Assignment 2 – Selected Model Answers

Due Date: March 4, 2022, 23:59

Exercise 1. Suppose you have to process n advance bookings of rooms for a hotel with k identical rooms. Bookings contain an arrival date and a departure date. You have to find out whether there are enough rooms in the hotel to satisfy the demands.

Design an algorithm that solves this problem in time $O(n \log n)$.

Solution. Let a booking be a pair (a, d) with an arrival date a and a departure date d. We can map dates to $\mathbb{N}$, so without loss of generality assume $a, d \in \mathbb{N}$ with $a < d$. We can then view the i'th booking as a pair comprising $(a_i, 1)$ (for arrival) and $(d_i, 0)$ for departure. The set of all advance bookings is given by the list $[(a_1, 1), (d_1, 0), (a_2, 1), (d_2, 0), \dots, (a_n, 1), (d_n, 0)]$.

We can sort this list using the order $(x_1, y_1) \leq (x_2, y_2)$ iff $x_1 < x_2$ or $(x_1 = x_2$ and $y_1 \leq y_2)$. As the components are natural numbers, the most appropriate sorting algorithm is radix sort.

If we now have a sorted list, we can iterate over it using a counter H, which is initialised as 0. A list element $(x, 1)$ increments H, while a list element $(x, 0)$ decrements H. After each change of H we have to check, if $H \leq k$ holds. If true, we can continue. If false, the capacity of the hotel has been exceeded.

Exercise 2. It is easy to check to check whether an algorithm produces a sorted output. It is less easy to check whether the output is also a permutation of the input. However, for integers there exists a fast and simple algorithm:

(i) Show that $[e_1, \dots, e_{n_i}]$ is a permutation of $[e'_1, \dots, e'_{n_i}]$ iff the polynomial

$$P(x) \;=\; \prod_{i=1}^{n}(x - e_i) - \prod_{i=1}^{n}(x - e'_i)$$

in the variable x is identically zero.

(ii) For any $\varepsilon > 0$ let p be a prime with $p > \max\{n/\varepsilon, e_1, \dots, e_{n_i}, e'_1, \dots, e'_{n_i}\}$. The idea is to evaluate the above polynomial $P(x)$ modulo p for a random value $x \in [0, p-1]$.

Show that if $[e_1, \dots, e_{n_i}]$ is not a permutation of $[e'_1, \dots, e'_{n_i}]$, then the result of the evaluation is zero with probability at most ε.

Solution.

(i) Clearly, if $[e_1, \dots, e_n]$ is a permutation of $[e'_1, \dots, e'_n]$, the factors in both products are the same, so the polynomial is identical zero.

Conversely, both products are polynomials of degree n with roots $\{e_1, \dots, e_n\}$ and $\{e'_1, \dots, e'_n\}$. If $P(x)$ is identically zero, these two polynomials are the same, and hence have the same roots with the samne multiplicities, which in other words means that $[e_1, \dots, e_n]$ is a permutation of $[e'_1, \dots, e'_n]$.

(ii) $P(x)$ is a polynomial $x^n + a_{n-1}x^{n-1} + \cdots + a_0$ of degree n, and as such has at most n roots, unless it is identically zero. As p is a prime and all e_i, e'_i are in $[0, p-1]$, the same holds modulo p.

Taking random values over the interval $[0, p-1]$ means we have p possible values for x, and the outcome $P(x) = 0$ can appear at most n times. So the probability that $P(x) = 0$, i.e. the probability that x is one of the at most n roots, is at most $\frac{n}{p} < \varepsilon$.

EXERCISE 3. Show how to reverse the order of elements on a stack S

(i) using two additional stacks;

(ii) using one additional queue;

(iii) using one additional stack and some additional non-array variables.

SOLUTION.

(i) Let S, S_1, S_2 be the three stacks, and assume that initially $a_1, \dots, a_k$ are stored in S from bottom to top, while S_1 and S_2 are empty.

With $\texttt{isempty}_S()$ we can check, if the stack S is empty or not. If it is non-empty, then with $\texttt{top}_S()$ we retrieve the top element a of the stack S, so a follow-on $\texttt{pop}_S()$ together with a $\texttt{push}_{S_1}(a)$ moves the element a to stack S_1. This can be iterated k-times until the stack S becomes empty and $a_k, \dots, a_1$ are stored in S_1 from bottom to top.

We do the same with S_1 and S_2 until S_1 becomes empty and $a_1, \dots, a_k$ are stored in S_2 from bottom to top. Finally, do the same again for S_2 and S, which results in an empty stack S_2 and $a_1, \dots, a_k$ in S (from bottom to top), i.e. the order of elements in S is reversed.

(ii) Let S be a stack and Q a FIFO queue. Assume that initially $a_1, \dots, a_k$ are stored in S (from bottom to top), while Q is empty.

With $\texttt{isempty}_S()$ we can check, if the stack S is empty or not. If it is non-empty, then with $\texttt{top}_S()$ we retrieve the top element a of the stack S, so a follow-on $\texttt{pop}_S()$ together with a $\texttt{pushback}_Q(a)$ moves the element a to the queue Q. This can be iterated k-times until the stack S becomes empty and $a_1, \dots, a_k$ are stored in Q from front to back.

As long as Q is non-empty, we can retrive the font-element a with $\texttt{popfront}_Q()$, and $\texttt{push}_S(a)$ moves the element a to stack S. This can be iterated k-times until the queue Q becomes empty. Then S contains $a_1, \dots, a_k$ (from bottom to top), i.e. the order of elements in S is reversed.

(iii) Let S, S' be the two stacks, and assume that initially $a_1, \dots, a_k$ are stored in S from bottom to top, while S' is empty. In addition use a counter `count`, in which we store a non-negative integer (initially 0), and a variable `item` for a single value of type T (the type of the elements in the stack).

In an initial round we move a_k to `item` and shift one-by-one all other stack elements onto S' (as in (i)) thereby incrementing `count` to $k-1$, so S' contains $a_{k-1}, \dots, a_1$ are stored in S from bottom to top.

In following rounds (provided `count` $= \ell \neq 0$) first push the value a in `item` onto the stack S, the move one-by-one all ℓ elements in S' onto the stack S. Then shift the top-element from S to `item`, decrement `count` and move again $\ell - 1$ elements from S to S'. After the i'th of these rounds we will have $a_k, \dots a_{k-i+1}$ on the stack S, a_{k-i} in `item`, $a_{k-i-1}, \dots, a_1$ stored in S^prime, and the value of `count` will be $k - i - 1$. So after k such rounds the order of elements in S is reversed.

Exercise 4. Implement bucket sort on sequences represented by unbounded arrays: The input is a sequence of real numbers x in the range $a \leq x < b$. For a sequence of length n split $[a, b)$ into n equi-distant buckets B_i $(0 \leq i \leq n-1)$, i.e. intervals $B_i = [a_i, a_{i+1})$ with $a_i = a + i \cdot \frac{b-a}{n}$. Use insertion sort to insert each element x of the input sequence into a list ℓ_i, provided $x \in B_i$ holds, then concatenate the lists $\ell_0, \dots, \ell_{n-1}$ to obtain the sorted output list.

total points: 20

Solution. See the C++ header and program files in `Ass2_Ex4solution.zip`.