Colorado CSCI 5454: Algorithms
Homework 3

Instructor: Bo Waggoner
Due: September 19, 2019 at 11:59pm
Turn in electronically via Gradescope.

Problem 1 (12 points)

In the *Longest Common Subsequence* problem, the input consists of two strings, x and y. The output should be the length of the longest sequence of characters that is a subsequence of both x and y.

Recall that a subsequence does not have to be *consecutive*. For example, if $x = ALGORITHM$ and $y = ANARCHISM$, then $ARHM$ is a subsequence of both, and so is $ARIM$.

(This is used by the `diff` utility and version control systems like `git` to compare text files line-by-line. In these cases, each line of a document is treated as a "character" and we want to find the longest common subsequence of lines.)

Part a (10 points) Give a dynamic programming algorithm to solve the longest common subsequence problem. Clearly identify the definition of a subproblem *(2 points)* and the recurrence (i.e. how to solve a subproblem given previous subproblem solutions, *2 points*), and justify correctness of your recurrence *(2 points)*. Then give the full algorithm *(2 points)*, and briefly argue running time and space usage *(2 points)*.

Part b (2 points) Explain how to modify your algorithm to return the subsequence itself.

Problem 2 (10 points)

(Max flow with vertex capacities.)

Recall the *maximum s to t flow* problem:

We are given a directed graph $G = (V, E)$ and two vertices $s, t \in V$, and we must output a function $f : E \to \mathbb{R}_{\geq 0}$. We say f is a *valid flow* if it satisfies *flow conservation* constraints and *edge capacity* constraints.

The flow constraints are:

$$\sum_{e \in \delta^{in}(v)} f(e) = \sum_{e \in \delta^{out}(v)} f(e) \qquad \forall v \in V \setminus \{s, t\} \tag{1}$$

where $\delta^{in}(v)$ is the set of incoming edges of v and $\delta^{out}(v)$ is the set of outgoing edges of v. This says the total flow into vertex v must equal total flow out. (Notice the source s and sink t are exempt from this requirement.)

We are also given an *edge capacity function* $c : E \to \mathbb{R}_{\geq 0}$ and the edge capacity constraints are:

$$f(e) \leq c(e) \qquad \forall e \in E. \tag{2}$$

This says the flow through an edge is at most its capacity.

The *value* of a flow f is then total flow going into t, which is $\sum_{e \in \delta^{in}(t)} f(e)$. We cover in class how to find the maximum-value s to t flow in polynomial time. Now for this problem, suppose we are also given a *vertex capacity* function $w : V \to \mathbb{R}_{>0}$. Now we say a flow is valid if it satisfies 1, 2, and

$$\sum_{e \in \delta^{in}(v)} f(e) \leq w(v) \qquad \forall v \in V \setminus \{s, t\}. \tag{3}$$

These are called the *vertex capacity* constraints.

Example: We wish to route traffic over the road network from s to t where road segments are edges and intersections are vertices. Both road segments and intersections can only handle a certain rate of traffic flow, so we have both edge and vertex capacity constraints.

Part a (2 points) Suppose that VERTCAPALG is an algorithm which solves the maximum s to t flow problem when edge and vertex capacities are specified. Describe how VERTCAPALG can be used to solve the original maximum s to t flow problem when only edge capacities are specified.

Part b (4 points) Give an efficient algorithm to solve the maximum s to t flow problem when edge and vertex capacities are specified. Argue correctness and running time. *(Hint: You should reduce to a problem we already know how to solve efficiently.)*

Part c (2 points) Now suppose that we have a variant of the max s to t flow problem where *only* vertex capacities $w : V \to \mathbb{R}_{>0}$ are given. Now, each edge can support any amount of flow. Describe how VERTCAPALG can be used to solve this problem.

Part d (2 points) Given a directed graph G and vertices s, t, an *s to t vertex cut* is a set of vertices such that, if they are removed from the graph, there is no path from s to t. If we are also given a vertex weight function $w : V \to \mathbb{R}_{>0}$, a *min s to t vertex cut* is a vertex cut S of minimum total weight, i.e. minimizing $\sum_{v \in S} w(v)$.

Describe an algorithm for finding a min s to t vertex cut. You do not need to argue correctness and efficiency (but it should be correct and efficient).

Hint: recall the max-flow min-cut theorem and your solution to the previous parts.

Problem 3 (14 points)

(Programming and theory assignment.)

You are a cashier whose goal is to give customers their correct change while using as few total coins as possible. The input is the set of available coin denominations as integers

$x_1, \ldots, x_n$; and the amount of change that is due, an integer W. The output is the fewest number of total coins that add up to W.

For example, the American coin denominations are $x_1 = 1$ (the penny), $x_2 = 5$ (the nickel), $x_3 = 10$ (the dime), $x_4 = 25$ (the quarter), $x_5 = 50$ (the 50-cent piece), and $x_6 = 1$ (the dollar coin). If $W = 17$, the optimal solution is a dime, a nickel, and two pennies for a total of 4 coins.

You may assume that $x_1 = 1$ and that $x_i < x_{i+1}$ for each $i = 1, \ldots, n-1$.

Part a (4 points) The *greedy* algorithm for this problem is: give as many as possible of the largest coin denomination without going over W; then as many of the the next-largest as possible, and so on.

Implement the greedy algorithm in a programming language of your choice and attach the source code. Run it on the following inputs and report the number of coins required as well as a list of how many of each coin is used.

- The U.S.A. coin denominations with $W = 42$.
- The U.S.A. coin denominations with $W = 1728$.
- Coin denominations $1, 8, 20, 30, 80, 200$ and $W = 42$.
- The previous denominations and $W = 1728$.

Part b (2 points) Give an example showing that the greedy algorithm is not always optimal.

Part c (4 points) Give a dynamic programming algorithm to solve this problem optimally. Argue correctness, running time, and space use.

Part d (4 points) Implement your dynamic programming algorithm in a language of your choice (attach source code) and run it on the examples from part (a); report the results.