

# **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  $w : E \rightarrow \mathbb{R}^+$ . 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  $\infty$ . For each vertex  $v \in V$ , run Kruskal's algorithm on  $G - v$  and get  $T_v$ . If  $T_v$  has exactly  $n$  edges  $\text{cost}(T_v) \leq \text{cost}(T)$ , 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 connectedness 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  $\text{cost}(T') < \text{cost}(T)$ . Say  $T = T_v$  is a MST on  $V \setminus \{v\}$  and  $T'$  is a subgraph on  $V \setminus \{u\}$ , for some  $v, u \in V$ . Since Kruskal gives an MST  $T_u$  on  $G - u$ , we know  $\text{cost}(T_u) \leq \text{cost}(T')$ . But then the algorithm picked  $T_v$  over  $T_u$ , so

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

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 initialization 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. Since Kruskal's algorithm furnishes a connected tree which spans the network, it remains to show 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  $T$ .  $\square$

## 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:** Two lists  $(A[1], \dots, A[n]), (B[1], \dots, B[n])$  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

$$C([(A[1], B[i_1]), \dots, (A[n], B[i_n])]) = \sum_{k=1}^n |A[k] - B[i_k]|$$

(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.  $\square$

(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.

**ExArg Claim:** Given input  $A, B$ , suppose  $OS$  is a valid solution which doesn't pair  $A[1]$  with  $B[1]$ . We show that there exists a better solution  $OS'$  which pairs up  $A[1]$  and  $B[1]$  but with lower cost.

**ExArg Proof:** Suppose  $OS$  pairs  $A[1]$  with  $B[k]$  and  $A[l]$  with  $B[1]$ . Let

$$OS' \cup \{(A[1], B[1]), (A[l], B[k])\} \setminus \{(A[1], B[k]), (A[l], B[1])\}.$$

We now show that

$$|A[1] - B[k]| + |A[l] - B[1]| \geq |A[1] - B[1]| + |A[l] - B[k]|. \quad (1)$$

Since  $A[1] < A[l]$  and  $B[1] < B[k]$ , there are *only* 6 possible cases:

**Case 1:**  $A[1] \leq A[l] \leq B[1] \leq B[k]$ .

$$(B[k] - A[1]) + (B[1] - A[l]) = (B[1] - A[1]) + (B[k] - A[l]).$$

**Case 2:**  $A[1] \leq B[1] \leq A[l] \leq B[k]$ .

$$\begin{aligned} (B[k] - A[1]) + (A[l] - B[1]) &= (B[1] - A[1]) + (B[k] - A[l]) + 2(A[l] - B[1]) \\ &\geq (B[1] - A[1]) + (B[k] - A[l]) \end{aligned}$$

**Case 3:**  $A[1] \leq B[1] \leq B[k] \leq A[l]$ .

$$\begin{aligned} (B[k] - A[1]) + (A[l] - B[1]) &= (B[1] - A[1]) + (A[l] - B[k]) + 2(A[l] - B[k]) \\ &\geq (B[1] - A[1]) + (B[k] - A[l]) \end{aligned}$$

**Case 4:**  $B[1] \leq A[1] \leq A[l] \leq B[k]$ .

$$\begin{aligned} (B[k] - A[1]) + (A[l] - B[1]) &= (A[1] - B[1]) + (B[k] - A[l]) + 2(A[l] - A[1]) \\ &\geq (A[1] - B[1]) + (B[k] - A[l]) \end{aligned}$$

**Case 5:**  $B[1] \leq A[1] \leq B[k] \leq A[l]$ .

$$\begin{aligned} (B[k] - A[1]) + (A[l] - B[1]) &= (A[1] - B[1]) + (A[l] - B[k]) + 2(B[k] - A[1]) \\ &\geq (A[1] - B[1]) + (A[l] - B[k]) \end{aligned}$$

**Case 6:**  $B[1] \leq B[k] \leq A[1] \leq A[l]$ .

$$(A[1] - B[k]) + (A[l] - B[1]) = (A[1] - B[1]) + (A[l] - B[k])$$

Hence, by (1),

$$C(OS) - C(OS') = |A[1] - B[k]| + |A[l] - B[1]| - |A[1] - B[1]| - |A[l] - B[k]| > 0,$$

and thus  $OS'$  achieves a lower cost value.

#### Induction Part:

We now show that the greedy solution is optimal for any input of size  $n \geq 1$  by induction on  $n$ . The base case is trivial, as there is only one possible solution. Suppose  $n \geq 2$ . Assume that the greedy strategy is optimal for all inputs of size less than  $n$ . Consider an input  $A, B$  of size  $n$ . Let  $OS$  be any arbitrary solution of  $A, B$ . By the Exchange Argument, there exists an as good or better solution  $OS'$  which pairs  $A[1]$  with  $B[1]$  and  $C(OS'(A, B)) \leq C(OS(A, B))$ . Let  $A' = A \setminus \{A[1]\}$  and  $B' = B \setminus \{B[1]\}$ . Since  $|A| = |B| < n$ , by induction,

$$\begin{aligned} C(OS(A, B)) &\leq C(OS'(A, B)) = C(\{(A[1], B[1])\} \cup S(A', B')) \\ &\geq C(\{(A[1], B[1])\} \cup GS(A', B')) \\ &= C(GS(A, B)). \end{aligned}$$

□

- (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.  $\square$

## 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:** Two lists  $(D[0], D[1], \dots, D[n]), (x[0], x[1], \dots, x[n])$ , where  $D$  is in sorted order. 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  $X = [x_1, \dots, x_k]$ , where  $1 \leq x_i \leq n$ .
- **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 travel 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.  $\square$

(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.  $\square$

(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 argument is correct.

**ExArg Claim:** Given input  $D$  and  $x$ , Let  $OS = [y_1, \dots, y_m]$  be some solution which contains some index  $y_i$  such that station  $y_i$  does not have the largest  $D[y_i] + x[y_i]$  value reachable from station  $y_{i-1}$ . Let  $OS' = [z_1, \dots, z_k]$  be another solution such that  $y_j = z_j$  for all  $j < i$  but  $z_i$  is the reachable battery station from  $z_{i-1}$  that can take you the farthest down the road. Then,  $z_k \geq y_m$  and  $k \leq m$ .

**ExArg Proof:** Let  $y_i$  be the smallest index in  $OS$  such that station  $y_i$  does not have the largest  $D[y_i] + x[y_i]$  value reachable from station  $y_{i-1}$ . Construct  $OS' = [z_1, \dots, z_k]$  such that  $y_j = z_j$  for all  $j \neq i$  and make  $z_i$  the station with the largest  $D[s] + x[s]$  value reachable from station  $z_{i-1}$ . Replace all stations in  $OS'$  after  $z_i$  with  $n$  if  $D[z_i] + x[z_i] \geq D[n]$ . Note that since  $D[z_i] + x[z_i] > D[y_i] + x[y_i]$ , station  $y_{i+1}$  is reachable from station  $z_i$ , and thus  $OS'$  is a valid solution.

If  $D[z_i] + x[z_i] \geq D[n]$ , then obviously  $z_k \geq y_m$  and  $k \leq m$ . Otherwise,  $OS$  and  $OS'$  agree with all terms except for the  $i$ th one, so obviously  $z_k \geq y_m$  and  $k \leq m$ . Hence,  $OS'$  is at least as good a solution as  $OS$ .

**Induction Part:**

We now show that for any input where  $n \geq 1$ , the greedy solution is optimal. When  $n = 1$ , there is only one possible solution, so the base case is done.

Suppose  $n \geq 2$ . Assume that the greedy strategy is optimal for all valid inputs of size less than  $n$ . Given input  $(D[0], D[1], \dots, D[n])$  and  $(x[0], x[1], \dots, x[n])$ , let  $OS = [y_1, \dots, y_m]$  be any solution. We may assume that some stations in  $OS$  does not have the station with the largest  $D + x$  value reachable from the previous station, otherwise  $OS = GS$  and we are done. Let  $y_i$  be the first such. The argument at the start yields a just as good or better solution  $OS' = [z_1, \dots, z_k]$ . Let  $D'$  be the list of the first  $z_i + 1$  entries of  $D$  and let  $D''$  be the list such that  $D''[k] = 0$  and  $D''[k] = D[z_i + k] - D[z_i + k - 1]$  for  $1 \leq k \leq n - z_i$ . Additionally, let  $x'$  be the list of the first  $z_i + 1$  entries of  $x$  and let  $x''$  be the rest of  $x$ . Then, by the construction of  $OS'$  and induction,

$$\begin{aligned} |OS(D, x)| &\geq |OS'(D, x)| = |GS(D', x') \circ S(D'', x'')| \\ &\geq |GS(D', x') \circ GS(D'', x'')| \\ &= |GS(D, x)|. \end{aligned}$$

Define  $last(L)$  to be the last element of a list  $L$ . Then, again by the construction of  $OS'$  and induction

$$\begin{aligned} last(OS(D, x)) &\leq last(OS'(D, x)) = last(S(D'', x'')) + z_i \\ &\leq last(GS(D'', x'')) + z_i \\ &= last(GS(D, x)). \end{aligned}$$

□