Algorithms #7
Jose Nino

Problem 1

a) This statement is true

For sake of contradiction let T be an MST for graph G and assume T is <u>not</u> an MST for graph G' (where w_e^2).

Then it must be that at least one edge e, which $e \in T$ $e \notin T'$ (MST for graph G'), an it was replaced by edge e', which $e' \notin T$ but $e' \in T'$. So it must be that $w_e^2(e') < w_e^2(e)$

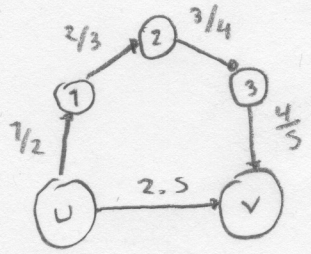
Thus this assumption implies that $\exists\ e, e'$ for which $w_e^2(e') < w_e^2(e)$ and

$w_e(e) < w_e(e')$

$\Rightarrow\Leftarrow$

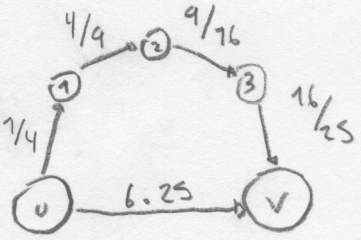
Because $f(x) = x^2$ is a linear function for which if $x < y$ then $f(x) < f(y)$. So it must be that after squaring T is still an MST of G'

b) This is false. Let me provide a counter example.



in the original graph the shortest path from u to v is the edge (u, v) which has a weight of 2.5

Squared Graph



Now the shortest path from u to v is $u \to 1 \to 2 \to 3 \to v$ which adds to be about 1.88

Algorithms # 7

Problem

2a) Lets prove this by induction. if Boruvka has edges at a certain step in the algorithm then these edges belong in the MST.

Base case: this is trivial as at the start there are no edges in the forest so trivially the are contained in the MST.

Inductive hypothesis: Assume that at step k we have a forest with a set of edges that belong in the MST.

Inductive Step: want to show that the edges added in step k+1 belong in the MST.

for sake of contradiction

let T be the minimum spanning tree of the graph, assume that edge e' which is added in step k+1 does not belong in T.

let P be the path in T that goes from the two vertices of e'. The ∃ edge e that is in P that has not being chosen by the algorithm yet. Because it didn't chose it and all values are distinct it must be that $w(e') < w(e)$. But then this means that the path from the two vertices connected by e' is cheaper than P in T that is T+e'-e has lower weight than T

$\Rightarrow\Leftarrow$

Because then T is not the MST of 6. So it must be that the edges added at every step must be part of the MST of 6.

Altogether this means that when the algorithm has finishing adding edges to the forest we end up with the MST.

2b To prove that every step (iteration of boruvka) runs in $O(m)$ we need to show that 3 operations run in $O(m)$

(1) Use a simple DFS to find all connected components of the current forest. Run a DFS over graph G, everytime that a DFS ends and starts from an unvisited vertex mark the nodes with a different component name. That is start with component a and run a DFS from an unvisited node when this DFS is over move to the next unvisited vertex but this time mark node with component name b. Notice that here a DFS can run only on marked edges, not on unmarked edges. This way the DFS will have to jump to an unvisited node when all nodes reachable from the start node via marked edges have been visited. If we didn't apply this constraint then all nodes would be reachable by any starting node and we would only have 1 component every time. Because we are running this special DFS on the graph this takes time $O(n+m)$ but this graph is connected so $O(n) = O(m)$ and thus $O(n+m) = O(m)$.

(2) Now that we have C connected components initialize the minimum of each component to have weight (∞) and be the edge $(\varepsilon, \varepsilon)$ edge Assuming we have a list of all edges in the graph. for each edge compare its weight to the minimum weight of the component of each of its two vertices, if the two vertices belong to different components. If it is $<$ the current minimum of either component make this edge the minimum of the respective component. Thus for every edge we are doing a constant amount of work. 1) Three comparisons: compare that the two vertices are of different components, compare its weight to the weight of both components. 2) Maximum of two assignments, if this edge happens to be smaller than the current minimum. Therefore, this step which iterated over all edges runs in $O(m)$

(3) Mark all new minimal edges. This takes time of the number of components because for every component we mark its minimal edge and add it to the set of marked edges (keep in mind that we might add the same edge twice, but it doesn't matter because we are keeping them in a set). Assymtotically this runs in $O(\text{number of components})$ which is $O(n)$ when we start but because the graph is connected $O(n)$ is $O(m)$.

Therefore each iteration of the algorithm runs in $O(m)$ time

Problem 3c

In part b we proved that each iteration of the algorithm runs in $O(m)$ time. To prove an upper bound of $O(m \log n)$ for the algorithm we need to show that we will have $O(\log n)$ iterations.

This is not that difficult to see. On every iteration for each edge we add we are connecting to components in to 1 component. Therefore at minimum every iteration we add edges that are reducing the amount of components ins the forest by ΔT least half.

Therefore because we run the algorithm untill we have only 1 component, if we started with n coponents (at the begginig every node is a componet of the forest) and on every iteration we are at least halving the the amount of corponents, then we can run a maximum of $\log n$ iterations.

Altogether this means that if we run $O(\log n)$ iteratrons each taking $O(m)$ time, then the total running time is $O(m \log n)$.

Problem 3

a) Monotone: the prove is straight forward

let ① $w: U \to \mathbb{R}_{\geq 0}$

② $f(s) = \sum_{e \in s} w(e)$

③ $S, T \subseteq U$ where $S \subseteq T$

Then by ③ we know that S has up to all the elements in T.

if ⓐ S has all the elements in T it is clear that $f(S) = f(T)$.

ⓑ we remove elements of S it holds $S' \subseteq T$. Moreover because $w: U \to \mathbb{R}_{\geq 0}$ then by removing elements from S to make S' we have that if we remove e from S then

$f(S') = f(S) - w(e) \leq f(S) = f(T)$ (by ⓐ above)

so $f(S') \leq f(T)$

so clearly $\forall\, S, T \subseteq U$ where $S \subseteq T$ $f(S) \leq f(T)$ and f is therefore monotone

Submodular: For function f we have that for sets S and $T \subseteq U$

$$f(S) = \sum_{a \in S} w(a)$$

$$f(T) = \sum_{b \in T} w(b)$$

Now note that $f(S) + f(T)$ is adding all the weights of all the elements that pertain to S and all the elements that pertain to T therefore it is adding the weights of all the elements that pertain exclusively(x) to one of the sets once and the weights of all elements that pertain to both of the sets (y) twice, once for each set.

Notice that $S \cup T$ contains all elements so $\sum_{c \in S \cup T} w(c)$ would count the weights of all elements once. And $S \cap T$ contains all elements pertaining to both sets so $\sum_{d \in S \cap T} w(d)$ would count this elements an additional time.

So $\sum_{c \in S \cup T} w(c) + \sum_{d \in S \cap T} w(d)$ counts the same weights that $\sum_{a \in S} w(a) + \sum_{b \in T} w(b)$ did.

so $\underbrace{\sum_{a \in S} w(a)}_{f(S)} + \underbrace{\sum_{b \in T} w(T)}_{f(T)} = \sum_{c \in S \cup T} w(c) + \sum_{d \in S \cap T} w(d)$

and $\uparrow f(S \cup T)$ notice that $\uparrow f(S \cap T)$

so for our function f

$$f(S) + f(T) = f(S \cup T) + f(S \cap T)$$

and thus f is submodular.

b) consider e which $e \notin T$ then because $e \notin T$ and $S \subseteq T$ then $e \notin S$. because if $e \in S$ then $e \in T$. so $e \notin T$, $e \notin S$

By submodularity we have $f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$

we want to prove $f(S \cup \{e\}) - f(S) \geq f(T \cup \{e\}) - f(T)$

Consider the sets $S \cup \{e\}$ and T both of them are $\subseteq U$.

so $f(S \cup \{e\}) + f(T) \geq f(S \cup \{e\} \cup T) + f((S \cup \{e\}) \cap T)$

① $S \cup \{e\} \cup T$ -> we have that $S \cup T$ is T because $S \subseteq T$
so $S \cup \{e\} \cup T = T \cup \{e\}$.

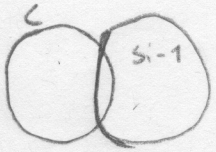
② $(S \cup \{e\}) \cap T = S$ because $S \subseteq T$ and $\{e\} \notin S$, $\{e\} \notin T$

so $f(S \cup \{e\}) + f(T) \geq f(T \cup \{e\}) + f(S)$

$f(S \cup \{e\}) - f(S) \geq f(T \cup \{e\}) - f(T)$ ☒

c) $\sum_{a \in C \setminus S_{i-1}} f_{S_{i-1}}(a) \geq f(C) - f(S_{i-1})$

Suppose we have the following sets



and lets number the elements in $C \setminus S_{i-1}$ from $a_i, \ldots, a_k$.

Now let set T_{k-1} be $S_{i-1} \cup a_i \cup a_{i+1} \cup \ldots \cup a_{k-1}$, where a_1 is the current element that we are adding to T. Therefore, by the law of diminishing returns. as we add elements to T and T grows the marginal value of any given element decreases with respect to the value if had if we had added it to the "frozen" S_{i-1}. That is the marginal value of adding a_i right now to S_{i-1} is greater than or equal to the marginal value of adding a_i to T_{i-1} (or in other words after adding $a_1, a_2, \ldots, a_{i-1}$ to S_{i-1}).

Therefore we have that $f_{S_{i-1}}(a_i) \geq f_{T_{i-1}}(a_i)$

So we have that

$$\sum_{a \in C \setminus S_{i-1}} f_{S_{i-1}}(a) \geq f_{T_i}(a_i) + f_{T_{i+1}}(a_{i+1}) + \ldots f_{T_{k-1}}(a_k).$$

and we know that $f_{T_i}(a_i) = f(S_{i-1} \cup \{a_i\}) - f(S_{i-1})$

so the RHS of the inequality above is the sum

$$f(S_{i-1} \cup \{a_i\}) - f(S_{i-1}) + f(S_{i-1}) \cup \{a_i\} \cup \{a_2\}) - f(S_{i-1} \cup \{a_i\}) + \ldots$$
$$\ldots + f(S_{i-1} \cup C) - f(S_{i-1}) \cup \{a_i\} \cup \ldots \cup \{a_{k-1}\}) = f(S_{i-1} \cup C) - f(S_{i-1})$$

So we have that

$$\sum_{a \in C \setminus S_{i-1}} f_{S_{i-1}}(a) \geq f(S_{i-1} \cup C) - f(S_{i-1})$$

and because it is clear that $C \subseteq C \cup S_{i-1}$, then $f(C) \leq f(C \cup S_{i-1})$

so $\sum_{a \in C - S_{i-1}} t_{S_{i-1}}(a) \geq f(C \cup S_{i-1}) - f(S_{i-1}) \geq f(C) - f(S_{i-1})$ ∎

d) $t_{S_{i-1}}(a_i) \geq \frac{1}{k}(f(C) - f(S_{i-1}))$

Need to prove the above.

Notice that the greedy algorithm picked to add a_i to S_{i-1} because it gave the maximal marginal return.

Now in part c we saw the following sum $\sum_{a \in C \setminus S_{i-1}} t_{S_{i-1}}(a)$

dividing this sum by $k-(i-1)$ gives us the average maximal marginal gain if we add any given a to the static S_{i-1}. Because the greedy algorithm returned a_i from $U \setminus S_{i-1}$ then it must be that this value gives a marginal return by adding it to S_{i-1} that is greater than or equal to the average maximal marginal return of values in $C \setminus S_{i-1}$ because if this weren the case then the greedy algorithm would have choosen a different a from $U \setminus S_{i-1}$.

from c we have that $\sum_{a \in C \setminus S_{i-1}} t_{S_{i-1}}(a) \geq f(C) - f(S_{i-1})$ so if we multiply both sides by $\frac{1}{k-(i+1)}$ we get.

$$t_{S_{i-1}}(a_i) \geq \frac{1}{k-(i-1)}\left(\sum_{a \in C \setminus S_{i-1}} t_{S_{i-1}}(a)\right) \geq (f(C) - f(S_{i-1}))\frac{1}{k-(i-1)}$$

Now it is obvious that for any $i \geq 1$ (which is what we care about)

$k \geq k-(i-1)$ so $(f(C) - f(S_{i-1}))\frac{1}{k-(i-1)} \geq \frac{1}{k}(f(C) - f(S_{i-1}))$

$$\therefore \quad t_{S_{i-1}}(a_i) \geq \frac{1}{k}(f(C) - f(S_{i-1}))$$ ∎

d) we have the base case in the problem description. for $i-1$ assume by inductive hypothesys that $f(c) - f(s_{i-1}) \leq \left(1 - \frac{1}{k}\right)^{i-1} f(c)$

Starting from the equality

$$f(c) - f(s_i) = f(c) - f(s_{i-1}) - f_{s_{i-1}}(a_i)$$

from d we have that

$$f_{s_{i-1}}(a) \geq \frac{1}{k}\left(f(c) - f(s_{i-1})\right)$$

so $f(c) - f(s_i) = f(c) - f(s_{i-1}) - f_{s_{i-1}}(a_1)$

$$\leq f(c) - f(s_{i-1}) - \frac{1}{k}\left[f(c) - f(s_{i-1})\right]$$

$$\leq \left(1 - \frac{1}{k}\right) f(c) + \left(\frac{1}{k} - 1\right) f(s_{i-1})$$

$$= \left(1 - \frac{1}{k}\right) f(c) - \left(1 - \frac{1}{k}\right) f(s_{i-1})$$

$$= \left(1 - \frac{1}{k}\right)\left[f(c) - f(s_{i-1})\right]$$

Now we have from the inductive hypothesys that

$$f(c) - f(s_{i-1}) \leq \left(1 - \frac{1}{k}\right)^{i-1} f(c)$$

so $f(c) - f(s_i) \leq \left(1 - \frac{1}{k}\right)\left[f(c) - f(s_{i-1})\right]$

$$\leq \left(1 - \frac{1}{k}\right)\left[\left(1 - \frac{1}{k}\right)^{i-1} f(c)\right]$$

$$= \left(1 - \frac{1}{k}\right)^{i} f(c)$$

So, $f(c) - f(s_i) \leq \left(1 - \frac{1}{k}\right)^{i} f(c)$ ▨

BF

By induction we have proven that after k steps.

$$f(c) - f(s_k) \leq \left(1 - \frac{1}{k}\right)^k f(c)$$

Now we have that $\left(1 - \frac{1}{k}\right)^k \leq \frac{1}{e}$ for all $k \geq 1$

Since the 0 iteration is trivial because we have nothing in either C or S then we know that for any significant k ($k \geq 1$) then

$$\left(1 - \frac{1}{k}\right)^k f(c) \leq \frac{1}{e} f(c)$$

So $f(c) - f(s_k) \leq \left(1 - \frac{1}{k}\right)^k f(c) \leq \frac{1}{e} f(c)$

so it is clear that

$$f(c) - f(s_k) \leq \frac{1}{e} f(c)$$ ∎