CS180 Final

Xilai Zhang

TOTAL POINTS

64 / 100

QUESTION 1

1 Problem 1 25 / 25

- **0 pts** Correct. Used greedy algorithms and scan/remove from leafs.
- **0 pts** Correct DP solution

✓ **- 0 pts Correct. Modeled it as bipartite matching or network flow**

- **5 pts** Used Dynamic Programming and gave a recursion. However, the recursion cannot reflect how you make sure that the edges do not share nodes.
- **5 pts** Did not indicate how do you construct two disjoint node sets and converted this problem to a bipartite graph matching problem. A simple approach is taking any node as the root and coloring nodes level by level.
- **10 pts** Pointed out that we first solve the subproblem of the child node and merged the result. However, no correct DP recursion is given or you do not consider the two cases separately that the current root can be taken/not taken.
- **5 pts** The way that you divide nodes into two sets is not right. A simple approach is taking any node as the root and coloring nodes level by level.
- **15 pts** Misunderstood of the problem / gave not quite related solution. The correct solution is modeling the tree as a bipartite graph or using dynamic programming.
- **25 pts** No answer
- **20 pts** No clear algorithm
- **10 pts** Used greedy algorithm but not started from leafs. The algorithms cannot always yield correct solution if you just start from a random node.

QUESTION 2

2 Problem 2 25 / 25

✓ **- 0 pts Correct**

- **25 pts** It's not NP-complete
- **10 pts** Only answered the feasibility question but did not give an algorithm that returns the assignment
- **8 pts** Didn't consider the availability of classrooms
- **15 pts** Modest attempt. Your algorithm may output no feasible assignment when there is indeed one
- **2 pts** Did not describe how to assign edge capacity
- **3 pts** Did not answer when there is or is not a feasible solution. (When max flow equals to what?)
- **3 pts** Wrong edge capacity design
- **12 pts** Correct direction but wrong algorithm. You need "perfect matching" but not a "matching"
- **20 pts** No solution is given for (b) / No related is solution given for (b)
- **+ 5 pts** Mentioned that we can use max flow/matching though no algorithm is given
- **5 pts** It's not NP-complete. The algorithm is fine, though

QUESTION 3

3 Problem 3 3 / 25

- **0 pts** Correct
- **0 pts** (a) full credit
- **10 pts** (a) wrong reduction

✓ **- 7 pts (a) Reducing from Hamiltonian cycle or tsp. but no discussion about the length of the cycle. no or wrong discussion about how to set the edge weights and no or wrong discussion about why the constructed edges weights satisfy the triangle inequality.**

- **5 pts** (a) the edge weights are set improperly. The triangle inequality is broken.
- **0 pts** (b) full credit
- **3 pts** (b) reduction from euclidean TSP, but fail to

explain the edge weight setting correctly.

✓ **- 5 pts (b) wrong reduction or fail to explain how to set the edge weights**

- 3 pts (b) the reduction details are missing or wrong

- 0 pts (c) full credit

- 5 pts (c) Apply the MST idea with wrong MST details.

- 8 pts (c) The polynomial solution is missing or wrong. No explanation why it works.

✓ **- 10 pts (c) empty answer or wrong answer**

- 5 pts (c) confirm it as a NPC and give a MST-like solution

- 25 pts Empty answer for three sub-questions

QUESTION 4

4 Problem 4 11 / 25

- 0 pts a. Correct

✓ **- 6 pts a. wrong recursion (formula)**

- 8 pts a. no/irrelevant recursion (formula)

- 10 pts a. No answer / irrelevant answer

- 0 pts b1. Correct

✓ **- 3 pts b1. Incorrect argument to avoid O(v^2 log v) total cost**

- 5 pts b1. no answer to how to avoid O(v^2 log v) total cost

- 0 pts b2. Correct

✓ **- 5 pts b2. no(wrong) explaination for log N rounds in total**

- 6 pts b2. algorithm lacks details

- 7 pts b2. The algorithm lacks many details and no explanation for log N rounds in total

- 10 pts b2. no answer

CS 180: Introduction to Algorithms and Complexity
Midterm Exam

Mar 19, 2019

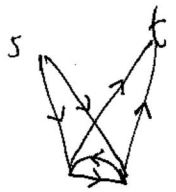
Name	Xilai Zhang
UID	804796478
Section	Friday 12-1:50

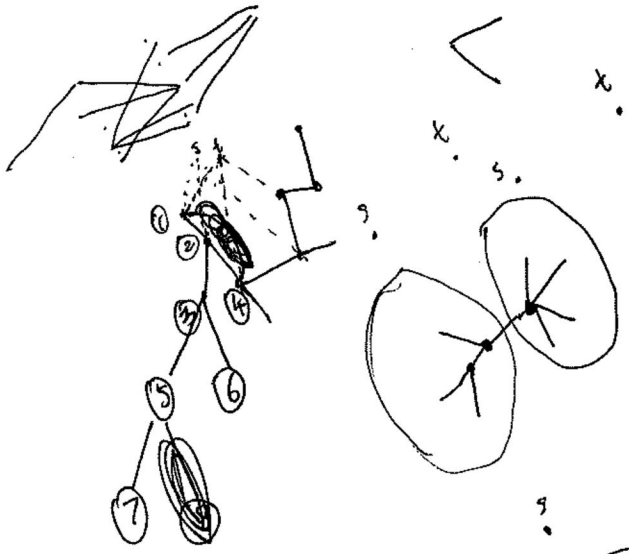
1	2	3	4	Total

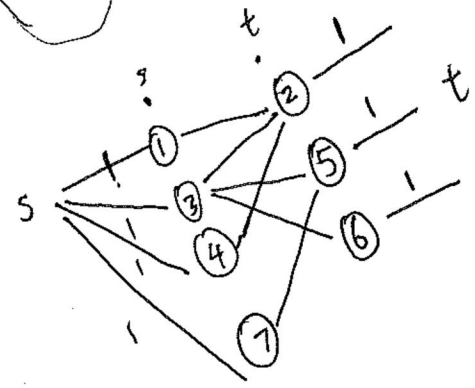
★ **Print your name, UID and section number in the boxes above, and print your name at the top of every page.**

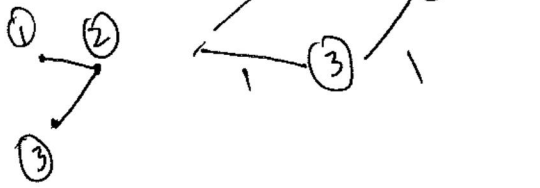
★ **Exams will be scanned and graded in Gradescope. Use Dark pen or pencil. Handwriting should be clear and legible.**

- The exam is a closed book exam. You can bring one page cheat sheet.
- There are 4 problems. Each problem is worth 25 points.
- Do not write code using C or some programming language. Use English or clear and simple pseudo-code. Explain the idea of your algorithm and why it works.
- Your answer are supposed to be in a simple and understandable manner. Sloppy answers are expected to receiver fewer points.
- Don't spend too much time on any single problem. If you get stuck, move on to something else and come back later.


s
t


s
t
1
2
3
4
5
6
7


s
t
1
2
3
4
5
6
7

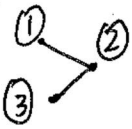

1
2
3

Name: Xilai Zhang UID: 804796478 CS180 Final Exam

1. We have seen in class a polynomial time algorithm for maximum (cardinality) matching in bipartite undirected graphs. In general undirected graph, the problem is not *NP*-complete but the algorithm is quite involved. Suppose we take a tree (undirected tree) and ask for maximum matching. Can you give a polynomial time algorithm? If you can, outline the algorithm. [25 pts]

To convert undirected to directed graph, we first replace each edge with two parallel edges.

Then we connect each node in G to node s and node t, where s is the source and t is the sink.

We can create two biparite groups A and B by following along the tree. For each edge (u, v) in the tree, if one of its end points is in one group of A and B, we assign the other endpoint to the other group of A and B. For example, if the tree is 

, where ①, ②, ③ are nodes. We assign node ① to group A, and since ①② is an edge, we assign ② to the other group B. since ②③ is also an edge, and since ② is already assigned to group B, we assign node ③ to group A. This will give us a biparite graph, and if we give all (flow) edges that connect node t with node in group B the capacity 1, we will be able to perform max flow and find maximum matching.

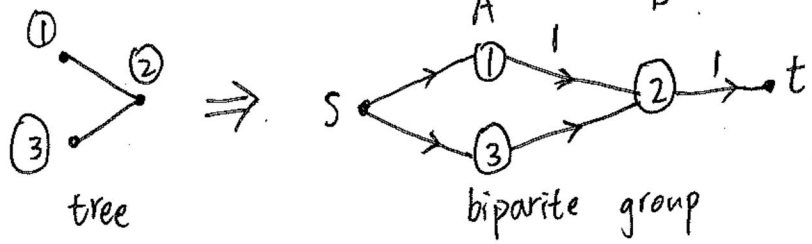


Figure 1. illustration

see back of the page for further illustration

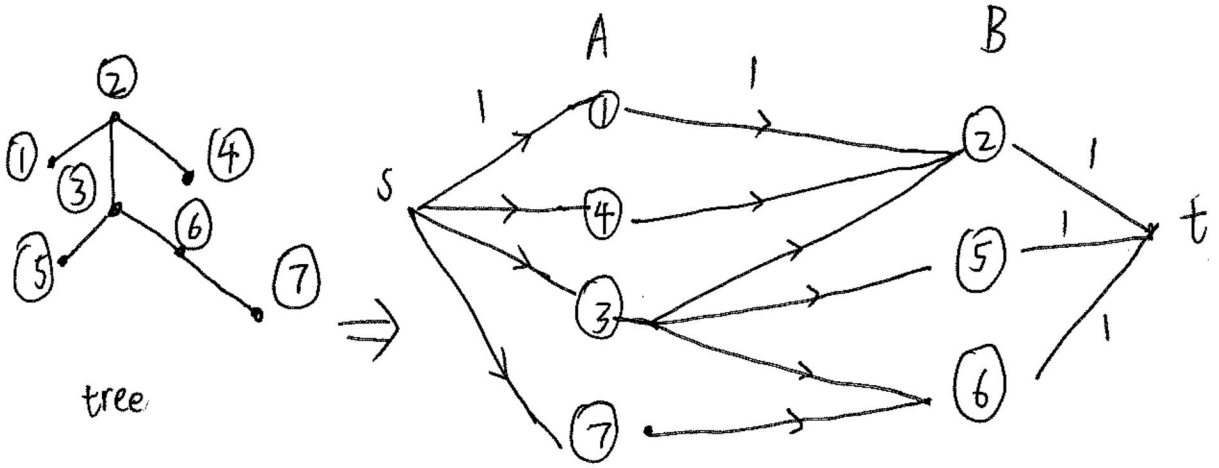
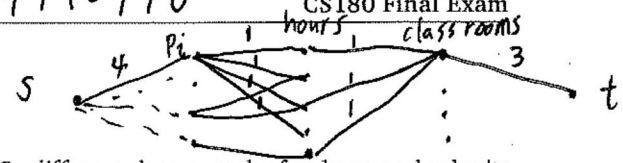


Figure 2. illustration of transformation of a more complex tree

for a more complex tree, we follow the same rule to generate biparite graph. For each edge, assign the endpoints to different groups. We can guarantee that there will be no flow between node in the same group, otherwise there will be a cycle. Since we are given a tree, which has no cycle, we can be sure that we can transform this tree into a biparite graph and perform max flow.



2. You are given a list of professors. Each professor P_i teaches C_{P_i} different classes, each of an hour, and submits C_{P_i} different hour intervals in which she wants to teach. She is indifferent to what class she teaches in each hour interval which she submitted. On the other hand we have a list of classrooms. H_{R_k} is the list of time intervals when each classroom R_k is available. We want to answer whether it is feasible for all professors' requests to be satisfied, and if it is, output the assignment. This problem is obviously in *NP* (why?).

   (a) Is the problem *NP*-complete? [5 pts]

   (b) If yes, prove it is *NP*-complete. If not, give a polynomial time algorithm to answer the feasibility question and output a feasible assignment if there is one. [20 pts]

We can solve this problem by performing max flow. connect source s to each professor node P_i, and give the edge capacity C_{Pi}. Then we create all the time interval nodes, H_{R_K}, connect professor node and time interval node if the professor wishes to teach in the specific time interval. Then we create all the classroom nodes R_k, connect classroom node to time interval node if the classroom is available at the time interval. Finally we connect all classroom node with sink node t, and give each edge capacity H_{R_K}, the number of total intervals. We can then perform max flow, if max flow is $\sum C_{Pi}$, ~~then~~ the sum of all classes that all professors wish to teach, then we say there is an assignment.

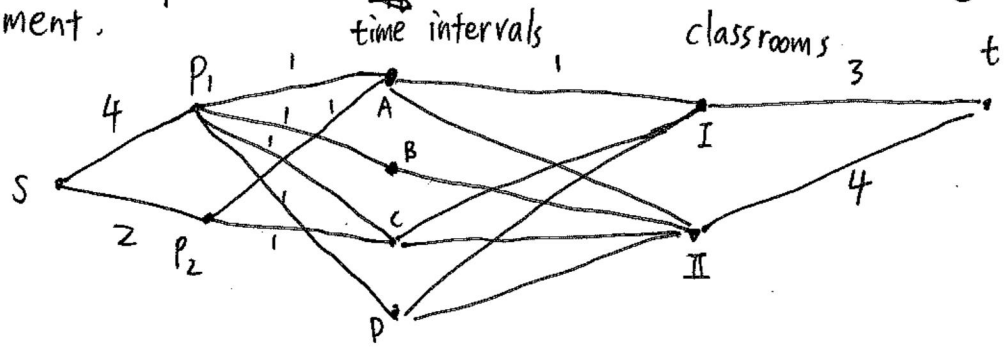


Figure 1. An illustration of a sample max flow chart if P_1 wish to each at intervals ABCD, P_2 wants to teach at intervals AC. classroom I is available at interval ACD, and classroom II is available at interval ABCD. In this flow, if max flow is 6 we are guaranteed to have an assignment.

see back of this page for further illustration

this problem is in NP because given an assignment, we can simply check if all requests of the professors are satisfied.

(a) However, this problem is not NP-complete because we can solve it in polynomial time

(b) we can solve it via max flow, the details are given in the previous page.

3. We have seen in class, by reduction from Hamiltonian cycle, that undirected TSP is *NP*-complete.

   (a) The Euclidean TSP (call in class mistakenly "planar") is a TSP problem where edge weights in the graph satisfy the triangle inequality ($\forall\ v_1, v_2, v_3,\ w\{v_1, v_2\} \leq w\{v_1, v_3\} + w\{v_3, v_2\}$). Prove that the Euclidean TSP is *NP*-complete. [10 pts]

   (b) We relax the condition on the (non-Euclidean) TSP that each city is to be visited only once. If the saleswoman goes to Chicago through Huston, she can fly back to Huston on the way to Miami (nevertheless, at the end she back to her city). Show that the relaxed-TSP is *NP*-complete (hint: do not ignore context). [5 pts] is

   (c) We are now in the relaxed-TSP, and not only the weights do not satisfy the triangle inequality but also they are terribly skewed with respect to each other. Namely, the weights when ordered from low to high $w_1, w_2, \ldots$ satisfy that $w_i > \sum_{j=1}^{j=i-1} w_j$ for all i. Is the relaxed-skewed-TSP problem *NP*-complete? If not, give a polynomial time algorithm and argue the correctness of your algorithm. [10 pts]

(a) Since Hamiltonian cycle requires each vertex to be visited only once, there cannot be any triangle or cycle on the hamiltonian cycle. Thus, we can first reduce hamiltonian cycle to a ~~no~~ TSP problem with equal weights, and after we find the hamiltonian cycle, we keep the weights on the hamiltonian-cycle-corresponding (figure illustration below) edges in TSP as same, and modify the weights of other edges to make the TSP Euclidean.

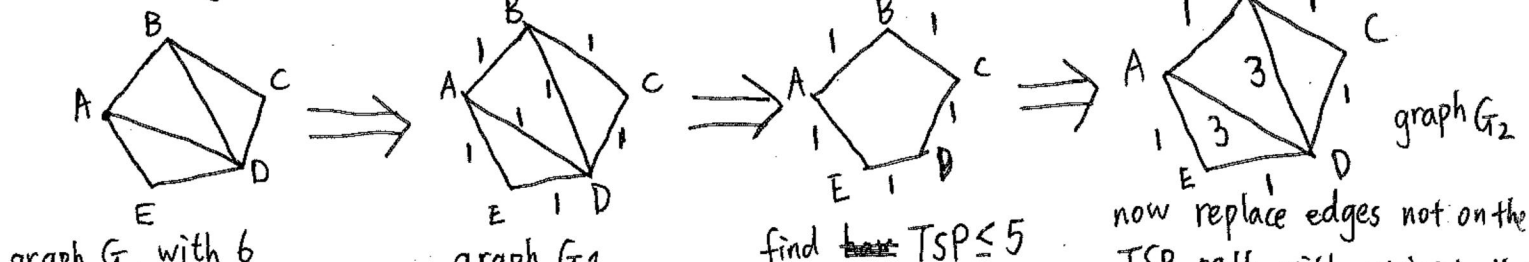
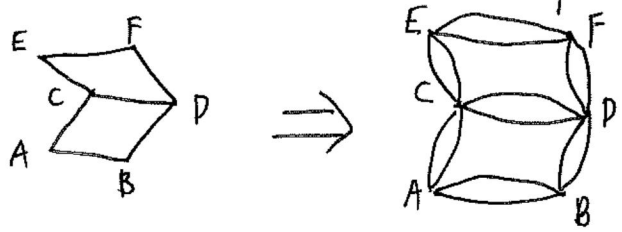


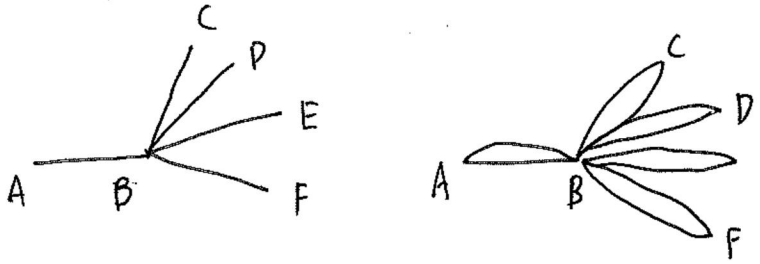
Figure 1. For example, the hamiltonian cycle in G can be reduced to TSP in G1. After we get the TSP path in G1, we keep the weights on the path, modify weights of edges not on TSP path and generate G2. Now hamiltonian cycle in G is reduced to TSP ≤ 5 in ~~non~~ Euclidean graph G2.

see back of this page

(b) we can simply replace each edge with parallel edges to allow another visit ("in the context", potentially revisit)



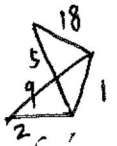
we can allow at most n-1 (the maximum number of other vertices we will visit) parallel edges. (If that is ~~wh~~ the number of times we wish to revisit)



n-1 pairs of parallel edges around B

(c)

G



G'

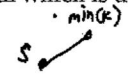
We know that if we take out the largest edge in G and generate a graph G'. We know for sure that TSP of G' is smaller or equal to TSP of G, Because if the largest edge is included, the TSP of G will be greater than TSP of G' since $W_i > \sum_{j=1}^{j=i-1} W_j$. Thus, the TSP of G will be the first TSP if we include edges into our graph in increasing order.

Thus, the TSP of G will be the first graph that has a hamiltonian cycle if we include edges into our graph in increasing order.

Since, there is no polynomial time to find hamiltonian cycle, there is no polynomial time algorithm to solve relaxed-skewed-TSP either.

H—T

4. (a) In class we have seen the Bellman-Ford algorithm for one-to-all shortest paths with negative edges but no negative cycle. Write a recursion for shortest paths problem such that you can argue the recursion is amenable to dynamic-programming. And argue that the Bellman-Ford algorithm is in fact an iterative implementation of your recursion (Do not confuse Bellman-Ford with Floyd-Warshal which is all-to-all shortest paths algorithm). [10 pts]



(b) We said in class that the "idea" of an algorithm is manifested in its recursion.
Here's a recursion to solve MST: For each node v find the min weight edge adjacent to it. These chosen edges create a forest (why no cycle?). We take these edges to be in the MST. Now we "contract" all nodes incident to the same tree into a single "new node", which is connected to other "nodes" by original edges that connect a node in one tree to a node in *another* tree. All the intra-tree edges (edges aside from the tree edges that connect nodes in the same tree) are "gone." Notice that this might create "parallel edges" but that is ok.
We want to implement this recursion into an $O(|E|\log|V|)$ time algorithm. The implementation that Prof. Gafni knows requires that for each node the edges around it are ordered by weights. Alas, this looks like resulting in a cost of $O(|V|^2\log|V|)$ algorithm which is larger than $O(|E|\log|V|)$ for a sparse graph.

- Help get Prof. Gafni out of this conundrum. [5 pts]
- Outline an algorithm and argue it achieves the desired complexity (To find whether an edge is inter or intra tree its better be that all nodes in the same tree in the recursion are named the same. You want to argue that throughout the algorithm a node changes name at most $\log|V|$ time. Recall Union-find.) [10 pts]

see back of this page for Bruvolsk's (I don't remember the exact name, starts with B) algorithm and star contraction

(a) recursion: $\min(v) = \min\left[\min(u) + w(u,v)\right]$ for all vertex u that connects to v.

Here, in this recursion $w(u,v)$ is weight of edge (u,v). $\min(v)$ is minimum distance from source s to v. $\min(u)$ is minimum distance from source s to u.

This recursion is improved by Bellman-Ford, such that Bellman-Ford remembers the minimum distances to vertices, so that $\min(v)$ can be calculated from $\min(u)$, given that u has already been calculated in phase before $\min(v)$. It is iterative because in each iteration, Bellman-Ford algorithm guarantees that the minimum distance of all the distances is the minimum distance to that vertex, (call this vertex x) and will thus start from x, examine all outgoing edges from x, and find a new minimum distance. In this way, one minimum distance to a vertex is calculated in each iteration. (vertex x is not considered when finding new minimum distance). Thus by going through n iterations, the algorithm will generate n minimum distances.

(b) We solve both parts by performing the MST algorithm in distributed (multiprocessor) ~~system~~ network and star contraction (potentially any graph contraction method would work)

In each round, we flip coins for each node and assign the node a value of either "head" or "tail". Now, for all the "tail" nodes, if its minimum edge connects to a "head" node (or a "cluster"), we merge the "tail" node to the "head" node. ~~Now~~ If we use a union-find structure (as prof gafni said in spec), we can make a pointer from the tail node to the head node, and then we would no longer consider the tail node. this will make "head" node "cluster" bigger and bigger in each iteration, and eventually we will be left with only one node. This way of contraction is called star contraction. (Figure illustration below) H stands for head, T stands for tail

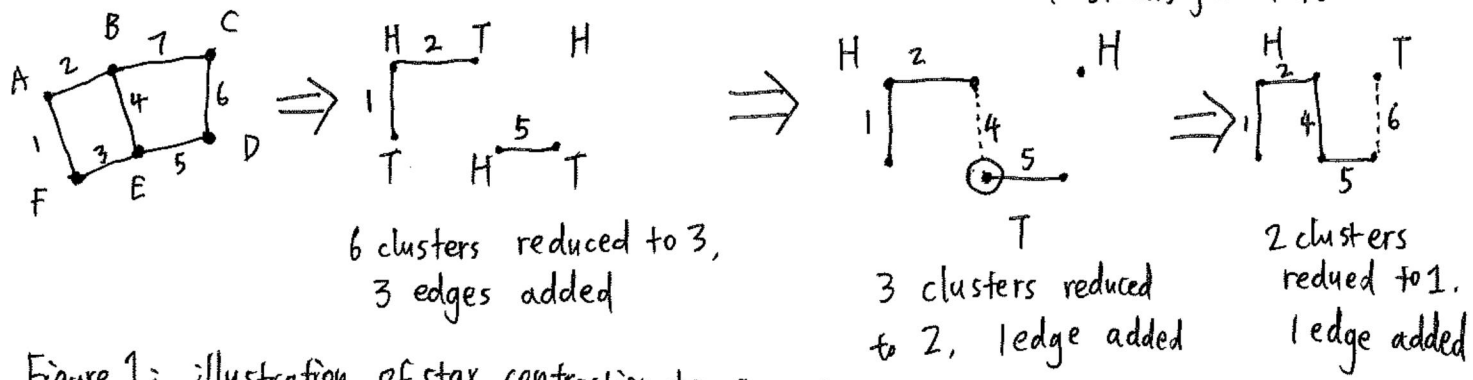


Figure 1: illustration of star contraction to generate MST.

Now we can analyze complexity. For each ~~node~~ cluster, the probability that it is assigned "tail" and the cluster its min edge connects to get assigned "head" is $\frac{1}{4}$. This guarantees us that the number of rounds that we need to assign "head" or "tail" is $\log V$. In each round, we only consider the min edge from each "tail" node, once this edge is included, we will no longer consider outgoing edges from previous "tail" nodes. Instead, we will only consider outgoing edges from the "head" nodes. Thus, ~~the~~ edges we consider is porportional to E. Thus the total time complexity is $O(E \log V)$.