CS4810 / CS7800: Advanced Algorithms — Fall 2022
Homework 3: Due Friday, October 14, 2022

Mahsa Derakhshan and Jonathan Ullman

Collaboration and Honesty Policy Reminder. Collaboration in the form of discussion is allowed and encouraged. However, all forms of cheating are not allowed and will be penalized harshly. These include, but are not limited to, copying parts of an assignment from a classmate, finding answers to problems on the internet or from anyone not enrolled in the class, and plagiarizing from research papers or old posted solutions. A rule of thumb is that you should be able to walk away from discussing a homework problem with no notes and write your solution on your own.

- You must write up all solutions by yourself, and may not share any written solutions, even if you collaborate with others to solve the problem.
- You must identify all your collaborators. If you did not collaborate, write “no collaborators" or something to that effect. You may have a maximum of two collaborators per assignment, and collaboration is transitive (if you list a collaborator they must list you).
- Asking and answering questions in class forums (lectures, office hours, Piazza) is allowed and encouraged, and you do not need to list these interactions as collaborators.
- Seeking out alternative sources (e.g. classmates, textbooks, the internet) for general concepts you need (e.g. greedy algorithms, probability) is allowed and encouraged.

Assigned Problems

The next two questions ask for dynamic programming algorithms. For each question, include:

- *A description of the subproblems your algorithm will solve (e.g. “Let* $\textsc{Opt}(i)$ *be the optimal...").*
- *A recurrence relating those subproblems and a justification for its correctness.*
- *A pseudocode description of your algorithm for finding the optimal solution.*
- *An analysis of your algorithm’s running time.*

Problem 1 You are the head pastry chef for the coronation of King Charles III and you are given instructions to prepare n desserts and a team of m pastry chefs to assign to work on the different desserts. All of the chefs are equally competent, but each chef can work on only one dessert, and you have to decide how many chefs to assign to each dessert to make the most delicious feast possible.

After some careful thought, you have figured out how delicious dessert i will be if j chefs are assigned to it, and you quantify this with a deliciousness value $a_{i,j} \geq 0$. You may assume that more chefs will make a dessert more delicious, so $a_{i,j+1} \geq a_{i,j}$ for every i and j. You may also assume that the dessert can’t be made without at least one chef, so $a_{i,0} = 0$ for every i.

Design a dynamic programming algorithm to determine how many chefs to assign to each dessert so that the sum of the deliciousness values is maximized.

Problem 2 Alice and Bob play the following game. There is a row of n tiles with values $x_1, \ldots, x_n$ written on them. Starting with Alice, Alice and Bob take turns removing either the first or last tile in the row and placing it in their pile until there are no tiles remaining. For example, if Alice takes tile 1, Bob can take either tile 2 or tile n on the next turn. At the end of the game, each player receives a number of points equal to the sum of the values of their tiles minus that of the other player's tiles. Specifically, if Alice takes tiles $A \subseteq \{1, \ldots, n\}$ and Bob takes tiles $B = \{1, \ldots, n\} \setminus A$, then their scores are, respectively,

$$\underbrace{\sum_{i \in A} x_i}_{\text{Alice}} \quad \text{and} \quad \underbrace{\sum_{i \in B} x_i}_{\text{Bob}}$$

For example, if the tiles are $10, 2, 8$ then taking the first tile guarantees Alice a score of at least $10 + 2 = 12$, whereas taking the last tile would only guarantee Alice a score of at least $8 + 2 = 10$.

Design an algorithm to determine the maximum score that Alice can guarantee for herself, assuming Bob plays optimally to maximize his score.[1]

Problem 3 Let G be a flow network with source s, sink t, and integer capacities $c(e)$ on each edge e. Call an edge e an *acute edge* in G if e belongs to *every minimum s-t cut* of G. There can be many minimum cuts, so we can't enumerate them, but there is an easier way to determine if an edge e is acute.

3.1 Prove that an edge e is acute if and only if increasing its capacity by 1 increases the maximum flow of the network by 1.

3.2 Using the result of 3.1, describe an algorithm that takes a graph G and a maximum flow f in that graph and determines if an edge e is acute. Your algorithm should be more efficient than computing a new maximum flow from scratch—ideally just $O(m)$ time. Justify that your algorithm is correct and analyze its running time.

Problem 4 Suppose that instead of capacities, we consider networks where each edge e has some non-negative, integer-valued *demand* $d(e) \geq 0$. We say that an (s, t)-flow f is *feasible* if $f(e) \geq d(e)$ for every edge e, in addition to satisfying the flow-conservation constraints. A natural problem in this setting is to find a feasible flow of *minimum* value.

4.1 Describe an efficient algorithm to compute a *feasible* (s, t)-flow (not necessarily of minimum value). Justify your algorithm's correctness, and analyze its running time.

4.2 Suppose you have access to a subroutine MaxFlow that solves the maximum s-t-flow problem. Describe an efficient algorithm to compute a minimum flow in a network with edge demands. Your algorithm should call MaxFlow exactly once.[2] Justify that your algorithm is correct and analyze its running time, including the time required to execute MaxFlow.

4.3 State and prove an analogue of the max-flow/min-cut theorem for this setting. Do minimum flows correspond to maximum cuts for this problem?

---

[1] **Hint:** Note that the sum of their scores is always equal to the same value, $\sum_{i=1}^{n} x_i$, so if Bob is playing optimally to maximize his own score, then he is also playing optimally to minimize Alice's score.

[2] **Hint:** Start with the feasible (s, t)-flow your algorithm finds in 4.1 and try to *remove* as much flow as possible while still satisfying the demands.