“CS 374” Spring 2015 — Homework 6

Due Tuesday, March 17, 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. Suppose we want to run Dijkstra's algorithm on a graph with edge lengths that are integers in the range $0, 1, \ldots, L$. We wish to explore different tradeoffs in the running time that are achievable when L is small but not too small.

   (a) Show how to implement Dijkstra's algorithm to run in time $O(nL + m)$ where n and m are the number of nodes and edges in G. Note that a reduction to BFS by expanding each edge as we discussed in lecture would take $O(n + mL)$ time.

   (b) Show how to implement Dijkstra's algorithm to run in time $O((n + m)\log L)$ time.

   You only need to prove why your implementation correctly follows the steps of Dijkstra's algorithm. *Hint:* Use appropriate data structures to take advantage of the edge length property.

2. Recall that a *palindrome* is any string that is exactly the same as its reversal, like `I`, or `DEED`, or `RACECAR`, or `AMANAPLANACATACANALPANAMA`.

   Any string can be decomposed into a sequence of palindrome substrings. For example, the string `BUBBASEESABANANA` ("Bubba sees a banana.") can be broken into palindromes in the following ways (among many others):

   `BUB` • `BASEESAB` • `ANANA`
   `B` • `U` • `BB` • `A` • `SEES` • `ABA` • `NAN` • `A`
   `B` • `U` • `BB` • `A` • `SEES` • `A` • `B` • `ANANA`
   `B` • `U` • `B` • `B` • `A` • `S` • `E` • `E` • `S` • `A` • `B` • `ANA` • `N` • `A`

   Describe and analyze an efficient algorithm to find the smallest number of palindromes that make up a given input string. For example, given the input string `BUBBASEESABANANA`, your algorithm would return the integer 3.

3. You have a group of investor friends who are looking at n consecutive days of a given stock at some point in the past. The days are numbered. $i = 1, 2, \ldots, n$. For each day i, they have a price $p(i)$ per share for the stock on that day.

   For certain (possibly large) values of k, they want to study what they call *k-shot strategies*. A k-shot strategy is a collection of m pairs of days $(b_1, s_1), \ldots, (b_m, s_m)$, where $0 \le m \le k$ and

   $$1 \le b_1 < s_1 < b_2 < s_2 \cdots < b_m < s_m \le n.$$

   We view these as a set of up to k nonoverlapping intervals, during each of which the investors buy 1,000 shares of the stock (on day b_i) and then sell it (on day s_i). The *return* of a given k-shot strategy is simply the profit obtained from the m buy-sell transactions, namely,

   $$1000 \cdot \sum_{i=1}^{m} (p(s_i) - p(b_i)).$$

   Design an efficient algorithm that determines, given the sequence of prices, the k-shot strategy with the maximum possible return. Since k may be relatively large, your running time should be polynomial in both n and k.

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).