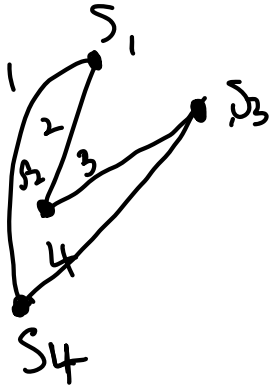
1 Study Group

Jialiang Tang (myself), SID: 3039758308

Yuanzhe Wang, SID: 3039749520

2. (a) Assume no two edges have same weight.



We prove it by induction. Firstly S_i are all vertice itself. They are all subsets of MST.

After one iteration of while loop, edge e_1 between S_i and $V \setminus S_i$ which is lightest has been added, getting $T \cup \{e_1\}$ which is also a subset of MST.

Then consider $S_2, S_3, \cdots, S_k$, if lightest edge between S_i and $V \setminus S_i$ has been added before, do nothing so we still have $\{S_i\}$ subsets of MST. If new edge e_{ij} is to be added between S_i and $V \setminus S_i$, it's a lightest edge so we get a new subset of MST having $S_i \cup S_j \cup \{e_{ij}\}$ So we keep subsets of MST in each iteration of while loop. all connected components

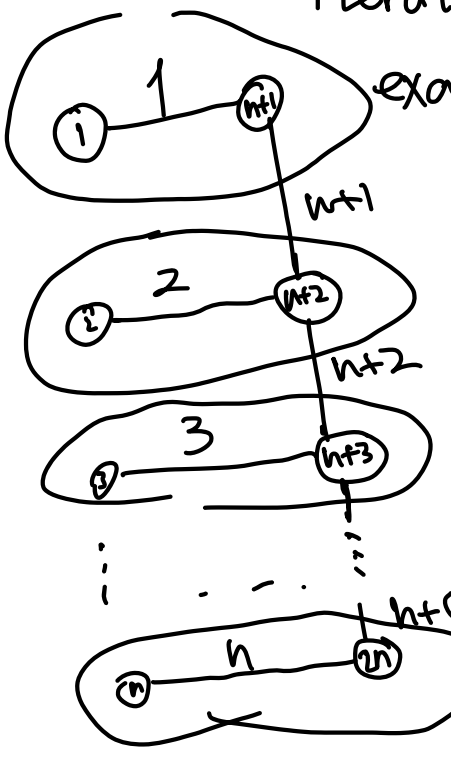
On the other hand, each time we operate while loop, we finally connects at least 2 connected components. So we'll end up with a subset of MST with exactly one connected component, which is MST of G.

(b) $\log n$ (n is $|V|$)

Proof: Since each time we add an edge, we get subsets of MST and closer to real MST of $(n-1)$ edges, we should reduce the edges added in each iteration as much as possible.

In the worst case, we only have half of $\{S_i\}$ able

to add new edges. (lightest edges of half of $\{S_i\}$ have been added before since they are also lightest edges of the other half). It's the only situation that a lightest edge won't be added. So the upper bound iterations is $\log n$. (each iteration reduces num of connected components by 2) Next we construct an example to prove it's tight.



→ Here's the tight example,

1-st iter, we get $S_1 = \{1, n+1\}$, $S_2 = \{2, n+2\}$, ..., $S_n = \{n, n+n\}$

2-nd iter, we get $S_1 = \{1, n+1, 2, n+2\}$

$S_2 = \{3, n+3, 4, n+4\}$ ...

3-rd iter, we get $S_1 = \{1, n+1, 2, n+2, 3, n+3, 4, n+4\}$,

...

$\log n$-th iter, we get MST.

(c) Runtime : $O(m \log n)$

While loop takes $O(\log n)$.

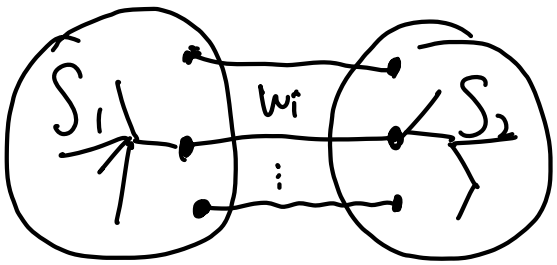
In the loop, finding connected components takes $O(n+m)$, finding $\{e_i\}$ takes $O(m)$, renewing T takes $O(m)$

It takes $O((n+m)\log n) = O(m \log n)$

3. Description of Algorithm:

Modify Kruskal so that it choses the heaviest edge each time. Run new alg on G to find the Maximum Spanning Tree of G. Cut the lightest edge w_i in Max Spanning Tree and all edges not in Maximum Spanning Tree but lighter than or equal to w_i.

Proof of correctness: Seperate V into 2 sets that w_i is a bridge edge (there can be other bridge edges) and w_i cuts Max Spaning Tree.



All bridge edges are not in Max Spanning Tree because w_i cuts the Max Spanning Tree.

On the other hand, after running algorithm, there are also no bridge edges other than w_i! That's because all bridge edges lighter than w_i have been cut, and assume that there is a bridge edge w_j heavier than w_i, we can replace w_i by w_j to produce a heavier spanning tree. Contradiction! So after algorithm, we find a cut of G. The cost of ~~that~~ cut is w_i and couldn't be better since we must cut at least one edge in Max Spanning Tree and w_i is the lightest already.

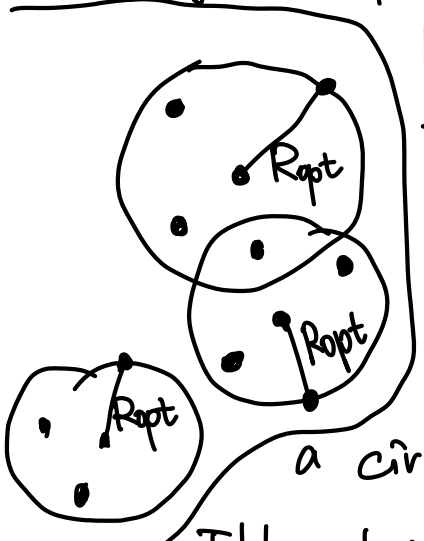
Runtime Analysis: Modified Kruskal takes $O(|E| \log |V|)$ too, and finding and cutting edges not in MST and lighter than w_i takes $O(|E|)$

It takes $O(|E| \log |V| + |E|)$ in total.

4. Description of Algorithm: Firstly choose a city randomly, then repeatedly we choose the next city farthest from any existing center. Stop when it runs k iterations.

Proof of correctness: Assume optimal set of cities with fire station is F^*. The set of cities with fire station our algorithm returns is F. Define $dist(c, F)$ as the response time of city c according to fire stations F.

An observation is that every city is in the circle with some firestation f and radium Ropt. (Since Ropt is the longest response time among all cities, every city must has a firestation within Ropt distance). That's to say, these circles cover all the cities!

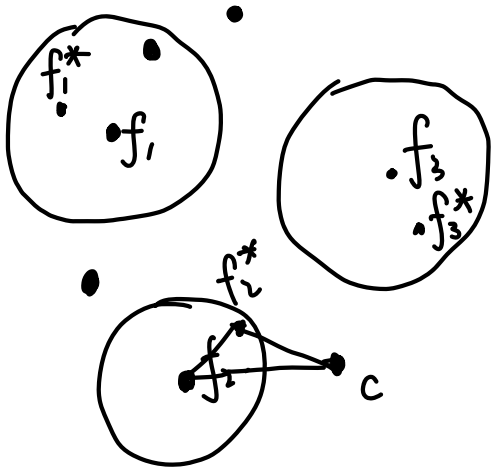


Now consider F. For each $f \in F$, construct a circle with centre f and radium Ropt.

It's obvious that there must be some $f^* \in F^*$ in each circle (Ropt is the longest response time). We try to bound Rg (precisely, $dist(c, F)$)

For every city c satisfying $dist(c, f_i^*) \le Ropt$, there must be a station f_i whose circle includes f_i^*.

So $dist(c, F) \le dist(c, f_i) \le dist(c, f_i^*) + dist(f_i^*, f_i)$



$\leq R_{opt} + dist(f_i^*, f_i)$

$\leq R_{opt} + R_{opt} = 2R_{opt}$.

So $R_g \leq 2R_{opt}$

It holds for every city, so it's a 2- approximation algorithm.

Runtime Analysis: Seleeting a fire station city must compare the $dist(c, F_{current})$ for every city c, taking

$$O(1+(N-1)+2(N-2)+\ldots+K(N-K)) = O(NK^2) = O(N^2K)$$

$$\left(\sum_{i=1}^{K} i(N-i) = \sum_{i=1}^{K} iN - i^2 \leq \sum_{i=1}^{K} iN = N \cdot \frac{K(K+1)}{2} = O(NK^2) \right.$$

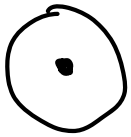
$$\left. \text{and } K = O(N) \right)$$

Supplementary Proof: It's a supplement of proof of correctness. To prove the approximation factor, we should guarantee that for some f_i^* satisfying $dist(c, f_i^*) \leq R_{opt}$ of c, there must be a f_i in the circle of f_i^* with radium R_{opt}. (It holds for every c)

By our choose strategy of F, the first f_1 must be in some circle. Assume we have chosen $f_1, f_2, \ldots, f_m$, each in different circles of $f_1^*, \ldots, f_m^*$. Next we choose f_{m+1}, which should be out of every circles of $f_1^*, \ldots, f_m^*$ and in a circle of some f_{m+1}^* (if there are still cities not covered

by circles of $f_1^*, \cdots, f_m^*$. Otherwise $f_{m+1}^*, \cdots, f_k^*$ are all in circles,

then the m circles all have a f_i and these circles can cover all cities. So for every c, we can find a f_i^* that $dist(c, f_i^*) \leq R_{opt}$ and there's some f_i in circle of f_i^*).



If "otherwise" never happens, we also get k circles coving all cities and the fact still holds.