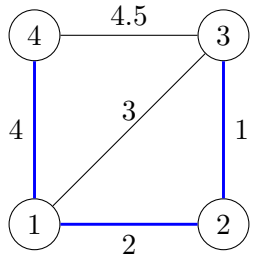
CSE421: Design and Analysis of Algorithms	April 18, 2024
Shayan Oveis Gharan	Section 4

P1) You are given a connected undirected weighted graph $G = (V, E)$ with n vertices and m edges, their weights, w_f for all edges $f \in E$, a minimum spanning tree $T \subseteq E$ together with an edge $e \in E$ and a new weight w'_e for e, design an algorithm that runs in time $O(n + m)$ to test if T still remains a minimum spanning tree of the graph if we change the weight of e to w'_e. Your algorithm should output "yes" if T is still an MST and "no" otherwise. Note that the edge e may or may not belong to T and we may decrease or increase weight of e. For simplicity, assume that all edge weights are distinct both before and after the update. For example in the following picture the MST is colored in blue. If we update the weight of the edge $(3, 1)$ to 1.5 the blue tree is no longer the MST.



Solution: Let $e = \{u, v\}$ and let w'_e be the new weight of e. There are two cases.

Case 1: e is in T, then let $T' = T - \{e\}$.

We label the connected components of u with u and the connected component of v with v using BFS or DFS. We iterate over all edges of G: For any edge $\{u', v'\}$ if u', v' are in two different connected component and $w_{u',v'} < w'_e$ we output "no", i.e., T is no longer the MST. Otherwise, output "yes", i.e., T is still the MST.

Case 2: e is not in T. In this case we add e to T. This will create a cycle C. By the cycle property, the largest edge of C is not in the MST. So, if e is the largest edge, T will remain the MST, otherwise it won't.

Runtime: Suppose $e \in T$. Labeling the two connected components of T' takes $O(n)$ since T' is the union of two trees. Iterating over all edges in G is $O(m)$, giving a total complexity of $O(m+n)$ for first case. In the second case, it takes time $O(n)$ to find the path P in T between u, v. It also takes time at most the length of the cycle to find the largest edge of P. Overall, this case will run in time $O(n)$.

Correctness: In the first case, we will create a cut $(S, V - S)$ by removing edge e from T. By the cut property of the MST, $\{u, v\}$ is contained in the MST if it is the smallest weight edge in the cut. If some other edge e' has a smaller weight, then by the cut property T is no longer the MST.

```
Function InMSTAfterRemoval(G, w_1, w_2, T, e = (u, v)))
    if e ∈ T then
        T' ← T − e
        BFS on T' starting at u and label all vertices found with u
        BFS on T' starting at v and label all vertices found with v
        for each edge e' = (u', v') in G do
            If label[u'] ≠ label[v'] AND w_{u',v'} < w'_e then return "no"
        end
        return "yes"
    end
    else
        Use DFS to find the unique path P from u to v in T
        Let e' be the largest edge of path P. If w_{e'} > w'_e output "no", else output "yes"
    end
```

Otherwise if e is the smallest edge of $(S, V-S)$ we claim T remains MST. To show that we show all edges of T satisfy the cut property. Consider an arbitrary edge $f \neq e$ of T. Note that f remains the smallest edge of the unique cut of $T-f$, call it $(S', V-S')$, as $e \notin (S', V-S')$ and the weight of all edges of $(S', V-S')$ are not changes. So, all edges of T satisfy the cut property and T remains MST.

In the second case, adding edge e to cycle C will create a unique cycle (as discussed in section 3). By the cycle property, if e is not the largest edge of C, T is no longer the MST and we should output "no".

Now, assume that e is the largest edge of C. We claim T remains MST. To prove that we show that all edges of T satisfy the cut property. Consider an arbitrary edge $f \in T$ and let $(S, V-S)$ be the unique cut of $T-f$. For contradiction, suppose f is not the smallest edge of this cut. Since before changing weight of e, T was MST, f must have been the smallest edge of this cut before and $e \in (S, V-S)$ and $w'_e < w_f$. But since f is the unique edge of T in $(S, V-S)$, and the endpoints of e are on the two sides of the cut, this means that f is on the unique path between endpoints of e in T, i.e., $f \in C$. But, this contradicts with the assumption that e is the largest edge of C. So, in this case T is MST and we should output "yes".

P2) Amy, a TA in 421 is asked to grade n sheets of exams, $1, \ldots, n$. Let t_i be the time that takes her to grade the i-th sheet (perhaps, t_i depends on how long the i-th proof is). Whenever she grades the i-th sheet the grade will be published in Gradescope right away. Say she finishes grading i-th sheet at time f_i. Then, the average grading time of all exams is $\frac{1}{n}(f_1 + \cdots + f_n)$. To make the students happy, Shayan asked Amy to minimize the average grading time, that is the average time for a 421 student to see his/her grade. We want to help Amy by figuring out the optimal order to grade these sheets. Design an efficient algorithm which outputs the minimum average grading time (note that your algorithm just needs to output a number).

For example, if $t_1 = 3, t_2 = 2, t_3 = 4$ then the optimal order to grade is, $2, 1, 3$. Then, we have $f_2 = 2, f_1 = 2 + 3 = 5, f_3 = 2 + 3 + 4 = 9$ and the average grading time is $16/3$.

Algorithm: Sort the sheets in an increasing order of the grading time, say $t_1 < t_2 < \cdots < t_n$ and output $\frac{1}{n}\sum_{i=1}^{n}(n-i)t_i$.

Runtime: The algorithm obviously runs in time $O(n \log n)$.

Correctness, Greedy stays ahead: We use the Greedy is ahead strategy. Say the processing time of job i is t_i. Let $j_1, \ldots, j_n$ be the optimal order. We show that for every k, our k-th job finishes no later than the k-th job in the optimal order. In our order, the k-th job finishes at time $t_1 + \cdots + t_k$ whereas the k-th job in the optimal order finish at time $t_{j_1} + \cdots + t_{j_k}$. But since we sorted the jobs, $t_1, \ldots, t_k$ has the *smallest* processing time among any set of k jobs. Therefore,

$$t_1 + \cdots + t_k \leq t_{j_1} + \cdots + t_{j_k}.$$

Since the above inequality holds for all k, the total sum of finishing time in our schedule is no more than the optimum.

Correctness, Exchange argument: Suppose $j_1, \ldots, j_n$ is the order of jobs in the optimum solution and for the sake of contradiction, suppose these are not sorted based on their length. It follows that there must exists some $1 \leq i < n$ such that the job j_i takes longer than j_{i+1}, $t_{j_i} > t_{j_{i+1}}$. Now, consider another solution in which the order of jobs j_i and j_{i+1} is swapped, i.e.,

$$j_1, j_2, \ldots, j_{i-1}, j_{i+1}, j_i, j_{i+2}, \ldots j_n.$$

We claim that this new order has a smaller total sum of finishing times. To see that first notice the first $i-1$ jobs finish at the same time as in the optimum. The i-th job ends earlier in the new order since

$$t_{j_1} + \cdots + t_{j_{i-1}} + t_{j_{i+1}} < t_{j_1} + \cdots + t_{j_{i-1}} + t_{j_i}.$$

Furthermore, the $i+1$-st job all the way to the n-th job also finish at the same time as the optimum: This is because the $i+1$-st job in both schedule ends at $t_{j_1} + \cdots + t_{j_{i+1}}$. So, in summary every job (other than the i-th job) ends at the same time as the optimum but the i-th job ends at an earlier time in the new order. This implies that the new order has a smaller average grading time which is a contradiction.