Modern Algorithm Design (Monsoon 2024)
Homework 2

Deadline: 30th September, 2024, 11:59 pm (IST)

Release Date: 16th September, 2024

1. (*10 points*) This problem concerns running time optimizations to the Hungarian algorithm for computing minimum-cost perfect bipartite matchings. Recall the $O(mn^2)$ running time analysis from lecture: there are at most n augmentation steps, at most n price update steps between two augmentation steps, and each iteration can be implemented in $O(m)$ time.

   (a) By a phase, we mean a maximal sequence of price update iterations (between two augmentation iterations). The naive implementation in lecture regrows the search tree from scratch after each price update in a phase, spending $O(m)$ time on this for each of up to n iterations. Show how to reuse work from previous iterations so that the total amount of work done searching for good paths, in total over all iterations in the phase, is only $O(m)$.

   Proof. We modify the BFS procedure to ensure that in each iteration the total work done to compute augmenting path is $O(m)$. To achieve this, along with the BFS-queue Q, we maintain two boolean variables for each vertex $v \in V$, respectively $odd[v]$ and $even[v]$ such that,

   - Both $odd[v]$ and $even[v]$ are initialized as NULL, meaning that v has not been visited in the search tree yet during BFS traversal.
   - $odd[v]$ = TRUE when v has been visited at an odd level in the BFS tree, i.e. has been visited as an odd vertex.
   - $even[v]$ = TRUE when v has been visited at an even level in the tree, i.e. has been visited as an even vertex.

   We proceed with the BFS as usual. Once we get stuck and update the vertex prices, we get a set S of the new tight edges. For each edge $(u, v) \in S$, one of the following cases can arise

   - Neither u nor v have been visited. In this case, we discard (u, v).
   - $even[u]$ = TRUE. In this case, we set $odd[v]$ = TRUE and run our modified BFS from v if $odd[u]$ = TRUE.
   - Only $odd[u]$ = True. Here, we do not visit v since we only select the matched edges for odd vertices.
   - In all other cases, discard (u, v).

   During our modified BFS, if we ever encounter a vertex v at an odd level with $odd[v]$ = True already, or a vertex v at an even level with $even[v]$ = True already, we do not expand it further. This is because revisiting a vertex with the same parity does not lead to a new path to an unvisited vertex.

   Now, each edge (u, v) on the search tree is visited at most twice, once when u is being visited as an odd vertex an once when even. Thus, the total work done searching for good paths over all the $O(n)$ iterations of one phase is now $O(m)$. This obviously excludes the work done to compute δs and update prices.

   □

   (b) The other non-trivial work in a price update phase is computing the value of δ (the minimum magnitude by which the prices can be updated without violating any of the invariants). This is easy to do in $O(m)$ time per iteration. Explain how to maintain a heap data structure so that the total time spent computing δ over all iterations in the phase is only $O(m \log n)$. Be sure to explain what heap operations you perform while growing the search tree and when executing a price update.

   [This yields an $O(mn \log n)$ time implementation of the Hungarian algorithm.]

Proof. To compute the value of δ for all price updates in a phase in $O(m \log n)$ time, we now also maintain a min-heap H_i for each phase i, along with the BFS-queue Q and the vertex properties. The heap is structured as [Priority, Item].

$$H_i = \{[\widehat{w}_{uv}, (u, v)] \mid \widehat{w}_{uv} = w_{uv} - p_u - p_v \neq 0 \text{ and } v \text{ is at an even level}\}$$

Heap H_i is built incrementally during the search. While doing BFS, when we encounter an non-tight edge $e = (u, v)$ with v at an even level, we insert $[\widehat{w}_{uv}, (u, v)]$ to H_i. This ensures that the heap only contains edges of where u is at an odd level and v is at an even level. As expected, we get the δ for a price update by

$$[\delta, \_] \leftarrow EXTRACT - MIN(H_i)$$

The time complexity of each extraction depends on the total number of elements k in the heap. There can be at most $m \leq n^2$ elements in the heap during the phase 1. So, the extraction time is proportional to $\log k$, and $\log n^2 = 2 \log n = O(\log n)$. Finally, there can be at most $O(m)$ inserts during the search tree expansion and at most $O(m)$ extractions 2 overall during the phase.

Thus, the total time spent in computing δs over the entire phase is $O(m \log n)$. This yields an $O(mn \log n)$ Hungarian Algorithm, as there can be at most $O(n)$ phases, and each phase can be implemented in $O(m + m \log n)$ time. □

2. (*10 points*) We used low-stretch trees to approximate APSP on general graphs. We explore this connection further via the k-point-facility problem: Given a graph $G = (V, E)$ with positive edge lengths and $k \in \mathbb{Z}_{\geq 0}$. Define $d_G(u, v)$ to be the length of the shortest path between u and v in G according to these edge-lengths. For a set $C \subseteq V$, define

$$d_G(v, C) := \min_{c \in C} d_G(v, C)$$

The k-point-facility problem asks you to find a set $C \subseteq V$ with $|C| = k$ to minimize $\Phi_G(C) := \sum_{v \in V} d_G(v, C)$.

(a) **(3 points)** Design a poly(k, n) time algorithm to solve the k-point-facility problem on an edge-weighted path of n vertices.

Proof. We use DP to solve the problem. Consider we order the vertices on the path from left to right and assign names $v_1, v_2, \cdots, v_n$ according to the order. Now our dynamic programming solution scans through the vertices in the order and at any point of time i, it computes the cost of considering v_i as a center and the cost of not considering v_i as a center. Following this, two the subproblems are defined.

Subproblem definitions: **(+1 for correct subproblem)**

$center(i, j)$: The sub-problem outputs the cost of i facilities covering first j points and v_j is included in the facility centers.

$non\_center(i, j)$: The sub-problem outputs the i facilities covering the first j points and v_j is not included in the facility centers.

Base case: **(+1 for correct base case)**

We consider the case when there is exactly one center that is $i = 1$.

$$center(1, j) = \sum_{u=1}^{j} d_G(v_u, v_j)$$

$$non\_centers(1, j) = \begin{cases} \infty & if\ j = 1 \\ \min_{1 \leq l \leq j-1} \{center(1, l) + \sum_{m=l}^{j} d_G(v_m, v_l)\} & 2 \leq j \leq n \end{cases}$$

Recurrence: **(+1 for recurrence and the sub-problem which gives the final solution) (deducted 1.5 out of 3 if either one of the sub-problem is not considered by your recurrence)**

Here $i \in \{2, \cdots, k\}$. Fixing each i, we compute the following recurrences for each $j \in 1, \cdots, n$.

$$center(i, j) = \min_{i-1 \leq l \leq j} \left\{ center(i-1, l) + \min \left\{ \sum_{x=l}^{m} d_G(v_x, v_l) + \sum_{y=m+1}^{j} d_G(v_y, v_j) \mid l \leq m \leq j-1 \right\} \right\}$$

$$non\_center(i,j) = \min_{i \le l \le j-1} \left\{ center(i,l) + \sum_{x=l+1}^{j} d_G(v_x, v_l) \right\}$$

Subproblem that solves the actual problem :

$$\min\{center(k,n), non\_center(k,n)\}$$

Running time:
The running time of the dynamic program is $O(kn^2)$. □

(b) **(2 points)** Given an algorithm to solve k-point-facility optimally on trees, show that the algorithm that samples a tree T from (random) low-stretch tree distribution with stretch α, solves k-point-facility on T to get C_T, and outputs this set C_T , ensures that the expected cost $\mathbb{E}_T[\Phi_G(C_T)] \le \alpha \cdot OPT$.

Proof. **(Score deduction as per step jumps in the calculation)** Let A be an algorithm to solve k-point-facility optimally on trees, C be the optimum set of centers, (that is $C \in \arg\min_{B \subseteq V} \sum_{v \in V} d_G(v,B)$) and D be an α-stretch tree distribution,

$$\begin{aligned}
\mathbb{E}_T[\Phi_G(C_T)] &= \mathbb{E}_T\left[\sum_{v \in V} d_G(v, C_T)\right] \\
&= \sum_{v \in V} \mathbb{E}_T[d_G(v, C_T)] \\
&= \sum_{v \in V} \mathbb{E}_T\left[\min_{c \in C_T} d_G(v,c)\right] \\
&\le \sum_{v \in V} \mathbb{E}_T\left[\min_{c \in C_T} d_T(v,c)\right] \quad \left[\because d_G(v,c) \le d_T(v,c) \forall c \in C_T \implies \min_{c \in C_T} d_G(v,c) \le \min_{c \in C_T} d_T(v,c)\right] \\
&= \sum_{v \in V} \mathbb{E}_T[d_T(v, C_T)] = \mathbb{E}_T[\Phi_T(C_T)] \\
&\le \mathbb{E}_T[\Phi_T(C)] \quad \left[\because \Phi_T(C_T) \le \Phi_T(C) \forall C \subseteq V \text{ with } |C| = k \text{ since } C_T \in \arg\min_{B \subseteq V} \sum_{v \in V} d_T(v,B)\right] \\
&= \sum_{v \in V} \mathbb{E}_T[d_T(v, C)] = \sum_{v \in V} \mathbb{E}_T[\min_{c \in C} d_T(v,c)] \\
&\le \sum_{v \in V} \min_{c \in C} \mathbb{E}_T[d_T(v,c)] \quad \left[\because \forall c \in C, \min_{c \in C} d_T(v,c) \le d_T(v,c) \implies \mathbb{E}_T\left[\min_{c \in C} d_T(v,c)\right] \le \mathbb{E}_T[d_T(v,c)]\right] \\
&\le \sum_{v \in V} \min_{c \in C} \alpha d_G(v,c) \mathbb{E}_T[d_T(v,c)] \le \alpha d_G(v,c)) \\
&= \alpha \sum_{v \in V} \min_{c \in C} d_G(v,c) = \alpha \sum_{v \in V} d_G(v,C) = \alpha \Phi_G(C) = \alpha \cdot OPT
\end{aligned}$$

Thus we conclude $\mathbb{E}_T[\Phi_G(C_T)] \le \alpha \cdot OPT$ □

(c) **(2 points)** Show that if you perform $L := O(\frac{\log n}{\epsilon})$ independent runs of the above algorithm to get sets $C_1, C_2, \cdots, C_L$, and return the set with the least $\Phi_G(C_i)$ value from among these (call it C^*), then $\mathbb{P}r[\Phi_G(C^\star) > (1+\epsilon)\alpha \cdot OPT] \le 1/\text{poly}(n)$.

Proof. **(Score deduction as per step jumps in the calculation)** Let $C^* = \min_{i \in [L]} \Phi_G(C_i)$ and $L := O(\frac{\log n}{\epsilon})$, where $\epsilon > 0$ and $[L]$ implies the set $\{1, \cdots, L\}$,

$$\begin{aligned}
\mathbb{P}r[\Phi_G(C^*) > (1+\epsilon)\alpha \cdot OPT] &= \mathbb{P}r\left[\min_{i\in[L]} \Phi_G(C_i) > (1+\epsilon)\alpha \cdot OPT\right] \\
&\leq \mathbb{P}r[\forall i \in [L]; \Phi_G(C_i) > (1+\epsilon)\alpha \cdot OPT] \\
&\leq \prod_{i=1}^{L} \mathbb{P}r[\Phi_G(C_i) > (1+\epsilon)\alpha \cdot OPT] \\
&\leq \prod_{i=1}^{L} \mathbb{P}r[\Phi_G(C_i) \geq (1+\epsilon)\alpha \cdot OPT] \\
&\leq \prod_{i=1}^{L} \frac{\mathbb{E}[\Phi_G(C_i)]}{(1+\epsilon)\alpha \cdot OPT} \ \ [by\ Markov's\ inequality] \\
&\leq \prod_{i=1}^{L} \frac{\alpha \cdot OPT}{(1+\epsilon)\alpha \cdot OPT} \ \ [from\ the\ solution\ of\ 2(b)] \\
&= \prod_{i=1}^{L} \frac{1}{(1+\epsilon)} = (1+\epsilon)^{-O(\frac{\log n}{\epsilon})} \\
&= 2^{-O(\frac{\log(1+\epsilon)\log n}{\epsilon})} = n^{-O(\frac{\log(1+\epsilon)}{\epsilon})} \\
&= 1/n^{O(1)} = 1/poly(n) \ \ (\forall \epsilon > 0, \log(1+\epsilon) \leq \log(e^{\epsilon}) \leq \epsilon)
\end{aligned}$$

□

(d) **(3 points)** Show that the expected weight of an α-stretch tree is at most $O(\alpha)$ times the MST.

Proof. Let the MST of G be M and D be an α-stretch tree distribution,
(+1 point for this) Over all $T \in D$,

$$\begin{aligned}
\mathbb{E}_T\left[\sum_{(u,v)\in M} d_T(u,v)\right] &= \sum_{(u,v)\in M} \mathbb{E}_T[d_T(u,v)] \\
&\leq \sum_{(u,v)\in M} \alpha d_G(u,v) \\
&\leq \alpha \sum_{(u,v)\in M} w(u,v) \\
&= \alpha \cdot cost(MST)
\end{aligned}$$

$\forall T \in D$, let $T_{u,v}$ be the shortest path between u and v in T.
(+1 for this argument) Every edge (x,y) in T occurs at least in $\cup_{(u,v)\in M} T_{u,v}$ since $(x,y) \in T_{x,\alpha}$ and $(x,y) \in T_{\beta,y}$ for some (x,α) and (β,y) in M (since M is a spanning tree)

$$\mathbb{E}_T\left[\sum_{(u,v)\in T} d_T(u,v)\right] \leq \mathbb{E}_T\left[\sum_{(u,v)\in M} d_T(u,v)\right] \leq \alpha \cdot cost(MST)$$

(+1 for the final computation) □

(e) Extend your algorithm from (a) to solve k-point-facility on an edge-weighted tree.
(Hint: first solve it on a binary tree. Then show how to reduce the problem to general trees)

3. (*20 points*) Given an undirected graph $G = (V, E)$ with edge weights w_e, a subgraph H is a *γ-distance emulator* with stretch $\gamma \geq 1$ if for every edge $(x,y) \in E$,

$$d_H(x,y) \leq \gamma \cdot d_G(x,y).$$

*γ-distance emulator*s are similar in spirit to low stretch spanning trees except that they might have many more edges than a spanning tree.

(a) Show that for all $x, y \in V$, even if (x, y) is not an edge, $d_G(x, y) \leq d_H(x, y) \leq \gamma \cdot d_G(x, y)$.

Proof. Both G and H are undirected connected graph. So, for any pair of vertices (x, y) even if there is no direct edge among them, there must exists a path between x and y. For any pair of vertices $(x, y) \in V$ let P_G and P_H denotes the path in G and H respectively.
For any edge $e \in E$, by the definition of H,

$$d_G(e) \leq d_H(e) \leq \gamma d_G(e)$$

When $(x, y) \in V$ does not have any direct edge, then first we show $d_G(x, y) \leq d_H(x, y)$.

$$d_G(x, y) = \sum_{e \in P_G} w(e) = \sum_{e \in P_G} d_G(e) \leq \sum_{e \in P_G} d_H(e) \leq d_H(x, y) \text{ (Holds due to triangle inequality)}$$

$$d_H(x, y) = \sum_{e \in P_H} w(e) = \sum_{e \in P_H} d_H(e) \leq \sum_{e \in P_H} \gamma d_G(e) \leq \gamma d_G(x, y) \text{ (Holds due to triangle inequality)}$$

□

Clearly, a trivial distance emulator is the graph itself, that is $H = G$, but ideally we want a much sparser H.

(b) *Construction 1.* Sample $t = 4 \log n$ trees $T_1, T_2, \cdots, T_t$ from an α-stretch (randomized) low-stretch *spanning* tree, i.e., a low-stretch tree whose distribution is on spanning trees of the graph with the same edge lengths. Let H be the union of all these edges.

i. Show that for any fixed edge $(x, y) \in E$,

$$\mathbb{P}r[d_H(x, y)] \geq 2\alpha d_G(x, y)] \leq 2^{-t}.$$

(Hint: for any single value of i, bound $\mathbb{P}r[d_{T_i}(x, y) \geq 2\alpha d_G(x, y)]$)

Proof. Let $\mathcal{T}_\alpha$ be the distribution of α-stretch spanning trees. Let T_i be the i-th tree of $\{T_1, T_2, \cdots, T_t\}$ sampled from $\mathcal{T}_\alpha$. Then, for any pair of vertices $(x, y) \in V$ we have,

$$\mathbb{E}_{T_i}[d_{T_i}(x, y)] \leq \alpha d_G(x, y)$$

Now, for a fixed tree T_i,

$$\begin{aligned}\mathbb{P}r[d_H(x, y) \geq 2\alpha d_G(x, y)] &\leq \frac{\mathbb{E}_{T_i}[d_{T_i}(x, y)]}{2\alpha d_G(x, y)} \text{ (Using Markov's inequality)} \\ &= \frac{1}{2}\end{aligned}$$

We need to find $\mathbb{P}r[d_H(x, y) \geq 2\alpha d_G(x, y)]$. Since H is the union of the edges $\{T_1, T_2, \cdots, T_i\}$, we need to find the probability that, $d_{T_i}(x, y) \geq 2\alpha d_G(x, y)$ holds $\forall i \in \{1, \cdots, t\}$. Thus,

$$\begin{aligned}\mathbb{P}r[d_H(x, y) \geq 2\alpha d_G(x, y)] &\leq \mathbb{P}r\left[\bigcap_{i=1}^{t} (d_{T_i}(x, y) \geq 2\alpha d_G(x, y))\right] \\ &= \prod_{i=1}^{t} \mathbb{P}r[d_{T_i}(x, y) \geq 2\alpha d_G(x, y)] \text{ (Due to independent sampling)} \\ &\leq \left(\frac{1}{2}\right)^t = 2^{-t}\end{aligned}$$

□

ii. Given that, for any graph, there always exists a low-stretch spanning tree distribution with stretch $\alpha = O(\log n \log \log n)$. Use this to show that with probability $1 - \frac{1}{n^2}$, the graph H is an $O(\log n \log \log n)$-*distance emulator* with $O(n \log n)$ edges.

Proof. First, let us show that the number of edges in H denoted by $E(H)$ is $O(n \log n)$. Since $E(H)$ is the union of the edges in $\{T_1, \cdots, T_t\}$, we have,

$$|E(H)| = \left| \bigcup_{i=1}^{t} E(T_i) \right| \leq \sum_{i=1}^{t} |E(T_i)| \leq t(n-1) \leq 4 \log n \cdot (n-1) = O(n \log n)$$

From 3(b)(i) we have that,

$$\mathbb{P}r[d_H(x,y) \geq 2\alpha d_G(x,y)] \leq 2^{-t}$$

We want to show that H is $O(\log n \log\log n)$ distance emulator, that is

$$d_H(e) \leq O(\log n \log\log n) d_G(e);\ \forall e \in E$$

We want to bound the probability of the above event. Thus,

$$\begin{aligned}
\mathbb{P}r[\forall e \in E;\ d_H(e) \leq 2\alpha d_G(e)] &\geq \mathbb{P}r\left[\bigcap_{e \in E} (d_H(e) \leq 2\alpha d_G(e))\right] \\
&= 1 - \mathbb{P}r\left[\bigcup_{e \in E} (d_H(e) \geq 2\alpha d_G(e))\right] \\
&\geq 1 - \sum_{e \in E} \mathbb{P}r[d_H(e) \geq 2\alpha d_G(e)] \text{ (Using union bound)} \\
&\geq 1 - \frac{|E|}{2^t} \\
&= 1 - \frac{|E|}{2^{4 \log n}} \\
&= 1 - \frac{|E|}{n^4} \\
&\geq 1 - \frac{n^2}{n^4} = 1 - \frac{1}{n^2}
\end{aligned}$$

Putting the value of $\alpha = O(\log n \log\log n)$ in $\mathbb{P}r[\forall e \in E;\ d_H(e) \leq 2\alpha d_G(e)]$, we get the desired result. □

(c) *Construction 2.* A simple greedy algorithm approach is following.

i. Define the *girth* of graph G to be smallest number of edges on any cycle in G. We first show that any graph G with m edges and n nodes, and girth *strictly more than* g must have $m \leq O(n + n^{1+1/\lfloor g/2 \rfloor})$.

A. The average degree of G is $\overline{d} := \frac{2m}{n}$. Show that there exists a subset $S \subseteq V$ such that the induced subgraph $H := G[S]$ has minimum degree at least $\overline{d}/2$.
[Hint: drop some low-degree vertices]

Proof. To prove this, as the hint suggests, we remove all the vertices $v \in V$ such that degree of those vertices v is less than $\overline{d}/2$, that is $deg(v) < \overline{d}/2$. We argue that the resulting graph is a sub-graph of G with minimum degree at least $\overline{d}/2$. Let G' be the resulting graph.
By construction, G' must have minimum degree $\overline{d}/2$. Now we argue that G' is not empty. For contradiction, let G' is empty. Then, each vertex of G had degree at most $\overline{d}/2$. Each time we remove a vertex, the number of edges are reduced by at most $\frac{\overline{d}}{2} - 1 = \frac{m}{n} - 1$. Thus, over all vertices we remove,

$$\sum_{v \in V} \frac{m}{n} - 1 = m - n$$

Since, $(m - n) < n$, that implies we did not remove all the edges form G. Thus G' is not empty. □

B. For this graph H and any vertex $v \in H$, show that the number of distinct vertices reachable within $\lfloor g/2 \rfloor$ hops from v is at least $(\overline{d}/2 - 1)^{\lfloor g/2 \rfloor}$.

Proof. Let us fix a vertex $v \in H$. We will prove the statement by induction on the number of hops from v. Let k denotes the k-th hop from v

Base case: When $k = 0$, with 0 hops vertices reachable form v is v itself, hence the statemnet holds.

Induction hypothesis: Let the statement be true for every $k < \lfloor g/2 \rfloor$.

Induction step: Now we show that the statement is true when $k = \lfloor g/2 \rfloor$. Observe that, form Q3(c)(i)(A) we know that any vertex has degree at least $(\overline{d}/2)$. Let u be a vertex at hop $(k-1)$ from v. Moreover, observe that there can not exists two distinct paths of length $k-1$ (otherwise, it contradicts the fact that girth is $> g$). So, at most one neighbor of u lies in hop $k-2$. Thus each vertex u at level $(k-1)$ contributes at least $(\overline{d}/2 - 1)$ vertices at next level k. By induction hypothesis, there are at least $(\overline{d}/2-1)^{k-1}$ vertices at level $(k-1)$. Hence There are at least $(\overline{d}/2-1)^{k-1+1} = (\overline{d}/2-1)^k$ vertices at level k. □

C. Prove that the number of edges m in the original graph G satisfies $m \leq O(n + n^{1+1/\lfloor g/2 \rfloor})$.

Proof. For a graph G with n vertices, m edges and girth g, $\exists S \subseteq V$ such that $H := G[S]$ has minimum degree $\geq \frac{m}{n}$ and within H, $|N^{\lfloor g/2 \rfloor}[v]| \geq (\frac{m}{n} - 1)^{\lfloor g/2 \rfloor}, \forall v \in H$ and the set of vertices reachable within $\lfloor g/2 \rfloor$ hops is $N^{\lfloor g/2 \rfloor}[v] = \{u \mid d_{uv} \leq \lfloor g/2 \rfloor\}$
Finally $\forall v \in H$, we have a trivial inequality $n \geq |N^{\lfloor g/2 \rfloor}[v]| \geq (\frac{m}{n} - 1)^{\lfloor g/2 \rfloor}$

$$n \geq \left(\frac{m}{n} - 1\right)^{\lfloor g/2 \rfloor} \implies \frac{m}{n} \leq 1 + n^{\frac{1}{\lfloor g/2 \rfloor}} \implies m \leq n + n^{1+\frac{1}{\lfloor g/2 \rfloor}}$$

□

ii. The algorithm is a variant of Kruskal's algorithm for $\alpha \geq 1$. Consider the edges of G in increasing order of lengths $e_1, e_2, \cdots, e_m$. Initialize $H_0 = \emptyset$. When considering edge $e_i = (x, y) \in E$, if the current distance $d_{H_{i-1}}(x, y) \leq \alpha d_G(x, y)$, then discard e (i.e., set $H_i \leftarrow H_{i-1}$), else take it (i.e., set $H_i \leftarrow H_{i-1} \cup \{e_i\}$).

A. Show that if we set $\alpha = n - 1$, then you will get Kruskal's algorithm. Also, observe that by construction, the graph H at the end of the process is an $(n-1)$-*distance emulator*. (In fact, an $(n-1)$-stretch spanning tree.)

Proof. If $\alpha = n-1$, then for any iteration i and edge e_i, the condition $d_{H_{i-1}}(x, y) \leq (n-1)d_G(x, y)$ occurs if:

- $d_G(x, y) = \infty$, i.e. x and y are disconnected in G but the edge (x, y) cannot exist then **(Contradiction)**
- $d_G(x, y) \neq \infty$, i.e. $\exists$ a path Y connecting x and y of edges previously selected with $\sum_{\lambda=(u,v)\in Y} d_G(u, v)$ $\leq |Y| d_G(x, y) \leq (n-1)d_G(x, y)$ (Edges are visited in increasing order of weight and path length is $\leq n-1$)
  Since x and y are already connected in H_{i-1}, adding e_i will necessarily induce a cycle in H_{i-1}

Therefore the only edges that will be discarded are the ones that induce cycles in the previously maintained subgraphs, i.e. induces cycles. Hence, the given algorithm produces the same output as Kruskal's algorithm.
Note: Let H be a MST of G obtained via Kruskal's. For all $(x, y) \in E$, $d_H(x, y) = \sum_{\lambda=(u,v)\in Y} d_G(u, v)$ for some path Y in H,

- If $(x, y) \in H$, $d_H(x, y) = d_G(x, y)$
- If $(x, y) \notin H$, $d_H(x, y) = \sum_{\lambda=(u,v)\in Y} d_G(u, v) \leq (n-1)d_G(x, y)$ (If $(x, y) \notin H$, it was inducing a cycle among two points already connected with (at most $n-1$) edges of smaller weight, i.e. (x, y) was the heaviest edge in a cycle since Kruskal's uses Cycle Rule to discard edges)
  Thus for all $(x, y) \in E$, $d_H(x, y) \leq (n-1)d_G(x, y)$ which makes MST H a $(n-1)$-distance emulator.

□

B. If we set $\alpha = O(\log n)$, use (c) with $g = \Theta(\log n)$ to show the final graph H is a $O(\log n)$-*distance emulator* with $O(n)$ edges.

Proof. First we argue that the girth $g = \Omega(\log n)$. Consider the first edge in the sorted order uv which creates a cycle C. Assume for contradiction that length of the cycle is less than $c \log n$, where $\alpha = c \log n$. According to the condition that $\alpha = c(\log n)$, we know that $w_{uv} < (d_{H \setminus (uv)}(u, v))/(c \log n) \leq (c(\log n) w_{uv})/(c \log n) = w_{uv}$. The last inequality follows from two facts - (1) shortest path in $H \setminus (uv)$ is upper bounded by the length of the uv path in $C \setminus (uv)$ and (2) w_{uv} is at least as large as all the previously added edges by virtue of the sorted order in which they are considered.
According to the algorithm, for any edge (x, y) that is not added to H, the following holds,

$$d_H(x, y) \leq \alpha d_G(x, y) = O(\log n) d_G(x, y)$$

In case an edge (x, y) was added to H the shortest path x between and y in H is the edge (x, y) itself, that is,

$$d_H(x, y) = d_G(x, y) < O(\log n) d_G(x, y)$$

Now we argue about $m = O(n)$ for H. We have that $g = \Theta(\log n) = c \log n$ for some constant c. From part Q3(b)(i)(C), we know that for a graph with girth g, the number of edges m satisfies:

$$m \leq O\left(n + n^{1 + \frac{1}{\lfloor g/2 \rfloor}}\right)$$

Putting $g = c \log n$, we get :

$$\begin{aligned} m &\leq O\left(n + n^{1 + \frac{1}{c \log n}}\right) \\ &= O\left(n + n \cdot n^{\frac{1}{c \log n}}\right) \\ &= O\left(n + n \cdot 2^{\frac{1}{c}}\right) \\ &= O(n) \end{aligned}$$

Hence, H is an $O(\log n)$-distance emulator with $O(n)$ edges.

□

1 Acknowledgement

The following solutions are copied form the submitted solution of:

1. Divyajeet Singh : Q1, Q3(c)(i)(A)
2. Farhan Ali : Q2(b), Q2(c), Q2(d), Q3(c)(i)(C), Q3(c)(ii)(A)