“CS 374” Spring 2015 — Homework 7

Due Tuesday, March 31, 2015 at 10am

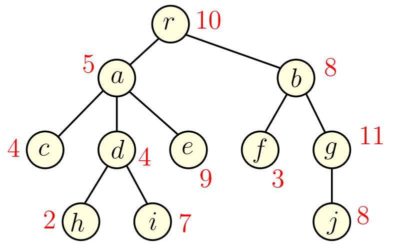
• • • Some important course policies • • •

- **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.

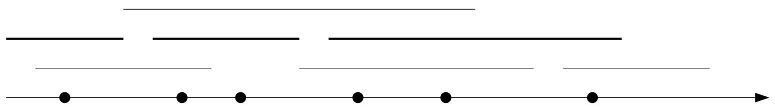
See the course web site for more information.

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

1. Graphs are used to model many types of networks including telecommunication networks. We consider the following link monitoring problem. Given $G = (V, E)$ we want to place monitors on a subset $S \subseteq V$ of nodes so that all the links/edges can be observed by these monitors. If a monitor is placed on a node v it can observe all the edges incident to v. Naturally we wish to minimize the number of monitors. We will consider an even more general version where each node v has a weight $w(v) \geq 0$ and the cost of placing a monitor at v is $w(v)$. The goal is to find a minimum weight subset of nodes that can monitor all the edges. Describe an efficient algorithm for this problem when the given graph is a tree $T = (V, E)$. A subset $S \subset V$ is called a feasible solution if every edge is incident to some node in S. In the tree shown in the figure below $\{a, d, b, j\}$ is a feasible solution. Another feasible solution is $\{r, c, e, h, i, f, g\}$. You can assume without loss of generality that the given tree is rooted at some arbitrary node as shown in the figure.



2. You are given n points $p_1, p_2, \ldots, p_n$ on the real line. The location of p_i is given by its coordinate x_i. You are also given m intervals $I_1, I_2, \ldots, I_m$ where $I_j = [a_j, b_j]$ (a_j is the left end point and b_j is the right end point). Each interval j has a non-negative weight w_j. An interval I_j is said to *cover* p_i if $x_i \in [a_j, b_j]$. A subset $S \subseteq \{I_1, I_2, \ldots, I_m\}$ of intervals is a cover for the given points if for each p_i, $1 \leq i \leq n$, there is some interval in S that covers p_i. In the figure below, the intervals shown in bold form a cover of the points.



The goal is to find a minimum weight cover of the points. Note that a minimum weight cover may differ from a cover with minimum number of intervals.

- Give an example to demonstrate that the greedy algorithm that iteratively picks the interval with minimum ratio of weight to number of remaining uncovered points covered (this changes as algorithm picks intervals) does not yield an optimum solution.
- Describe a dynamic programming-based algorithm that computes an optimum solution.

3. Consider the same problem as the previous one. Now each interval has the same weight and hence the goal is to find the minimum number of intervals that cover the points. Describe a greedy algorithm that solves this problem. Don't forget to prove its correctness.

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).