"CS 374" Fall 2015 — Homework 5

Due Tuesday, October 13, 2015 at 10am

••• Some important course policies •••

- **You may work in groups of up to three people.** However, each problem should be submitted by exactly one person, and the beginning of the homework should clearly state the names and NetIDs of each person contributing.
- **You may use any source at your disposal**—paper, electronic, or human—but you ***must*** cite ***every*** source that you use. See the academic integrity policies on the course web site for more details.
- **Submit your pdf solutions in Moodle.** See instructions on the course website and submit a separate pdf for each problem. Ideally, your solutions should be typeset in LaTex. If you hand write your homework make sure that the pdf scan is easy to read. Illegible scans will receive no points.
- **Avoid the Three Deadly Sins!** There are a few dangerous writing (and thinking) habits that will trigger an automatic zero on any homework or exam problem. Yes, we are completely serious.
  - Give complete solutions, not just examples.
  - Declare all your variables.
  - Never use weak induction.
- Unlike previous editions of this and other theory courses we are not using the "I don't know" policy.

See the course web site for more information.

If you have any questions about these policies,
please don't hesitate to ask in class, in office hours, or on Piazza.

---

1. Let $w \in \Sigma^*$ be a string. We say that $u_1, u_2, \ldots, u_h$ where each $u_i \in \Sigma^*$ is a valid split of w iff $w = u_1 u_2 \ldots u_h$ (the concatenation of $u_1, u_2, \ldots, u_h$). Given a valid split $u_1, u_2, \ldots, u_h$ of w we define its ℓ_2 measure as $\sum_{i=1}^{h} |u_i|^2$.

   Given a language $L \subseteq \Sigma^*$ a string $w \in L^*$ iff there is a valid split $u_1, u_2, \ldots, u_h$ of w such that each $u_i \in L$; we call such a split an L-valid split of w. We saw in lecture an efficient algorithm that given w and access to a language L via a subroutine IsStringInL(x) (the subroutine outputs whether the input string x is in L or not) decides whether there is an L-valid split of w. To evaluate the running time of your solution you can assume that each call to IsStringInL() takes constant time.

   Describe an efficient algorithm that given a string w and access to a language L via IsStringInL(x) outputs an L-valid split of w with minimum ℓ_2 measure if one exists.

2. Given a sequence $s := a_1, a_2, \ldots, a_m$ of numbers, let $\text{sum}(s) = \sum_{i=1}^{m} a_i$ denote the sum of the numbers in the sequence. Consider the following problem. The input consists of a sequence $s := a_1, a_2, \ldots, a_n$ of n numbers and an integer k such that $1 \leq k \leq n$. The goal is to split the sequence s into k contiguous non-empty subsequences $s_1, s_2, \ldots, s_k$ such that $\max_{j=1}^{k} \text{sum}(s_i)$ is minimized. A contiguous non-empty subsequence of s is a subsequence of the form $a_i, a_{i+1}, \ldots, a_j$ for some $i \leq j$. A split of s into k subsequences means that every number of s is in one of the k subsequences. For example if $s := 5, 15, -30, 10, -5, 40, 10$ and $k = 2$, one possible split would be into $s_1 := 5, 15, -30$ and $s_2 := 10, -5, 40, 10$ with $\max\{\text{sum}(s_1), \text{sum}(s_2)\} = \max\{-10, 55\} = 55$. The optimum split is $s_1 := 5, 15$ and $s_2 := -30, 10, -5, 40, 10$ with the maximum of the sums being 25. If $k = 3$ the split into $s_1 := 5, 15$, $s_2 := -30, 10, -5, 40$ and $s_3 = 10$ has the maximum of the sums equal to 20; in this case this is optimal.

   Describe an algorithm that given a sequence s stored in an array $A[1..n]$ and an integer k such that $1 \leq k \leq n$ finds a split of s into k contiguous subsequences $s_1, s_2, \ldots, s_k$ that minimizes the quantity $\max_{i=1}^{k} \text{sum}(s_i)$.

   Avoid the temptation to use a greedy algorithm.

3. Suppose you are given a DFA $M = (Q, \Sigma, \delta, s, F)$ and a binary string $w \in \Sigma^*$ where $\Sigma = \{0, 1\}$. Describe and analyze an algorithm that computes the longest subsequence of w that is accepted by M, or correctly reports that M does not accept any subsequence of w.

Rubric (for all dynamic programming problems): As usual, a score of x on the following 10-point scale corresponds to a score of $\lceil n/2 \rceil$ on the 5-point homework scale.

- 6 points for a correct recurrence, described either using mathematical notation or as pseudocode for a recursive algorithm.
  - + 1 point for a clear **English** description of the function you are trying to evaluate. (Otherwise, we don't even know what you're *trying* to do.) **Automatic zero if the English description is missing.**
  - + 1 point for stating how to call your function to get the final answer.
  - + 1 point for base case(s). −½ for one *minor* bug, like a typo or an off-by-one error.
  - + 3 points for recursive case(s). −1 for each *minor* bug, like a typo or an off-by-one error. **No credit for the rest of the problem if the recursive case(s) are incorrect.**
- 4 points for details of the dynamic programming algorithm
  - + 1 point for describing the memoization data structure
  - + 2 points for describing a correct evaluation order. If you use nested loops, be sure to specify the nesting order.
  - + 1 point for time analysis
- It is *not* necessary to state a space bound.
- For problems that ask for an algorithm that computes an optimal *structure*—such as a subset, partition, subsequence, or tree—an algorithm that computes only the *value* or *cost* of the optimal structure is sufficient for full credit.
- Official solutions usually include pseudocode for the final iterative dynamic programming algorithm, ***but this is not required for full credit***. If your solution includes iterative pseudocode, you do not need to separately describe the recurrence, data structure, or evaluation order. (But you still need to describe the underlying recursive function in English.)
- Official solutions will provide target time bounds. Algorithms that are faster than this target are worth more points; slower algorithms are worth fewer points, typically by 2 or 3 points for each factor of n. Partial credit is scaled to the new maximum score, and all points above 10 are recorded as extra credit.

  We rarely include these target time bounds in the actual questions, because when we do include them, significantly more students turn in algorithms that meet the target time bound but don't work (earning 0/10) instead of correct algorithms that are slower than the target time bound (earning 8/10).