f. $\widehat{w}_i(u,v)$ can be compute in $O(E)$ as described in **d**.
$\widehat{\delta}_i(s,v)$ is bounded by $|E|$ as shown in **e**, using results in **a** to compute $\widehat{\delta}_i(s,v))$ in $O(E)$.

From above results, we can compute $\delta_i(s,v)$ by the equation in **e** in $O(V) = O(E)$ for each vertex.

As shown in **b**, compute $\delta_1(s,v)$ in $O(E)$, then compute $i = 2$ from $\delta_1(s,v)$ in $O(E)$, $\cdots$, when $i = k$, that we have $\delta(s,v) = \delta_k(s,v)$, s.t. the $\delta(s,v)$ is computed in $k(O(E)) = O(E \lg W)$.

□

Problem 4. Problem 25-2 on Page 706: Shortest paths in -dense graphs. Skip a). For a d-ary min-heap, Insert takes time O(logd n); Extract-Min takes time O(d logd n); and Decrease-Key takes time O(logd n). Check Chapter 6 and Problem 6-2 if you are interested in d-ary min-heaps. But for this problem you may use these facts for free.

Solution.

b. Set $d = n^\varepsilon = |V|^\varepsilon$, Dijkstra's algorithm takes:

$$O(V \cdot d \log_d V + E \cdot \log_d V) = O(\frac{V}{\varepsilon} V^\varepsilon + \frac{E}{\varepsilon}) = O(V^{1+\varepsilon} + E) = O(E)$$

c. Run the algorithm of **b** in $|V|$ times for each vertex as the source, which takes $O(VE)$.

d. By using d-array min-heaps, we can invoke Johnson's algorithm, which first perform Bellman-Ford to re-weight edges to be all non-negative, then perform Dijkstra's algorithm as described. The running time takes $O(VE)$ totally.

$\square$

Problem 5. Show that if CLIQUE (the decision problem, where a pair (G, k) is in the language iff the undirected graph G has a clique of size at least k) is in P, then there is a polynomial-time algorithm that, given any undirected graph G, finds a clique of G of maximum size.

Solution.

If CLIQUE is in P to solve $(G(V, E), k)$, so it takes $O(V^{\varepsilon})$ as the input is the undirected graph. We can run this CLIQUE of decision problem from $k = |V|$, if return == 'no', call CLIQUE on $k = k - 1$, if return == 'yes', current k is the maximum size of the clique of G. The loop will run $O(V)$ in the worst case, so the algorithm takes $O(V^{\varepsilon+1})$ – that is in P as well.

□

Problem 6. In the Dominating Set (decision) problem we are given a directed graph G = (V, E) and an integer k. We are asking whether there is a set D of k or fewer vertices such that for each v / D there is a u D with (u, v) E. Show that Dominating Set is NP-complete. (Start from Vertex Cover, obviously a very similar problem, and make a simple local replacement.)

Solution.

First of all, Dominating Set is in NP. given any set D, we can verify each $v \notin D$ and all its neighbors u whether or not there exists $u \in D$ easily. The verification takes polynomial time.

Recall the Vertex Cover is in NP-Completeness: given a graph $G = (V, E)$ and an integer k. We are asking whether there is a set D of k at most k size of vertex cover (A vertex cover is a subset $D \subseteq V$ that for each edge (u, v) either $u \in S$ or $v \in S$ or both)?

Reduction: Given a directed graph $G = (V, E)$, we replace each $e \in E$ by a triangle to form G', s.t., $k' = k$, $G' = (V', E')$ where $V' = V \cup V_{add}$ s.t. $V_{add} = \{v_{e_i} | e_i \in E\}$ and $E' = E \cup E_{add}$ where $E_{add} = \{(v_{e_i}, v_m), (v_{e_i}, v_n) | e_i = (v_m, v_n) \in E\}$.

If yes-instance for Vertex Cover, that is G has a the subset D of k, then D also can form a dominating set in G'. Because for u either in the D, or u is not but v must in for the edge (u, v).

On the other hand, if there is k' size subset dominating set for G', this dominating set D can just use the vertex $u \in V$, because whenever a v_{e_j} is in the dominating set, we can just substitute it with one of v_m, v_n to keep the k' unchanged, then for any $u \notin D$, there is an edge (u, v) that $v \in D$. So there also exists $k = k'$ size subset vertex cover for graph G.

$\square$