Comprehensive Exam: Analysis of Algorithms
Autumn 1991

Problem 1: (20 points total) *Counting*

A group of n people comes to a party, each person carrying a hat and an umbrella. At the end of the party each person leaves with a hat and an umbrella, neither of which is his. Notice that in order to find out the number of possibilities, we can check all possible assignments of hats and umbrellas and count the number of appropriate assignments. The problem with this approach is that it takes exponential time. Can you come up with an approach that will allow us to compute the number of possibilities in polynomial time ? For example, an expression like $\sum_{i=1}^{n} i^3$ is an acceptable answer. In other words, your answer should be a formula computable in polynomial time.

Answer: First, observe that there is no connection between umbrellas and hats, and therefore it is sufficient to count the number of possibilities to assign, say, hats, and square the result. Notice that although the first person to leave has exactly $n-1$ choices, the second person has either $n-1$ or $n-2$ choices, depending on whether the first one has grabbed the hat that belongs to the second one or not. Therefore, this approach will lead to an exponential algorithm.

There are several valid approaches, and we will present two of them. First, notice that we are looking for the number of permutations of n elements that do not have fixed points. Let $H(n)$ denote this number. Total number of permutations on n elements is $n!$. Out of those, we have $H(n)$ permutations with no fixed points, $\binom{n}{1}H(n-1)$ permutations with a single fixed point, $\binom{n}{2}H(n-2)$ with exactly 2 fixed points, etc. Hence, we have:

$$n! = \sum_{i=0}^{n} \binom{n}{i} H(i)$$

Using this formula, one can compute $H(1)$, use it to compute $H(2)$, etc.

A different, somewhat cleaner way, is to use inclusion-exclusion. Let S_i denote the set of permutations that fix i. We would like to compute

$\cup_i |S_i|$. In order to use inclusion-exclusion, we have to be able to compute the cardinality of intersection of r different sets S_i. Given r places that we want to fix, the cardinality of such intersection is $(n-r)!$, since all the rest of the elements can be assigned arbitrary. There are $\binom{n}{r}$ possible intersections associated with each r, and hence this leads to the following formula:

$$H(n) = n! - \sum_{i=1}^{n} (-1)^{i+1} \binom{n}{r} (n-r)!$$

Problem 2: (20 points total)

Given a sequence of numbers $a_1, a_2, \ldots, a_n$, a *subsequence* is a sequence $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$, where $i_j < i_{j+1}$ for all $1 \leq j \leq k-1$. You are given weights $w(a_i) \geq 0$ associated with each element of the given sequence. Describe an efficient algorithm to find an increasing subsequence of maximum weight, where the weight of the subsequence $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$ is defined as $\sum_{j=1}^{k} w(a_{i_j})$. What is the running time of your algorithm? Explain.

Answer: There was a confusion as to what does an "increasing subsequence" mean. It means that the numbers and not weights are increasing. If one tries to solve the problem where we require increasing weights in the output subsequence, then it is easy to see that this problem is a special case of the original problem (just set each number in the sequence to be equal to its weight). Therefore, we will present the solution to the original problem.

The idea is to use dynamic programming. Instead of presenting an algorithm to compute the optimum subsequence, we will concentrate on an algorithm to compute the weight of the subsequence. It is straightforward to change this algorithm to compute the subsequence itself. The following algorithm can be imroved, but we will ommit the optimization issues and concentrate on the main ideas. Notice that, without loss of generality, we can assume that the input sequence includes only numbers from 1 to n.

Let $S_i = a_1, a_2, \ldots, a_i$ be the i-th prefix of the given sequence. Assume that we have computed the maximum-weight increasing subsequence

for S_i. Notice that this is not sufficient in order to quickly compute the maximum-weight increasing subsequence for S_{i+1} ! The reason is that it might be beneficial to use a non-optimum subsequence with respect to S_i in order to be able to use a_i as well.

For each prefix S_i, we will keep $A(i,j)$, $1 \leq j \leq n$, where $A(i,j)$ is the weight of the maximum-weight increasing subsequence out of prefix S_i, where the restriction is that the subsequence does not include elements that are larger than j.

Initialization of $A(1,j)$ is trivial. Now, assume that we have computed $A(i,j)$ for some i and for all j. Consider $k = a_{i+1}$. For all $j < k$, existence of this element does not help us, and we just copy $A(i+1,j) = A(i,j)$. Consider $A(i+1,k)$. We can add a_{i+1} to the corresponding sequence, and hence $A(i+1,k) = A(i,k) + w(a_{i+1})$. For all $j > k$, we set $A(i+1,j) = max\{A(i,j), A(i+1,k)\}$.

The running time of this algorithm is $O(n^2)$ and it uses $O(n)$ space, since we need to keep only a single (current) column of A in memory.

Problem 3: (20 points total)

An ordered 2-3 tree T is used to implement a dictionary, with each element in the dictionary being assigned to a unique leaf in T. Initially, T is empty. Then the sequence of operations INSERT(a_1), INSERT(a_2), ..., INSERT(a_n) is performed, where each of the $n!$ possible orderings of the elements $a_1, a_2, \ldots, a_n$ is equally likely. Let h be the height of T following these n insertions (recall that a tree consisting of a single vertex has height 0).

1. (15 points)
   Give an exact formula for the maximum possible value of h (as a function of n).
2. (5 points)
   If $n = 30$, what is the expected value of h? [*Hint:* What are the minimum and maximum possible values of h?]

Answer: A 2-3 tree is a tree in which each internal node has either 2 or 3 children, and every path from the root to a leaf is of the same length.

When an ordered 2-3 tree is used to implement a dictionary, the elements in the dictionary are stored in the leaves of the tree in ascending order from left to right.

1. $\lfloor \log_2 n \rfloor$.
   If the elements are inserted in increasing order, then following 2^k insertions T is a complete binary tree with height k. Therefore, $h \geq \lfloor \log_2 n \rfloor$. Furthermore, no 2-3 tree with n leaves has height greater than $\lfloor \log_2 n \rfloor$ because all leaves are at the same level and each internal node has degree at least 2, so $h \leq \lfloor \log_2 n \rfloor$.
2. 4.
   Any 2-3 tree of height h has at least 2^h leaves and at most 3^h leaves. If $h \leq 3$ then $3^h < 30$, while if $h \geq 5$ then $2^h > 30$. Therefore, any 2-3 tree with 30 leaves has height 4.

Problem 4: (20 points total) *Recurrence Relations*

Given any constant c, where $0 < c < 1$, and any positive real number N, the function $T(N, c)$ is defined as follows:

$$T(N,c) = \begin{cases} 2T(cN,c) + N^2 & \text{if } N > 1 \\ N^2 & \text{if } 0 < N \leq 1 \end{cases}$$

1. (10 points)
   What is the asymptotic complexity of $T(N, 1/2)$ (to within a constant factor)?
2. (10 points)
   What is the largest value of c such that $T(N,c) = O(N^2 \log N)$?

Answer: 1. $O(N^2)$.

The values of the recursive calls form a geometrically decreasing sequence. For example, if $N > 4$ then

$$\begin{aligned} T(N,1/2) &= 2T(N/2,1/2) + N^2 \\ &= 4T(N/4,1/2) + N^2/2 + N^2 \\ &= 8T(N/8,1/2) + N^2/4 + N^2/2 + N^2. \end{aligned}$$