بسم الله الرحمن الرحیم

درس آنالیز الگوریتم - بهار ۱۳۹۴

تمرین سری دوم

تمرین‌های اصلی

6.53 The sequence comparison problem can be generalized to three (or more) sequences in the following way. In each step, we are allowed to insert, delete, or replace characters from any of the sequences. The cost of a step is 0 if the corresponding characters in all sequences are equal, and 1 otherwise (even if two sequences match and only one insertion or deletion is necessary). For example, suppose that the sequences are *aabb*, *bbb*, and *cbb*. One possible edit sequence is inserting *a* in front of *bbb* and *cbb* (which costs 1), replacing a *b* in *bbb* and a *c* in *cbb* with an *a*, and then the rest matches; the total cost is 2. Design an $O(n^3)$ algorithm to find the minimal edit distance between three given sequences.

6.40 Given a set of integers $S = \{x_1, x_2, ..., x_n\}$, find a nonempty subset $R \subseteq S$, such that

$$\sum_{x_i \in R} x_i \equiv 0 \text{ (modulo } n\text{)}.$$

6.23 Given two sets S_1 and S_2, and a real number x, find whether there exists an element from S_1 and an element from S_2 whose sum is exactly x. The algorithm should run in time $O(n \log n)$, where n is the total number elements in both sets.

6.35 The *weighted selection problem* is the following. The input is a sequence of distinct numbers $x_1, x_2, ..., x_n$ such that each number x_i has a positive weight $w(x_i)$ associated with it. Let W be the sum of all weights. The problem is to find, given a value X, $0 < X \leq W$, the number x_j such that

$$\sum_{x_i > x_j} w(x_i) < X,$$

and

$$w(x_j) + \sum_{x_i > x_j} w(x_i) \geq X.$$

Design an efficient algorithm to solve the weighted selection problem. (Notice that when all weights are 1, this problem becomes the regular selection problem.)

6.34 The input is a heap of size n (in which the largest element is on top), given as an array, and a real number x. Design an algorithm to determine whether the kth largest element in the heap is less than or equal to x. The worst-case running time of your algorithm should be $O(k)$, independent of the size of the heap. You can use $O(k)$ space. (Notice that you do not have to find the kth largest element; you need only determine its relationship to x.)

تمرین امتیازی:

*6.64 Let $r_1, r_2, ..., r_n$ and $c_1, c_2, ..., c_n$ be two sequences of integers whose sum is equal; namely,

$$\sum_{i=1}^{n} r_i = \sum_{i=1}^{n} c_i.$$

Such sequences are called **realizable** if there is an $n \times n$ matrix all of whose elements are either 0 or 1, such that, for all i, the sum of the ith row is exactly r_i and the sum of the ith column is exactly c_i. Not all sequences are realizable. For example, the two sequences 0,2 and 0,2 are not realizable since only the second element of the second row can be nonzero, but it cannot be more than 1. Design an algorithm to determine whether two given sequences are realizable, and construct a matrix with the corresponding row and column sums if they are. (Hint: First, strengthen the induction hypothesis to extend the problem to $n \times m$ matrices. Then, use induction on n (the number of rows). Try to place 1s in the first row so that the problem for the other $n-1$ rows can be solved if and only if the original problem can be solved.)

تمرین‌های اضافی (برای مطالعه و پیشرفت)

6.37 Consider the problem of finding the kth largest element, and suppose that we are interested only in minimizing space. Each element fills one memory cell. The input is a sequence of elements, given one at a time, inserted into a fixed cell C. That is, in the ith input step x_i is put into C (and C's previous content is erased). You can perform any computation between two input steps (including, of course, moving the content of C to a temporary location). The purpose is to minimize the extra number of cells required by the algorithm. Give an upper bound and a lower bound on the number of memory cells needed to find the kth largest element.

6.21 The input is a set S with n real numbers. Design an $O(n)$ time algorithm to find a number that is *not* in the set. Prove that $\Omega(n)$ is a lower bound on the number of steps required to solve this problem.

6.24 Design an algorithm to determine whether two sets are disjoint. State the complexity of your algorithm in terms of the sizes m and n of the given sets. Make sure to consider the case where m is substantially smaller than n.

6.25 Design an algorithm to compute the union of two given sets, both of size $O(n)$. The sets are given as arrays of elements. The output should be an array of distinct elements that form the union of the sets. No element should appear more than once. The worst-case running time of the algorithm should be $O(n \log n)$.