Homework 2: Greedy Algorithms

Handed out Thu, Sep 28. Due Friday, Oct 6, 11:59pm (electronic submission through ELMS.)

Problem 1. This problem involves an analysis of Huffman's algorithm in the special case where probabilities are all powers of 2.

(a) Show the result of running Huffman's algorithm on the 9-character alphabet shown below.

Symbol	Prob.	Symbol	Prob.	Symbol	Prob.
a	1/16	d	1/8	g	1/32
b	1/8	e	1/2	h	1/16
c	1/32	f	1/32	i	1/32

As part of your answer, you should indicate the following things:

- Show the final tree, and label each leaf node with its symbol and its codeword. (For example, see Fig. 2 from Lecture 6. You need only give the final tree, not the intermediate results.)
- Give the cost $B(T)$ for your tree, that is, the expected number of bits per codeword. Letting $d_T(x)$ denote the depth of symbol x in the tree (where the root is at depth 0), this is given by

$$B(T) \;=\; \sum_{x \in X} p(x) d_T(x).$$

(There was an error in this formula in the lecture notes. It is now fixed.)

Note: For the sake uniformity and to make the grader's life simpler, whenever you have a choice because two or more symbols that have equal probabilities, select the symbols that are lowest in alphabetic order or the tree that has the lowest symbol in alphabetic order.

(b) Looking at your result to (a), what is the relationship of the depth $d(x)$ of each symbol x (or equivalently the number of bits in its codeword) as a function of its probability $p(x)$?

(c) State your result for part (b) as a theorem, and prove it formally. For example "Given an alphabet X such that, for each $x \in X$, the associated probability $p(x)$ is a power of 2, the depth of x in the Huffman tree of X is ...." (Hint: Use induction on $n = |X|$. You may assume the easily proven fact that if all the probabilities are powers of 2, then the two lowest probabilities are equal to one another.)

Problem 2. You are given a collection of files $\{f_1, \ldots, f_n\}$ files that are to be stored on a tape. File f_i is s_i bytes long. The tape is long enough to store all the files. The probability of accessing file f_i is p_i, where $0 \le p_i \le 1$, and $\sum_{i=1}^{n} p_i = 1$. The tape is rewound before each access, and so the time to access any file is proportional to the distance from the front of the tape to the end of the file.

A *layout* of files on the tape is given by a permutation $\pi = \langle \pi_1, \ldots, \pi_n \rangle$ of the numbers $\{1, \ldots, n\}$. (For example, Fig. 1 shows the layout $(4, 2, 1, 3)$.) Given a layout π, the *expected cost* of accessing the ith file on the tape is the product of its access probability and the distance from the start of the tape to the end of the file. The *total cost* of a layout π is the sum of the expected costs for all the files, denoted $T(\pi)$.

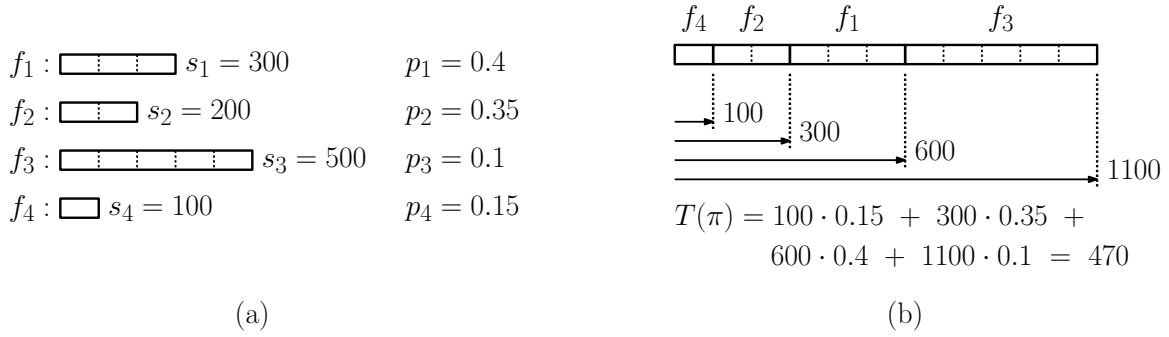

Figure 1: Placing files on a tape to minimize access time.

(a) Present a (short) counterexample so show that laying out the files on the tape in increasing order of size (s_i) is not optimal.

(b) Present a (short) counterexample so show that laying out the files on the tape in decreasing order of access probability (p_i) is not optimal.

(c) Present an algorithm, which given s_i's and p_i's, determines a layout π of minimum total cost. Prove your algorithm's correctness and derive its running time. (Hint: Use a greedy approach.)

Problem 3. You are given a collection I of n intervals $[a_i, b_i]$, for $1 \leq i \leq n$, where $a_i \leq b_i$ (see Fig. 2(a)). A *stabbing set* is a collection of points $X = \{x_1, \ldots, x_k\}$ such that for every interval $[a_i, b_i]$, there exists a point $x_j \in X$ that lies within this interval, that is, $a_i \leq x_j \leq b_i$ (see Fig. 2(b)). In this problem, we will consider the question of how to efficiently compute a stabbing set of the minimum size for a given set of intervals.

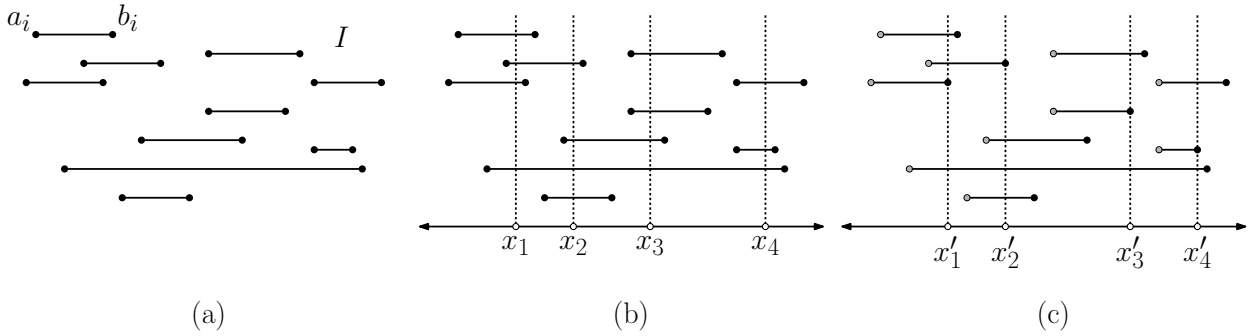

Figure 2: Minimum stabbing set problem.

(a) Show that for any set of intervals, there exists a stabbing set of minimum size where every stabbing point is the right endpoint (b_i) of some interval (see Fig. 2(c)). (Hint: Show how to convert any stabbing set that does not satisfy this property to one that has the equal or lower cardinality and does satisfy this property.) From now on, we will just consider stabbing sets of this form.

(b) Given a set of intervals and any point x, define depth(x) to be number of intervals that contains the point x. Present an $O(n \log n)$ time algorithm, which given a set of intervals, computes a right endpoint (b_i) of maximum depth. (Since there may be multiple endpoints with the same depth, your algorithm should return the leftmost such point. (For example, in Fig. 2(c), both x'_1 and x'_2 have the same depth of four, and there is one other right endpoint with this same depth. The answer should be x'_1.)

Consider the following *max-depth* (MD) greedy heuristic for computing a minimum stabbing set for a given set of intervals I. Apply the algorithm from (b) to compute the right endpoint of maximum depth. Add this endpoint to the stabbing set. Remove from I all the intervals that contain this point. Recompute the depths of the remaining intervals. Repeat until no intervals remain. Clearly, this produces a valid stabbing set, but is it optimal?

(c) What is the worst-case asymptotic running time of this algorithm? (Express your answer in terms of n, the number of intervals, and k, the number of elements in the final stabbing set.)

(d) Present an (ideally short) counterexample to show that the MD heuristic is not optimal. (Hint: There is a simple counterexample involving a small number of intervals if you assume that the whenever there are right endpoints of equal depth, the algorithm chooses the worst one. See Challenge Problem 1.)

(e) Prove that the MD heuristic produces a stabbing set whose size is larger than the optimum by a factor of at most $\ln n$. (Hint: Show that the stabbing set problem can be recast as an instance of the set-cover problem and that the MD heuristic is equivalent to running the greedy set-cover heuristic on this set-cover instance.)

Problem 4. Given the same stabbing-set problem above, consider a new heuristic called *leftmost right endpoint* (LRE). Sort the intervals in increasing order of their right endpoints. Take the first interval from this list, and add its right endpoint to the stabbing set. Remove all intervals that contain this point. Repeat until no intervals remain. Clearly, this produces a valid stabbing set, but is it optimal?

(a) A naive implementation of the LRE heuristic would be rather slow (since it would require traversing the entire list of intervals each time a new element is added to the stabbing set). Present a more efficient implementation that produces the same output, but runs in $O(n \log n)$ time, irrespective of the number of elements in the final stabbing set.

(b) Prove that LRE is optimal, that is, it produces a stabbing set of minimum size.

Challenge Problem. Repeat Problem 3(d), but show that you can construct a counterexample where the MD heuristic fails to return an optimum stabbing set *no matter how depth ties are broken.*