CIS 675 — Asif Salekin
due tuesday nov 6, 2019 at 11.59 pm est via Blackboard

H3.Peinan Weng

- The assignment is due at Blackboard on Tuesday Nov 6 at 11:59pm. There is no Late submission. You can do multiple submissions. Submit early and often. Only the last submission will be considered.
- You are permitted to study with friends and discuss the problems; however, *you must write up you own solutions, in your own words*. Do not submit anything you cannot explain. If you do collaborate with any of the other students on any problem, please do list all your collaborators in your submission for each problem. You can have at-most 3 collaborators.
- You can study online. But, finding and writing solutions of homework problems from the web, is strictly prohibited.
- We hope that all homework submissions are prepared in Latex. If you need to draw any diagrams, however, you may draw them with your hand. Please use *exactly 1 page* for each of the answers. You can choose not to write the homework in Latex, but you can not take a photo of your text book and attach it. You have to submit it in a pdf format.
- If you take photo of your textbook and attach it, or if you do not submit your assignment as PDF file, your assignment will not be graded.

problem 1 *Cross-Country trip planner part 1 (4 point)*

Suppose you and your friends need to drive across the country. You want to drive as much as you can during the day, but decide that you will rest every night. You have identified a list of suitable sleeping points along the drive and need a system for determining when to stop each evening. You decide to use a greedy approach. Design a greedy algorithm that minimizes the number of stops necessary for the trip

To formalize this problem a bit, consider the following definitions / assumptions:

- You may imagine the drive as a long straight line segment of length L and assume you can always drive at most d miles per day.
- You can assume the stopping points are located at $(x_1, x_2, X_3, \ldots, x_n)$. You can also assume that the distance between every two successive stops is less than or equal to d.
- You can assume that you are perfect at determining whether or not you can make it to the next stop in time.

PROBLEM 2 *Cross-Country trip planner part 2 (6 point)*

Prove that your algorithm always finds the optimal solution.

Problem 1 Answer:

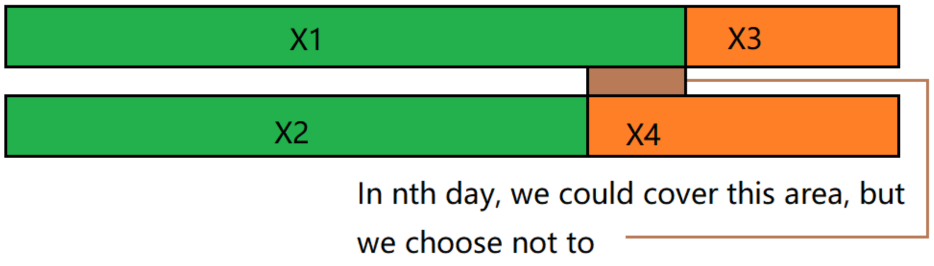
The greedy solution is go as further as they can every day. Assume d_l denotes the last miles they can go in one day, if the distance between them and the next stop is less than or equal to d_l, just go to the next stop, until distance between them and the next stop is greater than d_l, then take rests.

C++ code is as follows,

```
int problem1(vector<int>& input, int& d) {
    int count = 1;
    int cur = 0;
    for (int& i : input) {
        if ((cur += i) > d) {
            cur = i;
            ++count;
        }
    }
    return count;
}
```

Problem 2 Answer:

Assume there is a solution that contains one day, in which those people do not go as further as they can.



If we only drive x_2 miles instead of x_1 miles in the first n days(as we can cover x_1 in nth day) in nth day, it's obviously that consider the rest part, x_3 and x_4, the days we take to cover x_4 must be greater than or equal to the days we take to cover x_3(as the total distance is larger). And the final result must be greater than or equal to the former choice.

PROBLEM 3 *Kubal (Point 10)*

Every morning, customers: $1, \ldots, n$ show up to get their morning coffee from kubal. Suppose the omniscient barista actually knows all of the customers, and knows their orders, $o_1, o_2, o_3, \ldots, o_n$. All customers give tips (some give larger tips, some smaller); let T_i be the barista's expected tip from customer i. Some drinks like a simple double-shot, are easy and fast, while some other orders, like a hazelnut mocha soy-milk latte take longer. Let t_i be the time it takes to make drink o_i. Assume the barista can make the drinks in any order that he/she wishes, and that he/she makes drinks back-to-back (without any breaks) until all orders are done. Based on the order that he/she chooses to complete all drinks, let D_i be the time that the barista finishes order i. Devise an algorithm that helps the barista pick a schedule that minimizes the quantity:

$$\sum_{i}^{n} T_i D_i$$

In other words, he/she wants to serve the high-tippers the fastest but he/she also wants to take into consideration the time it takes to make each drink. (point 5) (Hint: think about a property that is true about an optimal solution.)

Prove the correctness of your algorithm. (point 5)

The solution is that we need to compare each two customers and sort the customers, the compare of customer A and customer B is based on $t_A \cdot T_B$ and $t_B \cdot T_A$, if the former is larger, than put customer A in front of customer B, otherwise, put customer B in front of customer A.

C++ code is as follows

```
vector<int> problem3(vector<int>& T, vector<int>& t) {
    vector<vector<int>> data(T.size());
    for (int i = 0; i < T.size(); ++i)
        data[i] = { i, T[i], t[i] };
    sort(data.begin(), data.end(),
    [](vector<int>& a, vector<int>& b) {
        return a[2] * b[1] < b[2] * a[1];
    });
    vector<int> result(T.size());
    for (int i = 0; i < data.size(); ++i)
        result[data[i][0]] = i;
    return result;
}
```

Correctness Proof

Assume we have already select the first $n-1$ customers with minimum sum of $Ti \cdot Di$, with $P = sum_{1 \to n-1}$ and $D = CurrentTime$. Then we need to choose which one is the nth customer to serve. Consider a base choice of customer A and customer B, who should be the nth to serve? There are only 2 choices, A in front of B, or B in front of A.

If we choose A in front of B, then the sum after we assigned both customer A and customer B P' is,

$P' = P + D \cdot (T_a) + (D + t_a) \cdot (T_b) = P + D \cdot (T_a + T_b) + ta \cdot Tb$

If we choose B in front of A, then the sum after we assigned both customer A and customer B P' is,

$P' = P + D \cdot (T_b) + (D + t_b) \cdot (T_a) = P + D \cdot (T_b + T_a) + tb \cdot Ta$

Notice that the difference is just $ta \cdot Tb - tb \cdot Ta$, NOT related to both P and D(that means this problem has optimal substructure). Then we could sort the customer array, compare customers each other and get the result.

PROBLEM 4 *Spanning Tree(Point 10)*

Let $G = (V, E)$ be a connected graph with distinct positive edge costs, and let T be a spanning tree of G (not necessarily the minimum spanning tree). The elite edge of T is the edge with the greatest cost. A spanning tree is said to be populist if there is no other spanning tree with a less costly elite edge. In other words, such a tree minimizes the cost of the most costly edge (instead of minimizing the overall cost).

- Is every minimum spanning tree of G a populist spanning tree of G? (Explain and prove your answer) (6 points)
- Is every populist spanning tree of G a minimum spanning tree? (Explain in details, no need for a proof) (4 points)

Question 1 True

Assume A is a minimum spanning tree of graph G, and B is a populist tree of graph G. At the same time, assume that A is NOT a populist tree of graph G.

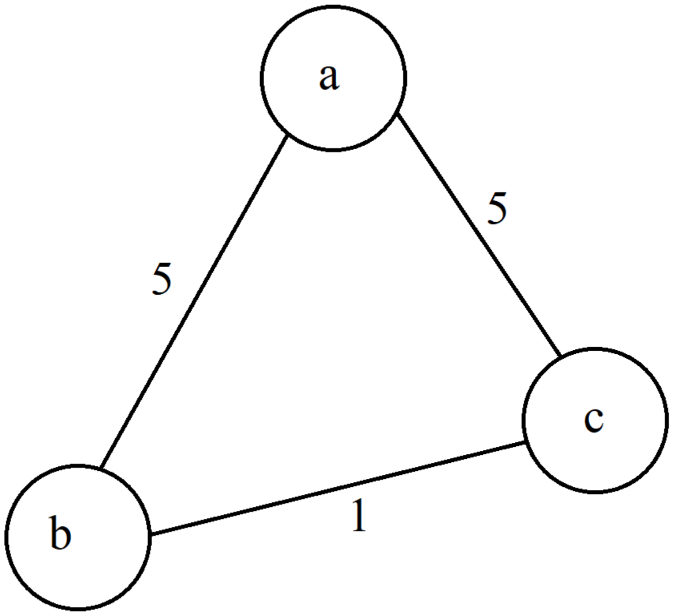
Then we assume E_1 is the most costly edge in A. We could cut down this edge and separate A to 2 cuts, cut A and cut B.

Notice that in tree B, every node must be connected to the whole tree. We definitely could find an edge E_2 in tree B to combine cut A and cut B, and because B is a populist tree but A is not. The cost of E_2 must be less than the cost of E_1, replace E_1 with E_2 in tree A, then we find the total cost of tree A is reduced, which means tree A is not a minimum spanning tree of graph G.

The assumption is invalid. Thus, the original proposition is true.

Question 2 False

Consider the following graph,



Notice that $\{ab, ac\}$ could be a populist tree. However, the minimum spanning tree is $\{ab, bc\}$ or $\{ac, bc\}$ So the populist spanning tree of G may not be a minimum spanning tree of G.

In fact, generating a populist tree ensure that the largest edge is the least. However, it do nothing with other edges, like the second largest edge, even though there is one edge to replace the second largest edge with a less one, the populist algorithm do nothing, it just take care of the largest one.