CMPSC 465 Fall 2024 | Data Structures & Algorithms Mingfu Shao and Debarati Das | HW 11

## Due November 15th (Friday), 11:59 pm

Formatting: Each problem should begin on a new page. When submitting in Gradescope, try to assign pages to problems from the rubric as much as you can. Make sure you write all your group members' names. For the full policy on assignments, consult the syllabus.

1. (15 pts.)
Apply the greedy algorithm for Horn formulas (covered in the lecture) to find the variable assignment that solves the following horn formulas:

   1. $(w \wedge y \wedge z) \Rightarrow x, (x \wedge z) \Rightarrow w, x \Rightarrow y, \Rightarrow x, (x \wedge y) \Rightarrow w, (\bar{w} \vee \bar{x} \vee \bar{y}), (\bar{z})$
   2. $(x \wedge z) \Rightarrow y, z \Rightarrow w, (y \wedge z) \Rightarrow x, \Rightarrow z, (\bar{z} \vee \bar{x}), (\bar{w} \vee \bar{y} \vee \bar{z})$

   You need to describe each step of running the algorithm, i.e., the current assignment, which variable will be changed because of what, etc.

   Solution:

   1. 
      - Set everything to false initially.
      - $x$ must be true since we have the statement $\Rightarrow x$.
      - $y$ must be true since $x \Rightarrow y$.
      - $w$ must be true since $(x \wedge y) \Rightarrow w$.
      - Not all negative clauses are satisfied at this point, so there is no satisfying assignment.
   2. 
      - $z$ must be true since we have the statement $\Rightarrow z$.
      - $w$ must be true since $z \Rightarrow w$.
      - $x$ and $y$ need not be changed, as all our implications are satisfied.
      - All negative clauses are now satisfied, so we have found our satisfying assignment.

2. (15 pts.)
Given two strings $x = x_1x_2x_n$ and $y = y_1y_2y_m$, we wish to find the length of their longest common substring, that is, the largest $k$ for which there are indices $i$ and $j$ with $x_ix_{i+1} \ldots x_{i+k-1} = y_jy_{j+1} \ldots y_{j+k-1}$. Show how to do this in time $O(mn)$. Write the subproblem definition, recurrence relation, base case with explanation, and analyze the running time.

   Solution:

   Let $s[i][j]$ be the length of the longest common substring that ends with matching $x_i$ and $y_j$. This is either 0, or, if $x_i = x_j$ , then we can add $x_i$ to the longest common substring that ends with matching $x_{i-1}$ and $y_{j-1}$. The recurrence is:

$$s[i][j] = \begin{cases} 0 & \text{if } i = 0,\ j = 0 \text{ or } x_i \neq y_j; \\ 1 + s[i-1][j-1] & \text{if } x_i = x_j. \end{cases}$$

This can be implemented to run in $O(mn)$ time by filling the table in either row by row or column by column. To recover the longest common substring, we scan the table once to find the maximum value $s[i][j]$. The longest common substring is then $x_{i-s[i][j]-1} \dots x_i$ (equivalently, $y_{j-s[i][j]-1} \dots y_j$) or $\emptyset$ if the maximum is 0. Backtracking also takes $O(mn)$ time.

3. (20 pts.)
A contiguous subsequence of a list $S$ is a subsequence made up of consecutive elements of $S$. For instance, if $S$ is $1, 5, -10, 11, 20, -5$ then $5, -10$, is a contiguous subsequence but $5, 11, 20$ is not. Give a linear-time algorithm for the following task:

Input: A list of numbers $S = a_1, a_2, ..., a_n$.

Output: The contiguous subsequence of maximum sum (a subsequence of length zero has sum zero). For the preceding example, the answer would be $11, 20$ with a sum of 31. Write the subproblem definition, recurrence relation, base case with explanation, and analyze the running time.

Solution:

Let $S[j]$ be the sum of the maximum contiguous sequence that ends at position $j$. Then for $S[j+1]$ we can either start a new contiguous sequence with score $a_{j+1}$ or continue the contiguous sequence with score $S[j] + a_{j+1}$. The recurrence is:

$$S[j] = \begin{cases} a_j & \text{if } j = 0; \\ \max\{S[j-1] + a_j, a_j\} & \text{if } j > 0. \end{cases}$$

We can implement this recurrence in linear time by computing $S[0], S[1], \dots, S[n]$. To recover the sequence, we first scan through $S[]$ to find the location $S[j]$ which is maximum. We scan backwards from $j$ to find the negative element $s[i]$ at the beginning of the maximum contiguous sequence, or have $i = -1$ (we could also directly store these values during the search). Then the maximum contiguous sequence is $a_{i+1}, a_{i+2}, ...a_j$ with score $S[j]$. The backtracking takes linear time as well.