Computer Science C73 — October 25, 2023
Scarborough Campus — University of Toronto

Homework Assignment #6
Due: November 8, 2023, by 11:59 pm

• ***You must submit your assignment through the Crowdmark system.*** *You will receive by email an invitation through which you can submit your work in the form of separate PDF docucments with your answers to each question of the assignment. To work with a partner, you and your partner must form a group on Crowdmark. Crowdmark does not enforce a limit on the size of groups.* ***The course policy that limits the size of each group to at most two remains in effect:*** *submissions by groups of more than two persons will not be graded.*
• *It is your responsibility to ensure that the PDF files you submit are legible. To this end, I encourage you to learn and use the LaTex typesetting system, which is designed to produce high-quality documents that contain mathematical notation. You are not required to produce the PDF files you submit using LaTex; you may produce it any way you wish, as long as the resulting document is legible.*
• *By virtue of submitting this assignment you (and your partner, if you have one) acknowledge that you are aware of the policy on homework collaboration for this course.*[a]
• *For any question, you may use data structures and algorithms previously described in class, or in prerequisites of this course, without describing them. You may also use any result that we covered in class, or is in the assigned sections of the official course textbooks, by referring to it.*
• *Unless we explicitly state otherwise, you may describe algorithms in high-level pseudocode or point-form English, whichever leads to a clearer and simpler description. Do not provide executable code.*
• *Unless we explicitly state otherwise, you should justify your answers. Your paper will be graded based on* ***the correctness and efficiency of your answers, and the clarity, precision, and conciseness of your presentation****.*

[a] "In each homework assignment you may collaborate with at most one other student who is currently taking CSCC73. If you collaborate with another student on an assignment, you and your partner must submit only one copy of your solution, with both of your names. The solution will be graded in the usual way and both partners will receive the same mark. Collaboration involving more than two students is not allowed. **For help with your homework you may consult only the instructor, TAs, your homework partner (if you have one), your textbook, and your class notes. You may not consult any other source.**"

Recall that a dynamic programming algorithm to solve a given problem P involves the following elements:
(a) *A definition of a polynomial number of subproblems that will be solved (and from whose solution we will compute the solution to P — see (c) below).*
(b) *A recursive formula to compute the solution to each subproblem from the solutions to smaller subproblems. This induces a partial order on the subproblems defined in (a).*
(c) *A way to compute the solution to P from the solutions to the subproblems computed in (b).*

Proving the correctness of a dynamic programming algorithm amounts to justifying (i) why the recursive formula in step (b) correctly computes the subproblems defined in step (a), and (ii) why the computation in (c) yields a solution to the given problem. Part (ii) is often immediate from the definition of the subproblems.

Question 1. (20 marks) Let $a_1\, b_1\, a_2\, b_2\, a_3\, \ldots\, a_{n-1}\, b_{n-1}\, a_n$ be a sequence where each a_i, $1 \le i \le n$, is an integer, and each b_i, $1 \le i \le n-1$, is either $+$ or $-$. Thus, this sequence represents an unparenthesized

arithmetic expression such as $3-5-7+9$. The actual value of such an expression depends on the order in which the operations $+$ and $-$ are applied, i.e., on how the expression is parenthesized. For instance the above example can be parenthesized in five ways (check them out!), yielding four different values.

Give a ***polynomial-time dynamic programming algorithm*** that finds the ***largest possible value*** of a given unparenthesized expression, over all possible orders of performing the operations — i.e., over all possible parenthesizations. For instance in the above example your algorithm should return 14. More precisely, your algorithm should solve the following problem:

INPUT: Sequences $a_1, \ldots, a_n$ and $b_1, \ldots, b_{n-1}$, where a_i is a positive integer for each $1 \leq i \leq n$ and $b_i \in \{+, -\}$ for each $1 \leq i \leq n-1$.

OUTPUT: The maximum value that can be achieved by parenthesizing the expression $a_1 b_1 a_2 \ldots a_{n-1} b_{n-1} a_n$.

Justify the correctness of your algorithm and analyze its running time.

Hint. A parenthesization of the expression can be viewed as a tree, where the leaves are the a_i's (integers) and the internal nodes are the b_i's ($+$ or $-$).

Question 2. (20 marks) [Honour among thieves.] Two thieves break into a store and steal n items, labeled 1 to n. Item i has value $v_i \in \mathbb{N}$. The two thieves want to divide up the stolen goods so that each of them receives exactly half of the items and these items account for exactly half of the total value of the stolen goods $V = \sum_{i=1}^{n} v_i$. Give a dynamic programming algorithm that takes as input the sequence $v_1, \ldots, v_n$, and determines if such a division is possible. The running time of your algorithm should be polynomial in n and V.

Note: Such an algorithm is not a polynomial-time one; make sure you understand why. If you do come up with a polynomial-time algorithm for this problem, don't keep it secret: You may have just won the one-million dollar (US) Clay Institute prize for settling the P vs. NP question! (Those of you who have taken CSCC63 should be able to see an easy polynomial-time reduction of the Partition problem to this problem.)

Question 3. (15 marks) Let $G = (V, E)$ be a directed graph and $\mathbf{wt} : E \to \mathbb{Z}^+$ be an edge-weight function (note that edges have positive weights). The ***width*** of an $i \to j$ path in G is the ***minimum*** weight of any edge on that path; $+\infty$ if $i = j$ (i.e., there is no edge on that path; and 0 if there is no $i \to j$ path). If we think of the weight of an edge as its width, the width of a path is the width of the narrowest edge, i.e., the "bottleneck" of the path.

Modify the Floyd-Warshall algorithm to obtain an $O(n^3)$ algorithm that, given G and $\mathbf{wt}$, returns a pair (W, pre) of two-dimensional arrays, where $W[u, v]$ is the maximum width of a $u \to v$ path (0 if no such path exists) and $pre[u, v]$ is the predecessor of v on a maximum width $u \to v$ path ($\perp$ if no such path exists). Justify the correctness of your algorithm. (You don't need to analyze its running time, but it should be $O(n^3)$.)