高等演算法 第 5 次作業

1. Suppose we wish not only to increment a counter but also to reset it to zero (i.e., make all bits in it 0). Counting the time to examine or modify a bit as $O(1)$, show how to implement a counter as an array of bits so that any sequence of n INCREMENT and RESET operations takes time $O(n)$ on an initially zero counter. (Hint: Keep a pointer to the high-order 1.)

2. Consider an ordinary binary min-heap data structure with n elements supporting the instructions INSERT and EXTRACT-MIN in $O(\lg n)$ worst-case time. Give a potential function Φ such that the amortized cost of INSERT is $O(\lg n)$ and the amortized cost of EXTRACT-MIN is $O(1)$ and show that it works.

3. Design a data structure to support the following two operations for a dynamic multi-set S of integers, which allows duplicate values:
   INSERT(S, x) inserts x into S.
   DELETE-LARGER-HALF(S) deletes the largest $\lceil |S|/2 \rceil$ elements from S.
   Explain how to implement this data structure so that any sequence of m INSERT and DELETE-LARGER-HALF operations runs in $O(m)$ time. Your implementation should also include a way to output the elements of S in $O(|S|)$ time.

4. Suppose that in the dynamic table operations, instead of contracting a table by halving its size when its load factor drops below 1/4, we contract it by multiplying its size by 2/3 when its load factor drops below 1/3. What is the amortized cost of a TABLE-DELETE that uses this strategy?

5. Study the famous KMP algorithm for the string matching problem and give an amortized analysis of the algorithm.

6. Given an array $a[1..n]$, for each position in the array, search among the previous positions for the last position that contains a smaller value. That is, for each position i, $1 \le i \le n$, find the largest possible index p_i such that $p_i < i$ and $a[p_i] < a[i]$. For convenience, we attach a dummy element $a[0] = -\infty$. Then, each p_i, $1 \le i \le n$, is well-defined. Design a linear time algorithm to solve this problem.

7. Binary search of a sorted array takes logarithmic search time, but the time to insert a new element is linear in the size of the array. We can improve the time for insertion by keeping several sorted arrays. Specifically, suppose that we wish to support SEARCH and INSERT on a set of n elements. Let $k = \lceil \lg(n+1) \rceil$, and let the binary representation of n be $\langle n_{k-1}, n_{k-2}, \ldots, n_0 \rangle$. We have k sorted arrays $A_0, A_1, \ldots, A_{k-1}$, where for $i = 0, 1, \ldots, k-1$, the length of array A_i is 2^i. Each array is either full or empty, depending on whether $n_i = 1$ or $n_i = 0$, respectively. The total number of elements held in all k arrays is therefore exactly n. Although each individual array is sorted, elements in different arrays bear no particular relationship to each other.
   a) Describe how to perform the SEARCH operation for this data structure. Analyze its worst-case running time.
   b) Describe how to perform the INSERT operation. Analyze its worst-case and amortized running times.
   c) Discuss how to implement DELETE.