C2basic(MST3K)

Collaboration Policy: You are encouraged to collaborate with up to 3 other students, but all work submitted must be your own *independently* written solution. List the computing ids of all of your collaborators in the `collabs` command at the top of the tex file. Do not share written notes, documents (including Google docs, Overleaf docs, discussion notes, PDFs), or code. Do not seek published or online solutions for any assignments. If you use any published or online resources (which may not include solutions) when completing this assignment, be sure to cite by naming the book etc. or listing a website's URL. Do not submit a solution that you are unable to explain orally to a member of the course staff. Any solutions that share similar text/code will be considered in breach of this policy. Please refer to the syllabus for a complete description of the collaboration policy.

Collaborators: list collaborators's computing IDs

Sources: Cormen, et al, Introduction to Algorithms. *(add others here)*

PROBLEM 1 *Load Balancing*

You work for a print shop with 4 printers. Each printer i has a queue with n jobs: $j_{i,1}, \dots, j_{i,n}$. Each job has a number of pages, $p(j_{i,m})$. A printer's workload $W_i = \sum_\ell p(j_{i,\ell})$ is the sum of all pages across jobs for for that printer. Your goal is to *equalize* the workload across all 4 printers so that they all print the same number of total pages. You may only remove jobs from the end of their queues, i.e., job $j_{i,n}$ must be removed before job $j_{i,n-1}$, and you are allowed to remove a different number of jobs from each printer. Give a **greedy algorithm** to determine the maximum equalized workload (possibly 0 pages) across all printers. Be sure to state your greedy choice property.

Solution: Because we know the $p(j_{i,m})$ function, we can simply use a method similar to that of coin selections. Namely, we look at the first element in each printer's job queue and select the printer with the largest job. We then select more of the other printer's jobs (from the beginning) until we either reach the selected job length or we exceed it. In the case that any other printer exceeds the largest job, we then select the printer with the largest job sequence length (sum of pages from beginning to current) and repeat the process to get the other printers to match that value. In the case that all the job sequences match, we store the saved value as a value to potentially return and then repeat the process. The greedy property is that we can keep "adding" jobs until we either match another printer's sequence of jobs' length or we exceed it. We can keep making this selection until all the lengths match up or we repeat.

After this process is completed, we remove jobs from each printer until we get to our largest equal-length value determined using the prior steps. Some pseudo-code is listed below:

```
jobequalization( printer1[], printer2[], printer3[], printer4[] ){
    int i, j, k, l = 0;
    printerOfInterest = max(printer1[i], printer2[j], printer3[k], printer4[l]) // stores on

    best

    while ( i < printer1.length &  j < printer2.length & k < printer3.length & l < printer4.
        while otherprinters[index] < printerOfInterest[index]
            increment otherprinters' indeces

        if all printers[index] is equal
            best = printers[index]

        else
            continue
    }

    while printers[indeces] != best
        remove jobs from printers

}
```

PROBLEM 2 *Short Answer Questions. (You don't have to explain your answers in your submission, but you should understand the reason behind your answer.)*

A. True or false? Issuing the largest coin first will always solve the *coin change problem* if only two coins are available: the penny and one larger coin. Assume the amount of change is $\geq$ the larger coin.
Solution: True? The greedy method is to repeatedly select the larger coin until we cannot anymore. This proposed solution is similar enough to the greedy strategy.

B. True or false? The *interval scheduling problem* is always guaranteed to have an optimal solution that contains the interval with the earliest finish time.
Solution: True. By definition, the earliest finish time interval is guaranteed to be a part of an optimal solution.

C. Choose one: In our proof of the correctness of the greedy solution to the *interval scheduling problem*, we exchanged the interval i selected by our greedy choice with another interval that finished earlier/later than interval i. (Your answer is one of "earlier" or "later.")
Solution: Later I believe? This was the lecture on April 5th towards the end. We select the next interval a that ends immediately after our selected earliest ending interval $a*$.

D. True or false? A *feasible solution* for the *Huffman encoding problem* is any valid prefix-free code-word table T.
Solution: True? This does seem like a *feasible solutoin* for the encoding problem.

PROBLEM 3 *Optimal Substructure*

Please answer the following questions related to *Optimal Substructure*.

A. What's the difference in how dynamic programming algorithms versus greedy algorithms use *optimal substructure*?

Solution: While both have the substructure that solutions to larger problems contain optimal solutions to smaller subproblems, greedy algorithms assume that there is only one smaller subproblem to consider. Dynamic programming algorithms consider multiple smaller subproblems instead. This is what enables greedy algorithms to construct a optimal solution adding to the current optimal solution instead of having to perform pseudo-recursive calls to find and compare multiple potentially optimal solutions.

B. Why did we need to prove the optimal substructure for our greedy Huffman coding algorithm?

Solution: We need to prove the optimal substructure for our greedy Huffman coding algorithm because we can abstract away the details of some of the subproblems that we are solving. Namely, the organization of the other collations of children (σ). The Huffman coding algorithm's optimal substructure is that by simply adding a σ value (with its own frequencies and lengths) to a tree, we indirectly make a decision about the organization about the children of that σ. Therefore, by proving the optimal substructure for our greedy Huffman coding problem, we can show that making greedy decisions about the organization of σ leads us to an optimal solution.

PROBLEM 4 *Gradescope Submission*

Submit a version of this `.tex` file to Gradescope with your solutions added, along with the compiled PDF. You should only submit your `.pdf` and `.tex` files.