CSE 101: Homework #4

Due on May 9, 2024 at 23:59pm

Professor Jones

Ray Tsai

A16848188

Problem 1

You are given a connected graph G with n vertices and with positive edge weights $w : E \to \mathbb{R}^+$. You wish to find a connected subgraph of G with $n-1$ vertices that has minimum total cost.

Proof. Consider the following algorithm on G:

Set the cost of T to ∞. For each vertex $v \in V$, run Kruskal's algorithm on $G - v$ and get T_v. If T_v has exactly n edges $cost(T_v) \leq cost(T)$, update T to T_v. After all iterations, return T.

Runtime Analysis:

Running Kruskal takes $O(|E| \log |V|)$ time. Note that we don't actually need to remove v from G, as we may just ignore v while running Kruskal. Checking the the connectedness and and the minimality of T_v only takes constant time. Hence, each iteration takes $O(|E| \log |V|)$ time. Since the algorithm runs Kruskal's algorithm $|V|$ times, it takes $O(|V||E| \log |E|)$ time in total.

Justification for correctness:

We show that T is the MST of $n-1$ vertices with minimum total cost.

We first show that $T = T_v$ for some v. In particular, we need to show that T is guaranteed to be updated. Note that G is a connected graph, which contains a spanning tree S. Remove any leaf u from S yields a spanning tree of $n-1$ vertices, which ensures that G contains a connected subgraph with $n-1$ vertices. But then running Kruskal on $G - u$ gives a spanning tree on $n-1$ vertices, and this guarantees T to be updated to some T_u. Additionally, T is only updated to T_v if T_v has n edges. Since Kruskal ensures T_v has no cycles, T_v is a connected tree of $n-1$ vertices. Therefore, T is a MST of some $n-1$ vertices at the end of the algorithm.

Now we show that T is optimal. Let T' be a connected subgraph of G of $n-1$ vertices such that $cost(T') < cost(T)$. Say $T = T_v$ is a MST on $V \backslash \{v\}$ and T' is a subgraph on $V \backslash \{u\}$, for some $v, u \in V$. Since Kruskal gives an MST T_u on $G - u$, we know $cost(T_u) \leq cost(T')$. But then the algorithm picked T_v over T_u, so

$$cost(T') > cost(T_v) \geq cost(T_u) \geq cost(T'),$$

contradiction. Hence, T is optimal. □

Problem 2

Suppose you are managing a computer network of n computing sites. Between some pairs of computing sites, there is a link that has a positive initialization time. When you turn on the entire network, all links start initializing at the same time. The whole network is not operational until all links have been initialized. You wish to remove links so that the network stays connected yet you have minimized the maximum initializion time.

The network is given to you as a connected undirected graph with positive edge weights. You can assume that $|E| = O(|V|)$.

Design an algorithm that achieves this goal.

Proof. Consider running Kruskal's algorithm on the network graph and return the resulting MST T.

We already know Kruskal's algorithm takes $O(|E| \log |V|)$ time. But since $|E| = O(V)$, this algorithm only takes $O(|V| \log |V|)$ time.

We now give a justification for the correctness of the algorithm, particularly the optimality of T. Let e be the last edge added to T. Say e has weight $w(e)$. We know e is the heaviest edge in T. Consider $T - \{e\}$. Since T is a tree, $T - \{e\}$ has two components, which splits the vertex set V into two subsets, say X and $V - X$. Note that by the choice of Kruskal's algorithm, e is the lightest edge which connects X and $V - X$. But then any spanning graph of G must contain an edge which connects X and $V - X$. Hence, $w(e)$ is a lower bound of initialization time of any connected network, which is achieved by T. □

Problem 3

You are given two sets of n points each on the number line: $(A[1], \dots, A[n])$ and $(B[1], \dots, B[n])$ (all values are different and each list is sorted in increasing order.)

You wish to pair up the points (one from the first list and one from the second list):

$$[(A[1], B[i_1]), (A[2], B[i_2]), \dots, (A[n], B[i_n])]$$

(such that each point is in exactly one pair.)

You wish to pair them up in such a way to minimize:

$$\sum_{k=1}^{n} |A[k] - B[i_k]|$$

(a) Describe this problem as we have done in class in terms of:

- **Input:** Arrays A, B of size n in increasing order and all values are distinct.
- **Solution Format:** $[(A[1], B[i_1]), (A[2], B[i_2]), \dots, (A[n], B[i_n])]$.
- **Constraints:** $i_\alpha \neq i_\beta$ if $\alpha \neq \beta$.
- **Objective:** Minimize cost value $\sum_{k=1}^{n} |A[k] - B[i_k]|$.

(b) **Candidate Greedy Strategy I:** Find the pair $A[i], B[j]$ that is the closest (with the smallest overall $|A[i] - B[j]|$ distance) and pair $(A[i], B[j])$ (break ties by choosing the smaller value of $A[i]$.) Remove $A[i]$ from the first list and remove $B[j]$ from the second list and repeat on the remaining points until all points are paired.

Either prove that this strategy always yields an optimal solution or give a counterexample to show that it is not always optimal.

Proof. Consider $A = [1, 4]$, $B = [-3, 2]$. The algorithm would first pair up 1 and 2, then 4 and -3. Hence, the algorithm would return $x = [(1, 2), (4, -3)]$, which has a cost value of 8. But then there exists a pairing $y = [(1, -3), (4, 2)]$, which has a cost value of 6. Hence, this strategy is not optimal. □

(c) **Candidate Greedy Strategy II:**

Pair up $(A[1], B[1]), (A[2], B[2]), \dots, (A[n], B[n])$.

Either prove that this strategy always yields an optimal solution or give a counterexample to show that it is not always optimal.

Proof. This algorithm is correct. Let σ be a non-trivial permutation of B. There exists some i such that $\sigma(i) \neq i$. Let k be the smallest such index. We know $\sigma(k) > k$. Let $l = \sigma^{-1}(k)$. Since k is the smallest index which deviates, l must be larger than k.

We now show that

$$|A[k] - B[\sigma(k)]| + |A[l] - B[k]| \geq |A[k] - B[k]| + |A[l] - B[\sigma(k)]|. \tag{1}$$

Since $A[k] < A[l]$ and $B[k] < B[\sigma(k)]$, there are *only* 6 possible cases:

Case 1: $A[k] \leq A[l] \leq B[k] \leq B[\sigma(k)]$.

$$(B[\sigma(k)] - A[k]) + (B[k] - A[l]) = (B[k] - A[k]) + (B[\sigma(k)] - A[l]).$$

Case 2: $A[k] \le B[k] \le A[l] \le B[\sigma(k)]$.

$$\begin{aligned}(B[\sigma(k)] - A[k]) + (A[l] - B[k]) &= (B[k] - A[k]) + (B[\sigma(k)] - A[l]) + 2(A[l] - B[k]) \\ &\ge (B[k] - A[k]) + (B[\sigma(k)] - A[l])\end{aligned}$$

Case 3: $A[k] \le B[k] \le B[\sigma(k)] \le A[l]$.

$$\begin{aligned}(B[\sigma(k)] - A[k]) + (A[l] - B[k]) &= (B[k] - A[k]) + (A[l] - B[\sigma(k)]) + 2(A[l] - B[\sigma(k)]) \\ &\ge (B[k] - A[k]) + (B[\sigma(k)] - A[l])\end{aligned}$$

Case 4: $B[k] \le A[k] \le A[l] \le B[\sigma(k)]$.

$$\begin{aligned}(B[\sigma(k)] - A[k]) + (A[l] - B[k]) &= (A[k] - B[k]) + (B[\sigma(k)] - A[l]) + 2(A[l] - A[k]) \\ &\ge (B[k] - A[k]) + (B[\sigma(k)] - A[l])\end{aligned}$$

Case 5: $B[k] \le A[k] \le B[\sigma(k)] \le A[l]$.

$$\begin{aligned}(B[\sigma(k)] - A[k]) + (A[l] - B[k]) &= (A[k] - B[k]) + (A[l] - B[\sigma(k)]) + 2(B[\sigma(k)] - A[k]) \\ &\ge (B[k] - A[k]) + (B[\sigma(k)] - A[l])\end{aligned}$$

Case 6: $B[k] \le B[\sigma(k)] \le A[k] \le A[l]$.

$$(A[k] - B[\sigma(k)]) + (A[l] - B[k]) = (A[k] - B[k]) + (A[l] - B[\sigma(k)])$$

Hence, by (1), we may achieve a lower objective value by the value of $\sigma(k)$ and $\sigma(l)$. But then a lower objective value can be obtained whenever the permutation is non-trivial. Therefore, by inductively applying the swapping process to the smallest deviating index starting from 1, σ eventually becomes a trivial permutation, and thus

$$\sum_{k=1}^{n} |A[k] - B[k]| \le \sum_{k=1}^{n} |A[k] - B[\sigma(k)]|.$$

□

(d) **Candidate Greedy Strategy III:** Let $B[j]$ be the closest point to $A[1]$ in the B list (break ties by choosing the lower B value). Pair up $(A[1], B[j])$ and remove $A[1]$ and $B[j]$ from the lists and continue with $A[2]$ until all points are paired.

Either prove that this strategy always yields an optimal solution or give a counterexample to show that it is not always optimal.

Proof. Consider $A = [1, 4]$, $B = [-3, 2]$. The algorithm would first pair up 1 and 2, then 4 and -3. Hence, the algorithm would return $x = [(1, 2), (4, -3)]$, which has a cost value of 8. But then there exists a pairing $y = [(1, -3), (4, 2)]$, which has a cost value of 6. Hence, this strategy is not optimal. □

Problem 4

Suppose you are driving along a road in an electric car. The battery of the electric car can bring you $x[0]$ miles. There are battery stations along the way at positive positions $D[1], \dots D[n]$ (in sorted order.) Each battery station can *replace* your battery and give you a new battery that can bring you a certain number of miles. The distances of the batteries are given in the array $x[0], x[1], \dots, x[n]$

You wish to start at position 0 with a full battery and end at position $D[n]$ replacing the fewest batteries.

(a) Describe this problem as we have done in class in terms of:

- **Input:** A sorted list of n battery station positions and a list of $n+1$ distances of the batteries.
- **Solution Format:** An increasing list of indices $X = \{x_1, \dots, x_k\}$.
- **Constraints:** Let $x_0 = 0$. For all $x_i \in X$,

$$x[x_{i-1}] + D[x_{i-1}] \geq D[x_i].$$

- **Objective:** Reach $D[n]$ and minimize $|X|$.

(b) **Candidate Greedy Strategy I:** Travel to the farthest battery station without exceeding $x[0]$ miles. Replace the battery at that station and repeat the process starting from that station until you can reach position $D[n]$.

Either prove that this strategy always yields an optimal solution or give a counterexample to show that it is not always optimal.

Proof. Consider $D = [0, 1, 3, 6]$, $x = [3, 5, 1, 0]$. The strategy would traval to station 2 first and replace to a battery with distance $x[2] = 1$. But then the next station is 3 unit distances away, so the strategy fails to arrive at the destination. However, there exists solution $X = \{1, 3\}$ which could reach the destination, and thus the strategy is not optimal. □

(c) **Candidate Greedy Strategy II:**

Travel to the battery station with the largest $x[i]$ value without exceeding $x[0]$ miles. Replace the battery at that station and repeat the process starting from that station until you can reach position $D[n]$ and then go directly there.

Either prove that this strategy always yields an optimal solution or give a counterexample to show that it is not always optimal.

Proof. Consider $D = [0, 1, 3, 4]$ and $x = [3, 2, 1, 0]$. The strategy would first stop at station 1 then station 2 and finally arrive at the destination. However, there exists solution $X = \{2, 3\}$, which reaches the destination with lesser stops, so this strategy is not optimal. □

Problem 4 continued on next page. . .

(d) **Candidate Greedy Strategy III:**

Travel to the battery station with the largest $D[i] + x[i]$ value (in other words, the battery that can take you the farthest down the road.) Replace the battery at that station and repeat the process starting from that station until you can reach position $D[n]$ and then go directly there.

Either prove that this strategy always yields an optimal solution or give a counterexample to show that it is not always optimal.

Proof. This strategy is correct. We show that the solution $X = \{x_1, \dots, x_k\}$ given by the strategy is correct and optimal. Since the strategy picks the next *reachable* battery station with the largest $D[i] + x[i]$ value, the constraint is satisfied.

Assume that $D[n]$ is reachable from $D[0]$. Let $Y = \{y_1, \dots, y_m = n\}$ be any solution which reaches $D[n]$. Suppose for the sake of contradiction that $D[y_i] + x[y_i] > D[x_i] + x[x_i]$ for some i. Let l be the smallest such index. We then have

$$D[y_l] + x[y_l] > D[x_l] + x[x_l],$$
$$D[x_{l-1}] + x[x_{l-1}] \geq D[y_{l-1}] + x[y_{l-1}].$$

Suppose $y_l \geq x_l$. Then $D[y_l] \geq D[x_l]$. Since station y_l is reachable from station y_{l-1}, we have $D[y_l] \leq D[y_{l-1}] + x[y_{l-1}]$. But then

$$D[x_{l-1}] \leq D[x_l] \leq D[y_l] \leq D[y_{l-1}] + x[y_{l-1}] \leq D[x_{l-1}] + x[x_{l-1}].$$

Thus, station y_l is between $D[x_{l-1}]$ and $D[x_{l-1}] + x[x_{l-1}]$, so the choice of the strategy yields $D[y_l] + x[y_l] \leq D[x_l] + x[x_l]$. But then this contradicts our assumption that $D[y_l] + x[y_l] > D[x_l] + x[x_l]$.

Hence, we know $x_{j-1} \leq y_l < x_j$ for some $j \leq l$. But then station y_l is between $D[x_{j-1}]$ and $D[x_{j-1}] + x[x_{j-1}]$, so $D[y_l] + x[y_l] \leq D[x_j] + x[x_j]$, by the choice of the algorithm. Notice that $D[x_i] + x[x_i] \geq D[x_{i-1}] + x[x_{i-1}]$ for all i, as the worst case we may just stay at station x_{i-1}. But then combining the inequalities,

$$D[x_l] + x[x_l] < D[y_l] + x[y_l] \leq D[x_j] + x[x_j] \leq D[x_l] + x[x_l],$$

contradiction. Hence, $D[x_i] + x[x_i] \geq D[y_i] + x[y_i]$ for all i, which makes X a valid solution as Y is able to reach station n. But then $D[x_{k-1}] + x[x_{k-1}] \geq D[y_{k-1}] + x[y_{k-1}]$, so X reaches n in no more stops than Y. Therefore, $|X| \leq |Y|$, and this completes the proof. □