CS3000: Algorithms & Data — Summer I ’21 — Drew van der Poel

Homework 4
Due Saturday, June 12 at 11:59pm via Gradescope

Name:
Collaborators:

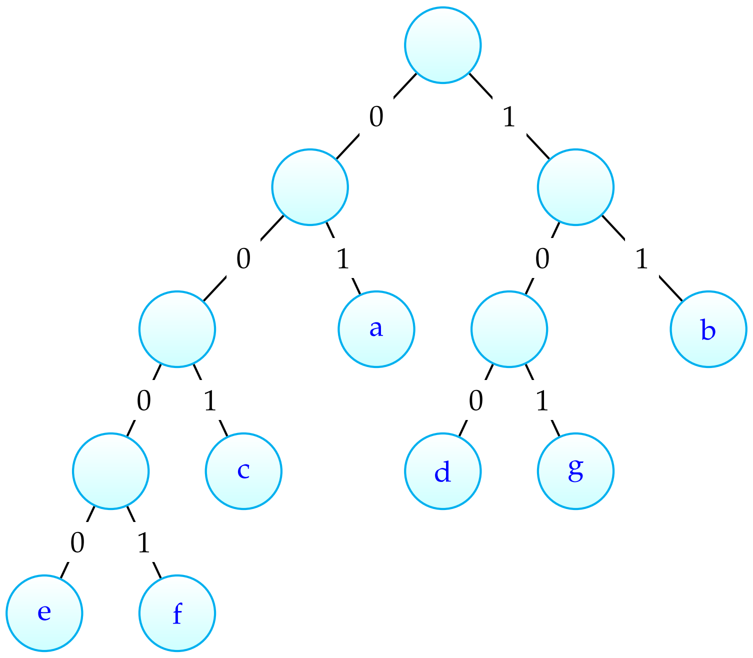
- Make sure to put your name on the first page. If you are using the LaTeX template we provided, then you can make sure it appears by filling in the `yourname` command.
- This assignment is due Saturday, June 12 at 11:59pm via Gradescope. No late assignments will be accepted. Make sure to submit something before the deadline.
- Solutions must be typeset in LaTeX. If you need to draw any diagrams, you may draw them by hand as long as they are embedded in the PDF. I recommend using the source file for this assignment to get started.
- I encourage you to work with your classmates on the homework problems. *If you do collaborate, you must write all solutions by yourself, in your own words.* Do not submit anything you cannot explain. Please list all your collaborators in your solution for each problem by filling in the `yourcollaborators` command.
- Finding solutions to homework problems on the web, or by asking students not enrolled in the class is strictly forbidden.

Problem 1. *Huffman Code Example* (10 points)

Given the following alphabet $\Sigma = \{a, b, c, d, e, f, g\}$ and corresponding frequencies, use Huffman's algorithm to compute an optimal prefix-free code. Represent the prefix-free code as a binary tree, with either 0 or 1 on each edge. Compute the expected encoding length.

Letter	a	b	c	d	e	f	g
Frequency	0.25	0.21	0.2	0.1	0.08	0.05	0.11

Solution:



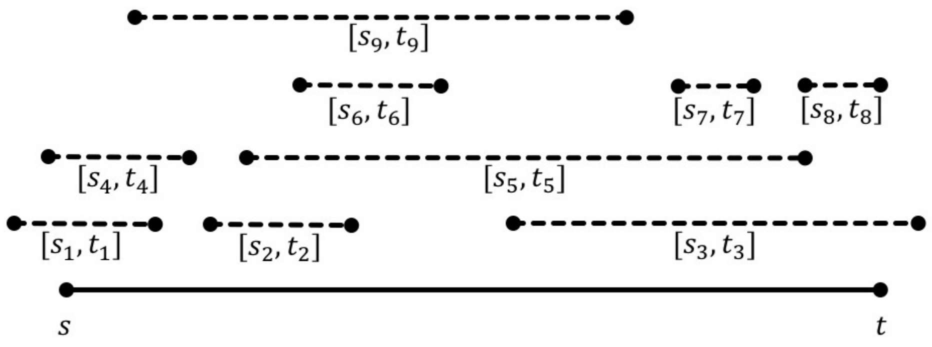
To compute the expected encoding length, we would do the following:
$f_a * len(a) + f_b * len(b) + f_c * len(c) + f_d * len(d) + f_e * len(e) + f_f * len(f) + f_g * len(g)$
$= (0.25 * 2) + (0.21 * 2) + (0.2 * 3) + (0.1 * 3) + (0.08 * 4) + (0.05 * 4) + (0.11 * 3)$
$= 2.67$

Problem 2. *I wish I was a baller* (32+4 points)

As president of the Northeastern chapter of the Big Ballers Club, you need to staff a booth at this year's open house. The open house runs for the interval $[s,t]$. There are n volunteers, each of which is able to cover the booth for the interval $[s_i,t_i]$ ($i \in \{1,\dots,n\}$). You need to select a set of volunteers $S \subseteq \{1,\dots,n\}$ to *cover* the entire open house, meaning that $\bigcup_{i\in S}[s_i,t_i] \supseteq [s,t]$. Equivalently, for every time $z \in [s,t]$, there is some volunteer $i \in S$ such that $z \in [s_i,t_i]$. Each member will be paid for their services, but ironically the Big Ballers Club has very limited funds, so you need to ensure $|S|$ is as small as possible.

In this problem you will design an efficient greedy algorithm that takes as input the numbers $s,t,s_1,t_1,\dots,s_n,t_n$ and outputs a set S that covers the open house and uses the minimum number of staffers. The running time of your algorithm should be at most $O(n^2)$. You may assume that a solution exists.

The following is an example input with 9 volunteers. One optimal solution is $S = \{1,3,9\}$.



(a) (**8 points**) Describe your algorithm in pseudocode. Provide a few sentences describing your algorithm.

Solution:

Algorithm 1: I wish I was a baller

```
Function $CoverOpenHouse(s,t,s_1,\dots,s_n,t_1,\dots,t_n)$:
    $S = \{\}$
    $currentTime \leftarrow s$
    While $currentTime < t$
        $nextVolunteer \leftarrow argmax_{1\le i\le n} t_i$ where $s_i \le currentTime$
        $S.add(nextVolunteer)$
        $currentTime \leftarrow t_{nextVolunteer}$
    Return S
```

We use the variable $currentTime$ to represent the next time we would need to begin coverage, initially set to s. We will repeatedly find the interval that includes $currentTime$ and has latest finish time, select it as part of our solution, and update $currentTime$ accordingly.

(b) (**4 points**) Analyze the running time of your algorithm.

Solution:

Time complexity: $O(n^2)$

The outer loop will run a maximum of n times. The inner argmax function will consider a maximum of n values as each volunteer is considered at most once. Updating S and $currentTime$ are constant time operations. Thus, the total runtime is at worst $O(n^2)$.

(c) (**12 points**) Prove that your algorithm finds a set of volunteers S of minimum size to cover the open house.

Solution:

We will prove this by showing that ***Greedy always stays ahead***.

Proof. Let $G_1, G_2, ..., G_l$ be the greedy solution, and let $O_1, O_2, ..., O_m$ be any other solution. We will show by induction that $f_{G_i} \leq f_{O_i}$ for all $i \in [1, ..., l]$ (noting that the greedy solution is a valid solution as it covers all points from s to t, inclusively). This implies that $l \leq m$, as otherwise the greedy algorithm would stop after selecting G_m (since this would cover the entire interval).

Base: The first interval in both solutions (G_1 and O_1) has to include s. By definition, G_1 is the interval with the latest endtime which includes s, thus the claim for $i = 1$ is true ($f_{G_1} \leq f_{O_1}$).

Inductive: We will assume the claim is true for all $i \in [1, k-1]$.

Assume that the claim doesn't hold for k, that $f_{O_k} > f_{G_k}$. By the inductive hypothesis, we know that $f_{G_{k-1}} \geq f_{O_{k-1}}$.

By definition we know that interval G_k was the latest finishing one which covers the time $f_{G_{k-1}}$. Because $f_{O_{k-1}} \leq f_{G_{k-1}}$, G_k finishes no earlier than any interval which begins by time $f_{O_{k-1}}$ (as otherwise we would pick such an interval). Thus, if $f_{O_k} > f_{G_k}$, we would have picked interval O_k in the greedy approach.

Because the greedy approach selects the interval with latest finish time, $f_{G_k} \geq f_{O_k}$, thus giving a contradiction.

□

(d) **[8 points]** In order to better your familiarity with greedy algorithms, we have created a hackerrank challenge (www.hackerrank.com/cs3000-summer1-2021-programming-assignment-4). Please implement your greedy strategy and submit it to the challenge. Your grade for this part will depend on (a) how many test cases your implementation passes and (b) actually implementing a greedy strategy (we will check!).

In order to allow for efficient grading, please write your hackerrank account below.

Solution:

(e) **[+4 points]** Describe a greedy algorithm which solves this problem in $o(n^2)$ time. State and justify your runtime.

Solution:

We can sort the intervals by their start time so that the i-th interval has the i-th earliest start time. We will make a single pass over the intervals as follows:

Algorithm 2: I wish I was a baller

```
Function $CoverOpenHouse(s, t, s_1, \ldots, s_n, t_1, \ldots, t_n)$:
    $cover = s$
    $S = \{\}$
    $temp = 0, f_0 = 0$
    For $i = 1, \ldots, n$
        If $s_i > cover$ :
            $S = S \cup temp$
            $cover = f_{temp}$
            $temp = i$
        If $f_i > f_{temp}$ :
            $temp = i$
    Return S
```

The idea is that we initially want to cover the starting point s. If when iterating, we go past this point, we need to select the interval that we have seen with the latest finishing time. We store this with the second if-condition in the $temp$ variable. Once we pass this point, we update the point to cover to be f_{temp} (the finish time of the recently added interval). We check for when we pass this point in the first if-condition.

Sorting the intervals takes $O(n * log(n))$ time. Otherwise, our for loop iterates n times, and each is $O(1)$ time. Thus, the sort dominates the runtime and the total time is $O(n * log(n))$.

Problem 3. *Total Weighted Finish Time* (24 points)

A scheduler needs to determine an order in which a set of n processes will be assigned to a processor. The i-th process requires t_i units of time and has weight w_i. For a given schedule, define the finish time of process i to be the time at which process i is completed by the processor. (Assume that the processor starts processing the tasks at time 0.)

For example, consider 4 processes with $t_1 = 5, t_2 = 2, t_3 = 7, t_4 = 4$. And weights $w_1 = 1, w_2 = 3, w_3 = 2$, and $w_4 = 2$. Consider the schedule 1,3,2,4. The finish time for process 1 is 5, for process 2 is $t_1 + t_3 + t_2 = 5 + 7 + 2 = 14$, for process 3 is $t_1 + t_3 = 5 + 7 = 12$, and for process 4 is $t_1 + t_3 + t_2 + t_4 = 5 + 7 + 2 + 4 = 18$. The total weighted finish time of the schedule is then $5 * 1 + 14 * 3 + 12 * 2 + 18 * 2 = 5 + 42 + 24 + 36 = 107$.

(a) **[10 points]** Give a greedy algorithm to determine a schedule that minimizes the total weighted finish time. (**Hint:** If you are stuck, trying figuring out a rule that *always* works when $n = 2$).

Solution:

Let w_i/t_i be the impact of job i. Sort the jobs in descending order by impact, and let this be the schedule.

This is greedy as it is equivalent to always selecting the remaining job with the highest impact.

(b) **[10 points]** Prove the correctness of your algorithm.

Solution:

We will say that two jobs i and j are inverted in a schedule if i has larger impact than j, but j precedes i in the schedule. Note that the greedy schedule has no inversions. We will show that any schedule with inversions is not optimal.

Assume false, that some schedule T with inversions is better (has smaller total weighted finish time) than the greedy schedule from part (a). There must be a pair i, j of consecutively scheduled jobs in T that are inverted. To see this, note that there is some pair of jobs in T that are inverted, a and b. Job a has higher impact than job b, but b comes before a in T. If we consider the series of jobs in T from b to a, at some point a job must have lower impact than the job after it. Let i and j be inverted and consecutive, with job j having lower impact and preceding job i.

Consider flipping jobs i and j to create schedule T'. Note that the only difference in the total weighted finish time between T and T' will be the contribution of jobs i and j. We will say that job j begins at time x in T.

In T, the combined contribution of i and j is: $(x + t_j)(w_j) + (x + t_j + t_i)(w_i)$

In T', the combined contribution of i and j is: $(x + t_i)(w_i) + (x + t_j + t_i)(w_j)$

We will show that the contribution is less in T'.

$(x + t_i)(w_i) + (x + t_j + t_i)(w_j) \leq (x + t_j)(w_j) + (x + t_j + t_i)(w_i)$

$xw_i + w_i t_i + xw_j + t_j w_j + t_i w_j \leq xw_j + t_j w_j + xw_i + t_j w_i + t_i w_i$

$t_i w_j \leq t_j w_i$

$w_j/t_j \leq w_i/t_i$ which we know is true because j has lower impact than i.

Thus, the optimal schedule has no inversions.

It remains to show all schedules with no inversions have the same total weighted finish time. There can only be multiple schedules with no inversions if multiple jobs have the same impact. We will show that flipping the order of two jobs with the same impact will not change the total weighted finish time. Note that applying a series of these flips will allow us to get from one inversion-less schedule to another, and none of the moves will increase the total weighted finish time.

Assume jobs p and q have the same impact and are scheduled consecutively in schedules S and S', but p comes first in S and q comes first in S'. The only difference in the weighted finish time of S and S' will be the contribution of p and q. Assume p starts at time x in S.

Contribution of p and q in S: $(x+t_p)(w_p)+(x+t_p+t_q)(w_q)$

Contribution of p and q in S': $(x+t_q)(w_q)+(x+t_p+t_q)(w_p)$

We will now show these are equal.

$(x+t_p)(w_p)+(x+t_p+t_q)(w_q)=(x+t_q)(w_q)+(x+t_p+t_q)(w_p)$

$t_p w_q = t_q w_p$

$w_q/t_q = w_p/t_p$ which is true as p and q have the same impact.

Thus, any schedule with no inversions is optimal, and the greedy solution is optimal.

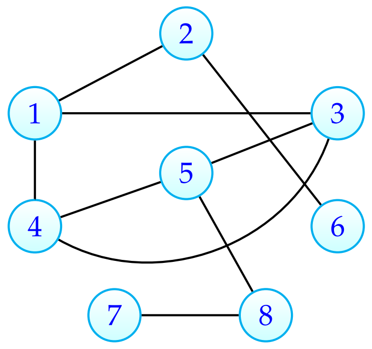
(c) **[4 points]** Analyze the worst-case running time of your algorithm.

Solution:

It is $O(n)$ to compute the impact of each job, and $O(n\log(n))$ to sort the jobs based on impact. Thus, the total runtime is $O(n\log(n))$.

Problem 4. *Graph Representations and Exploration (15 points)*

This problem is about the following graph. **Tip:** Use the LaTeXfor rendering the graph as a starting point if your solutions involve drawing a graph.



(a) **[5 points]** Draw the adjacency matrix of this graph.[1]

Solution:

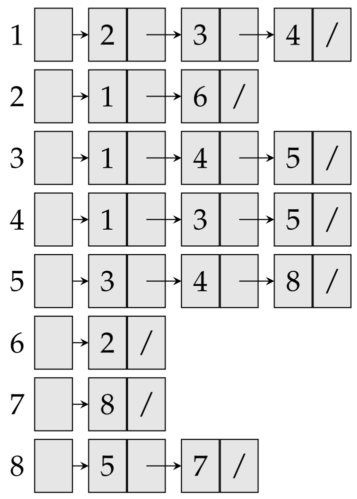
$$\begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

(b) **[5 points]** Draw the adjacency list of this graph.

Solution:

We have the following adjacency list.

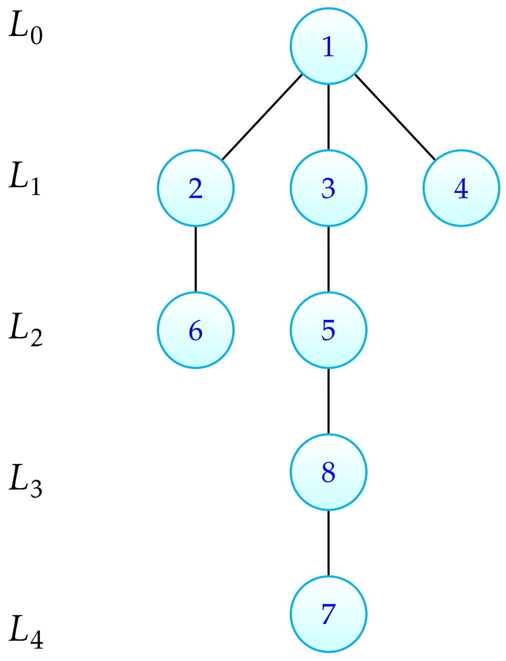
[1]**Hint:** If drawing the matrix in latex, I recommend using the `tabular` or `matrix` environments.



(it is fine to not use this linked list format)

(c) **[5 points]** BFS this graph starting from the node 1. Always choose the lowest-numbered node next. Draw the BFS tree and label each node with its distance from 1.

Solution:



Problem 5. *Graph Properties (10 points)*

Consider an undirected graph $G = (V, E)$. The *degree* of a vertex v is the number of edges adjacent to v—that is, the number of edges of the form $(v, u) \in E$. Recall the standard notational convention that $n = |V|$ and $m = |E|$.

(a) Prove by induction that the sum of the degrees of the vertices is equal to $2m$.

Solution:

Consider any number of vertices $n \in \mathbb{N}$. We will prove that for any graph on n vertices, for every number of edges $m \in N$, the sum of the degrees is $2m$. We prove this by induction on the number of edges m.
Base Case: For any n, when $m = 0$ all degrees are 0, so the base case holds.
Inductive Step: We assume the inductive hypothesis that for any graph with n vertices and $m - 1$ edges, the sum of degrees is $2(m - 1)$. Let G be any graph with n nodes and m edges. Consider an arbitrary edge $e = (u, v)$ and let G' be G with the edge e removed. G' has n nodes and $m - 1$ edges, so the sum of its degrees is $2(m - 1)$. The sum of degrees in G is thus $2(m - 1) + 2 = 2m$ (because the edge (u, v) increases the degrees of u and v each by one). Thus the claim holds by induction.