CSC 373 - Algorithm Design, Analysis, and Complexity **Summer 2016**

Assignment 1: Due Tuesday June 7, 10PM

Please follow the instructions provided on the course website to submit your assignment. You may submit the assignments in pairs. Also, if you use **any** sources (textbooks, online notes, friends) please cite them for your own safety.
You can use those data-structures and algorithms discussed in CSC263 (e.g. merge-sort, heaps, etc.) and in the lectures by stating their name. You do not need to provide any explanation or pseudo-code for their implementation. You can also use their running time without proving them: for example, if you are using the merge-sort in your algorithm you can simply state that merge-sort's running time is $\mathcal{O}(n \log n)$.

Every time you are asked to design an efficient algorithm, you should provide both a short high level explanation of how your algorithm works in plain English, and the pseudo-code of your algorithm similar to what we've seen in class. State the running time of your algorithm with a brief argument supporting your claim. You must prove that your algorithm finds an optimal solution!

1 Nap Time

Recall the music festival scheduling problem we saw in class. Before heading to FestTown where it's held, you wanted to plan your best nap yet during the festival[1]. You know that the quietest and least busy time, at the camp where you're staying, is when a maximum number of shows are happening simultaneously. They don't necessarily overlap from start to end, but there is a non-empty time interval in which they overlap. You're given the schedule ahead of time.

1/ Give an efficient algorithm that computes the maximum number of overlapping performances.

2/Prove your algorithm is correct.

2 Hmm Cookies!

You got a baking gig on TV, and as you know, the most important thing about TV cooking is looks (taste comes second)! You need to make cookies live on-air. You lay the dough flat and cut out the pieces for each cookie, which you will then place on a baking tray with maximum length L (suppose all the cookies fit in the tray). But, as we know, we need to leave some space in between, since cookies grow when they bake. Suppose you leave a fixed length space between side-by-side cookies. One way to present the trays nicely is to fill the rows of the tray as much as possible, and move to the next row when the current cookie doesn't fit. One way to penalize "ugly" trays is to assign a penalty to each row that is not neat enough. For the sake of simplicity, suppose all the cookies have equal height.

Formally, you have trays of fixed length L, n cookies of width $d_1, d_2, , \ldots, d_n$ where $d_i \leq L$, for $i \in [n]$. A fixed distance $s = 1$ that represents the space you leave between cookies. If in row k, you place cookies i to

[1] who goes to festivals to nap...?

j, then the unoccupied space in this row is computed as follows:

$$L - \sum_{h=i}^{j} d_h - (j - i)$$

Suppose your penalty function is computed as follows:

$$row\_penalty(i,j) = \begin{cases} \infty & \text{if cookies } i \text{ through } j \text{ do not fit in a row} \\ 0 & \text{if } j = n, \text{ last row} \\ (L - \sum_{h=i}^{j} d_h - (j - i))^3 & \text{otherwise} \end{cases}$$

Notice that you don't pay any penalty for the last row of cookies.

The total penalty for placing the cookies is the sum of the penalties of all rows. An optimal solution is a placement of the cookies into rows such that your total penalty is minimized. Notice that you do not shuffle the order of the cookies to achieve the best looking trays.

1/ Give a recurrence relation to compute the optimal penalty.

2/ Construct an efficient algorithm which solves the above problem. Give a high-level description of how your algorithm works, and prove that it is optimal.

3/ Consider the case where the penalty function is just the free space in each tray, that is:

$$row\_penalty(i,j) = \begin{cases} \infty & \text{if cookies } i \text{ through } j \text{ do not fit in a row} \\ 0 & \text{if } j = n, \text{ last row} \\ L - \sum_{h=i}^{j} d_h - (j - i) & \text{otherwise} \end{cases}$$

Devise an algorithm to compute the optimal solution, and show that it is correct.

3 Work Work Work Work Work

You work at Amazon. Every day at work, a set of orders are assigned to you. Each order requires the same steps before it is shipped: pack, label and mail. Each order o_i has a processing time p_i. The orders are distributed every morning at work and you do not leave until your orders have been shipped for the day. **Employee Of The Month** is selected based on how long it takes an employee to ship an order[2]. The faster the better of course. Your goal this month is to win this title[3].

1/Describe the function you need to optimize in order to win. Formalize the input and output of the problem.

2/ In this first scenario, you are not allowed to switch back and forth between orders. Once you start working on an order, you cannot move to the next one until the current one is done. Give an algorithm that will help you win your title. What is the complexity of your algorithm? Prove that your algorithm is optimal.

3/ Suppose now the orders are not handed to you all at once, but rather come in one at a time. In addition to the processing time p_i, each order o_i has a release time r_i. In this scenario, you can switch back and forth between orders.

[2] On average of course

[3] Apologies for such a cr*ppy job...

Example: You get an order $o_i(r_i = 1, p_i = 8)$. You can for instance work on this order from time 1 to 3, get back to it from time 6 to 11 and then finish it from time 14 to 15, for a total of 8 units of time.

Give an algorithm that allows you to win in this case. Does switching between tasks improve the complexity? Prove that your algorithm is optimal.

4 Some Graph Theory .. Because you haven't seen enough.[4]

Let $G(V, E, w)$ be a directed edge-weighted graph: $w : E \to \mathbb{R}$. For a given cycle $\mathcal{C} = \{e_1, e_2, \ldots, e_k\}$ in G, let $\mu(\mathcal{C})$ denote the value:

$$\mu(\mathcal{C}) = \frac{1}{k} \sum_{i=1}^{k} w(e_i)$$

For all cycles $\{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_h\}$ in G, let $\tilde{\mu}$ be the minimum over $\mu(\mathcal{C}_i), i \in [h]$.

Let $s \in V$ be a source vertex in G, and suppose that every vertex $v \in V$ is reachable from s. Let w_{sv} (resp. w^k_{sv}) denote the weight of a shortest sv path, (resp. sv k-path[5]) in G. If no such k-path exists between s and v, we set w^k_{sv} to ∞.

1/ Prove that if $\tilde{\mu} = 0$, the following holds:

1. $w(\mathcal{C}) = \sum_{e \in \mathcal{C}} w(e) \geq 0$ for all cycles $\mathcal{C}$ in G.
2. $w_{sv} = \min_{0 \leq k \leq n-1} w^k_{sv}$ for all $v \in V$.
3. $\max_{0 \leq k \leq n-1} \frac{w^n_{sv} - w^k_{sv}}{n-k} \geq 0$.

2/ Let $\mathcal{C}^*$ be a cycle of total weight 0, and let x, y be two vertices on $\mathcal{C}^*$ where $w(P_{xy}) = d$, the weight of the xy path in $\mathcal{C}^*$. Show that $w_{sy} = w_{sx} + d$.

3/ Suppose $\tilde{\mu} = 0$ and let $\mathcal{C}^*$ be the cycle to achieve it (i.e. $\tilde{\mu} = \mu(\mathcal{C}^*)$), show that:

1. For some vertex v on $\mathcal{C}^*$, $\max_{0 \leq k \leq n-1} \frac{w^n_{sv} - w^k_{sv}}{n-k} = 0$.
2. $\min_{v \in V} \max_{0 \leq k \leq n-1} \frac{w^n_{sv} - w^k_{sv}}{n-k} = 0$

4/ How does $\tilde{\mu}$ change if we increase the weight of all edges by a constant c? Show that $\tilde{\mu} = \min_{v \in V} \max_{0 \leq k \leq n-1} \frac{w^n_{sv} - w^k_{sv}}{n-k}$.

5/ Devise an algorithm to compute $\tilde{\mu}$. Show that your algorithm runs in $\mathcal{O}(mn)$ time.

[4] Seriously though..

[5] A path using exactly k edges