University of Freiburg
Dept. of Computer Science
Prof. Dr. F. Kuhn
M. Ahmadi, P. Schneider

Algorithms Theory
Exercise Sheet 3

Due: Monday, 3rd of December, 2018, 14:15 pm

Exercise 1: Packaging Marbles *(4+4 Points)*

We are given n marbles and have access to an (arbitrary) supply of packages. We are also given an array $A[1..n]$, where entry $A[i]$ equals the value of a package containing exactly i marbles. Our profit is the total value of all packages containing at least one marble, minus the cost of packaging, which is i for a package containing i marbles. We want to maximize our profit.

(a) Give an efficient algorithm that uses the principle of dynamic programming to package marbles for a maximum profit.

(b) Argue why your algorithm is correct. Give a tight (asymptotic) upper bound for the running time of your algorithm and prove that it is an upper bound for your solution.

Exercise 2: Breaking Eggs[1] *(1+5+5+5 Points)*

Imagine a building with n floors. Additionally, we are given a supply of k eggs. For some reason we need to find out at which floor eggs start breaking when dropped from a window on that floor.

Suppose that dropping an egg from a certain floor always produces the same result, regardless of which egg is used and any other conditions. Initially, we do not have any knowledge at which height eggs might break. If an egg does not break, then it does not take any harm and can be fully reused.

When eggs break when dropped from a floor, they also break when dropped from higher floors. When eggs survive being dropped from a floor, they survive being dropped from lower floors. We call the floor from which dropped eggs break but survive at all floors beneath, the *critical floor*. The goal is to find the critical floor with minimal number of attempts (number of times eggs are being dropped).

(a) Suppose we have only one egg. Give a strategy to always find the critical floor, if it exists.

(b) For $n \gg k$, describe a strategy that finds the critical floor with $\mathrm{O}\left(k \sqrt[k]{n}\right)$ attempts, if it exists.

(c) We want some advance knowledge before starting to drop eggs. For inputs n and k, we want to compute the *exact* number of attempts $a(n, k)$ which an optimal strategy requires *in the worst case*, until it finds the critical floor if it exists. Give an algorithm that uses the principle of dynamic programming to compute $a(n, k)$ in $\mathrm{O}(kn^2)$ time.

(d) Argue the correctness of your algorithm and its running time.

[1] This question has been asked in some of Google's job-interviews.

Exercise 3: Binary Counter *(3+4 Points)*

If we use a counter in standard binary representation, then an arbitrary series of `increment` and `decrement` operations can incur relatively high cost in terms of number of bits that need to be flipped. E.g. decrementing and incrementing the number 2^k in binary representation repeatedly in an alternating fashion incurs $\Theta(k)$ bit flips for each operation.

We introduce a more efficient data structure, where the counter is represented by two binary numbers P and N, and the current state of the counter is $P-N$. The counter is initialized with $P = N = 0$. The `increment` and `decrement` operations are implemented as follows.

Algorithm 1 `increment`(P, N)

$i \leftarrow 0$
while $P_i = 1$ **do** ▷ P_i is the i^{th} bit of P.
  $P_i \leftarrow 0$
  $i \leftarrow i + 1$
if $N_i = 1$ **then** $N_i \leftarrow 0$
else $P_i \leftarrow 1$

Algorithm 2 `decrement`(P, N)

$i \leftarrow 0$
while $N_i = 1$ **do** ▷ N_i is the i^{th} bit of N.
  $N_i \leftarrow 0$
  $i \leftarrow i + 1$
if $P_i = 1$ **then** $P_i \leftarrow 0$
else $N_i \leftarrow 1$

(a) Let $P = 100110$ and $N = 001000$. Give the state of P, N (in binary representation) and $P - N$ (in decimal representation) after each of the following operations: `increment`, `increment`, `increment`, `increment`, `decrement`, `decrement`.

(b) Use the accounting method to show that any series of `increment` and `decrement` operations on a counter initialized with $P = N = 0$ has amortized costs of O(1) (number of bits flipped) per operation.

Exercise 4: Dynamic Array *(2+7 Points)*

In the lecture we saw a dynamically growing array that implements the `append` operation in amortized O(1) (reads/writes). For an array of size N that is already full, the `append` operation allocates a new array of size $2N$ ($\beta = 2$) before inserting an element at the first free array entry.

Additionally, we introduce another operation `remove` which writes `Null` into the last non-empty entry of the dynamic array. However, by appending many elements and subsequently removing most of them, the ratio of unused space can become arbitrarily high. Therefore, when the dynamic array of size N contains $\frac{N}{4}$ or less elements after `remove`, we copy each element into a new array of size $\frac{N}{2}$.

(a) Given an array of size N which has at least $\frac{N}{2}$ elements, show that any series of `remove` operations has an amortized cost of O(1) (reads/writes) per operation.

(b) Use the potential function method to show that any series of `append` and `remove` operations has amortized cost of O(1) (reads/writes) per operation. Assume that the number of elements n in the array is initially 0 and assume that the array never shrinks below its initial size N_0 (we stop allocating smaller arrays in that case).

Remarks: You may assume that N is always a power of two. You may also assume that allocating an empty array is free, only copying elements costs one read and one write for each copied element. If you do part (b) absolutely correctly you automatically receive all points for part (a).