Algorithm Design and Analysis
Assignment 4

1. You are given a set of n jobs, where each job j is associated with a size s_j (how much time it takes to process the job) and a weight w_j (how important the job is). Suppose you have only one machine that can process one unit of jobs per time slot. Assume all jobs are given at time $t_0 = 0$ and are to be processed one by one using this machine. Let $C_j > t_0$ be the time that job j is completed. The goal is to find a schedule (of all the jobs) that minimizes the weighted completion time, i.e.,$\sum_{j=1}^{n} w_j C_j$.

   Greedy algorithms first order the jobs according to some criteria, and then process them one by one. Only one of the following criteria gives an algorithm that minimizes the weighted completion time.

   - Highest Weight First: process jobs in in descending order of their weights.
   - Smallest Size First: process jobs in ascending order of their sizes.
   - Highest Density First: process jobs in descending order of their weight to size ratio, namely in descending order of w_j/s_j.

   (a) Find out which one is correct.

   (b) Reason about its correctness.

   (c) Give counter examples for the other criteria.

   Solution.

   (a) Highest Density First is correct.

   (b) Given jobs i,j with $w_i/s_i > w_j/s_j$, we say that (i, j) is an **inversion** in schedule S, if S schedules j before i.

   Observations:

   1. The greedy algorithm does not have idle time
   2. The optimal algorithm does not have idle time.
   3. according to the greedy algorithm, the number of inversion is 0.
   4. If a schedule (with no idle time) has an inversion, it has one with a pair of inverted jobs scheduled consecutively.

   Then we show that swapping two adjacent, inverted jobs reduces the number of inversions by one and does not increase the weighted completion time.

   Proof. : suppose that (i,j) is an inversion in S with $\frac{w_i}{s_i} > \frac{w_j}{s_j}$, and S schedule j before i, and j starts at time T, after the swapping ($S^{'}$), the difference between the weight completion time between S and $S^{'}$ is

$W(S) - W(S') = w_j(T + s_j) + w_i(T + s_j + s_i) - (w_i(T + s_i) + w_j(T + s_i + s_j)) = w_i s_j - w_j s_i > 0$, where the last inequality is due to $\frac{w_i}{s_i} > \frac{w_j}{s_j}$.
So repeating this process shows the optimality of greedy. □

(c) 1. Counter example for Highest Weight First:
Suppose there are two jobs, job 1 with $s_1 = 100, w_1 = 10$, and job 2 with $s_2 = 10, w_2 = 9$. According to the greedy algorithm with criterion highest weight first, job 1 is before job 2, the weighted completion time is

$$W = 10 * 100 + 9 * (100 + 10).$$

And the optimal algorithm is to schedule job 2 before job 1, and the weight completion time is

$$W' = 9 * 10 + 10 * (10 + 100).$$

It's easy to see $W > W'$.

2. counter example for Smallest Size First:
Suppose there are two jobs: job 1, with $s_1 = 10, w_1 = 1$, and job 2, with $s_2 = 20, w_1 = 100$.
According to the greedy algorithm with criterion smallest size first, job 1 is before job 2, the weighted completion time is

$$W = 1 * 10 + 100 * (10 + 20).$$

And the optimal algorithm is to schedule job 2 before job 1, and the weight completion time is

$$W' = 100 * 20 + 1 * (10 + 20).$$

It's easy to see $W > W'$.

□

2. Let $G = (V, E)$ be a connected undirected graph with positive edge weights. Give an algorithm to find a subset of edges E' with the smallest total weight such that removing E' from G will leave a graph with no cycle. Note that E' must contain at least one edge on every cycle of G. You need to prove the correctness of your algorithm.

Solution. Let $E^* = E - E'$.

Given E, the task of finding E' with the smallest total weight is equivalent to finding E^* with largest total weight where we require $G^* = (V, E^*)$ does not contain any cycle.

Notice that $G^* = (V, E^*)$ must be connected since otherwise we can always add edge into E^* making its total weight larger while not breaking the no-cycle constraint.

Since G^* is acyclic and connected, G^* is a tree. Thus, finding G^* with E^* of largest total weight is equivalent to finding the Maximum Spanning Tree of the original graph G. To find the maximum spanning tree, we can first apply the following transformation to all edge weights:

$$w^*_{\mathrm{u}} = -w_{\mathrm{u}} + MAX_{v:v\in E}\{w_v\} \quad \forall u \in E$$

After such transformation, all edge weights are still non-negative. Hence, we can run Prim's algorithm or Kruskal's algorithm to find the Minimum Spanning Tree of the graph under the transformed edge weights $\{w^*_u\}$.

Firstly, adding a constant number to all edge weights does not change the MST of a graph. Secondly, reversing the signs of all edge weights makes the Minimum Spanning Tree found be the actual desired Maximum Spanning Tree under the graph with pre-transformation edge weights $\{w_u\}$.

After we have found the Maximum Spanning Tree E^* of G, we return $E' = E - E^*$ as final answer. Overall runtime is $O(E \log V)$ □

3. Alice wants to throw a party and is deciding whom to call. She has n people to choose from, and she has made up a list of which pairs of these people know each other. She wants to pick as many people as possible, subject to two constraints: at the party, each person should have at least five other people whom they know and five other people whom they don't know. Give an efficient algorithm that takes as input the list of n people and the list of pairs who know each other and outputs the best choice of party invitees. Give the running time in terms of n.

Solution.

(a) Algorithm:

- Construct an undirected graph $G(V, E)$ for these people(adjacent matrix), where each vertex represents a person and each $(i, j) \in E$ implies that i and j know each other.
- We recursively find and delete a vertex with degree d where $d < 5$ or $|V| - d - 1 > 5$ until there is no such vertex.
- The set of vertices left is exactly the best choice.

(b) Optimality:

If a vertex with degree d satisfies $5 \leq d \leq |V| - 6$, we call it a "good" vertex, otherwise it is "bad".

Notice that deleting a vertex is likely to transform a remaining "good" vertex to a "bad" one but will never transform a "bad" vertex to a "good" one because the deletion reduces either d or $|V| - d - 1$.

Assume there is a optimal algorithm OPT, the deletion of which in i-th round is d_i. The set of remaining vertices is R and the set of deleted vertices is D. $|R| \cap |D| = \emptyset$, $|R| \cup |D| = |V|$.

Similarly, in our algorithm, call the deletion in i-th round d'_i. The set of remaining vertices R' and the set of deleted vertices D'.

Now it is easy to prove that $D' \subseteq D$ by induction:

- In the first round, d'_i is deleted, which implies it is "bad". So it will be deleted in OPT too (though may not in the first round) because it will never change to "good".
- In the i-th round ($i > 1$), we have that $d'_1, \ldots, d'_{i-1}$ are deleted, and now d'_i is "bad" too (though it may be "good" before). Because $d'_1, \ldots, d'_{i-1}$ will be deleted in OPT too, d'_i will not be "better" in OPT than in our algorithm. d'_i will also be deleted in OPT.
- Finally, we have that $D' \subseteq D$, which implies $|R'| \geq |R|$. Because $|R|$ is optimal, we have $|R'| = |R|$, our algorithm is also optimal.

(c) Running time:

There are at most n deletions, so it runs at most n rounds. In each round, it takes up to $O(n)$ time each to search and to update if we store the degrees in an array. The total running time is $O(n^2)$

□