Jason Lai
204995126
Dis 1B

CS 180
April 25, 2019

Homework 3

1 Problem 3

For this greedy algorithm, the simple rule is that every box that arrives at the New York station should be, in order, packed in the one truck at the station until it reaches or is close to reaching its weight capacity W and can no longer take anymore boxes i of weight w_i. From this, we see an optimal solution can be obtained as we are choosing boxes to ship on a first come first serve basis, and shipping them on the first truck with available space, so they are shipped the fastest.

To show that this algorithm minimizes the number of trucks used, consider that there are i boxes b_1, b_2, ..., b_i for a given interval of time (business days) and n trucks t_1, t_2, ..., t_n, wherein the trucks do not make it back to the New York station until after this interval of time. Suppose there is another algorithm called "generous algorithm" that uses the approach the company has in mind, where trucks are sent off less than full, and also consider the current greedy algorithm in use. Say that the weight capacity W for each truck, in terms of boxes of fixed weight and size, is 10. For the greedy algorithm, it's obvious that of the i total boxes, the greedy algorithm can ship 10 * n boxes for n trucks by filling each of the n trucks to the brim, leaving i - (10 * n) boxes in reserve. Now consider the generous algorithm that sends trucks with a lighter load $l < 10$; to ship the same amount of boxes as the greedy algorithm, the generous algorithm must use k trucks such that $(k * l) \geq (10 * n)$. However, since we already established that $l < 10$, it is sufficient to say that, for this generous algorithm to match the greedy algorithm, it would need $k > n$ trucks to accomplish the same task in the same interval of time as the greedy algorithm, therefore the greedy algorithm is still the most optimal solution for minimizing the number of trucks to do work.

We can see that this relation scales for any number of trucks n and any weight capacity W: consider the expression $kL \geq nW$, where W is the weight capacity of each truck, L is the weight at which the trucks of the generous algorithm are sent off such that $L < W$, n is the number of trucks used in the current greedy algorithm, and k is the number of trucks needed for the generous algorithm to ship as many boxes as the greedy algorithm. From this expression, we see that k will always need to be greater than n in all cases.

2 Problem 7

Given that there is only one supercomputer and an essentially unlimited number of high-end PCs, from the problem, it is understood that the supercomputer can only run one job at a time, before handing it off to the unlimited number of high-end PCs. To get the shortest completion time, this would mean that all jobs will have to be ordered in terms of there high-end PC finishing time f_i from longest to shortest, since the supercomputer pre-processing time is trivial as all jobs will have to finish their pre-processing one at a time, so the total amount of pre-processing time P for all jobs is the same no matter the order. However, if the jobs with the longest finishing times are ran first, then they can be allocated one of the available high-end PCs the earliest after pre-processing and be finished sooner rather than later, while all other shorter jobs are also finishing in parallel. Therefore, a polynomial time algorithm that gives the schedule with the shortest possible completion time is to order jobs from longest to shortest in terms of their high-end PC finishing time f_i.

To show that this is the schedule that is the most optimal for completion time, consider two jobs i and j such that $f_i < f_j$; note that their pre-processing times p_i, p_j are negligible as the total amount of pre-processing time is the same no matter the order as mentioned before, but for the sake of the argument, suppose that $p_i = p_j < (f_j - f_i)$. Now consider two schedules A and B where A is the order i, j and B is the order j, i. From this, we see that schedule A's completion time is longer than B's as A will take $p_i + p_j + f_j$ time to complete, whereas B will only take $p_j + f_j$ time to complete. This is due to the fact that since schedule B processes and starts finishing the longer job j first, then while j is finishing, job i can process and finish in the meantime before j finishes, so i's pre-processing time is absorbed by the finishing time of j. From this, it is also evident that this idea extends to schedules with more than two jobs.

3 Problem 12

a) The statement is false, since the constraint only applies when the time interval starts at 0, so if the stream i has a bit length $b_i \geq rt_i$, the schedule is only invalid if i is first in the schedule when the time interval starts at 0. So, as long as i is not first in the schedule and a different stream j that has a bit length $b_j < rt_j$ goes first, then the schedule is valid.

b) An algorithm that takes a set of n streams, each specified by its number of bits b_i and its time duration t_i, and the link parameter r, and determines whether there exists a valid schedule will be to find all streams which have a bit length $b < rt$ and to have those streams start each new time interval so as to satisfy the constraint. The simplest way to do this would be to order the stream schedule in such a way that all streams that satisfy the constraint start first in each new time interval in the schedule; in essence, this would create a buffer for when the streams, such that $b \geq rt$, are then added to the schedule after all of the streams, such that $b < rt$ have been placed. This would run in $O(n)$, since computation of the constraint $b_i < rt_i$ for each stream i runs in $O(1)$ and adding the streams to the schedule runs in $O(n)$.

4 Problem 16

An algorithm to determine if a valid association exists between the account and the suspicious activity would be to iterate through the n recent events involving the account that took place at times (in chronological order) x_1, x_2, ..., x_n, then iterate through the n time intervals of suspicious activity t_1, t_2, ..., t_n and find an interval that contains a given event at time x_i. This is essentially an algorithm that runs in $O(n^2)$ that iterates through the n events x_1, x_2, ..., x_n and then iterates through the n intervals t_1, t_2, ..., t_n, matching each event to an interval. However, since each time interval t_i has a margin of error $\pm e_i$, this means that some time intervals may overlap, so an event x_i may match with multiple time intervals; in the event that this happens, the event x_i should be matched with the time interval that ends before the other potential, valid time intervals.

To show that this algorithm finds a valid association if there exists one, consider two events x_i and x_j such that x_i occurs before x_j so $x_i < x_j$, and consider two intervals t_m and t_n such that t_m is before t_n so t_m-$e_m < t_n$-$e_n < t_m$+$e_m < t_n$+e_n. Now, suppose that both x_i and x_j are both contained within the time span where t_m and t_n overlap; from this, we see that the valid association would have to be x_i matched with t_m and x_j matched with t_n, since it is understood that x_j comes later than x_i, so intuitively, interval t_n which covers a later span of time than t_m should be matched to x_j and t_m should be matched to x_i, which is exactly what the algorithm is doing.