University of Porto LEIC

Algorithms and Data Structures 2024/2025

Exercise Sheet 1

1. Consider the function REMOVE(v, x, n) which deletes all values greater than v from the array x of integers. The final result will be in $x[0], \ldots, x[t-1]$ and the function returns t. In the end, the values in the positions t to $n-1$ of x are not relevant (can be anything). Here, by "positions t to $n-1$" we mean the remaining ones, if there are any. As mentioned in the class, in this pseudocode, $\leftarrow$ stands for the assigning operation.

```
REMOVE(v, x, n)
1.  t ← 0
2.  p ← 0;
3.  while (p < n) do
4.      if x[p] ≤ v then
5.          x[t] ← x[p];
6.          t ← t + 1;
7.      p ← p + 1;
8.  return t;
```

a) Let $a_0, a_1, \ldots, a_{n-1}$ be the values of $x[0], \ldots, x[n-1]$ at line 1. Find a loop invariant that we can use to conclude that the function is correct (refer to line 3 to state the invariant). You must describe the state of all relevant variables and complete the sentence: *"When we are testing the condition in line 3 for the kth time, for $k \geq 1$, the value of p is $k-1$ and $p \leq n$, the value of v is ..., the value of t is ..., and the array contains ..."*

b) Using the invariant, justify that the loop **terminates** (check **Progress** and **Termination**).

c) Using the invariant, conclude that, in line 8, the value we return and the content of the array x satisfy the problem specification. We assume that the array x is *passed by reference* and v and n are *passed by value*.

d) Show that the invariant is true, by checking the properties we refer to as **Initialization** and **Maintenance**. Or, equivalently, prove the invariant by mathematical induction.

e) Find the number of times each instruction is executed in the *worst case* and in the *best case*. Define them as a function of n, which is assumed to be fixed (i.e., constant). For lines 3 and 4, count the number of times the condition is checked. The *worst case* occurs when all elements are less than or equal to v and the *best case* occurs when all elements are greater than v.

f) Programming practice (in C++): implement the function REMOVE and a program to test it.

2. Consider FUNC(x, n, v, p), where the parameters x and v are arrays of integers of length n and $p+1$, and $1 \leq x[k] \leq p$, for $0 \leq k < n$.

```
FUNC(x, n, v, p)
1   for k ← 0 to p do v[k] ← 0;
2   s ← x[0];
3   v[s] ← 1;
4   t ← 1;
5   for k ← 1 to n − 1 do
6       v[x[k]] ← v[x[k]] + 1;
7       if s = x[k] ∨ v[x[k]] > v[s] then
8           s ← x[k];
9           t ← 1;
10      else if v[x[k]] = v[s] then t ← t + 1;
11  return (s, t);
```

a) Analyse FUNC(x, n, v, p) and state the problem that it solves. For that, find the meaning of s and t.

b) State an invariant of the loop 5-10 that allows us to conclude that the function is correct for that problem. Explain how the invariant is kept in each iteration and how we conclude that the result (s, t) in line 11 is correct.

c) Programming practice (C++): write a program to test the function.

3. Consider a function that reads a sequence of integers that ends with `0` and returns a pair (v, k), where v is the **the last negative odd number** that was given and k is the **number of odd numbers** in the sequence. If there is no negative odd number, the value of v is `0`. You can assume that each line of the input contains a single integer.

a) Write the function in pseudocode (let $x\%y$ be the remainder of the division of x by y). **Do not use arrays** (i.e., the memory used by the algorithm does not depend on the number of elements given).

b) Prove that the function you wrote is correct. For the proof, define a (useful) loop invariant and prove it.

c) Find the number of times each instruction is executed in the worst case and in the best case (define them as functions of the number n of input values). Characterize the best case and worst case instances.

d) Programming practice (in C++): implement the function and a program to test it.

4. Any integer greater than or equal to 2 is prime or product of primes, being the factorization unique (except for the ordering of factors). For example, $368 = 2 \times 2 \times 2 \times 2 \times 23$. We want a program to compute the factorization of n.

a) Using C++, implement a function that given an integer n, such that $2 \leq n \leq 1000000$, and an array with the existing prime numbers between 2 and 1000, sorted in ascending order, as well as its length, outputs the factorization of n into primes, ordered in the same way. Recall that:

- the factorization can be found by **successive divisions**, starting from 2;
- if any prime p greater than $\sqrt{n}$ occurs in the factorization of n, then p occurs only once;
- if p_1 and p_2 are two prime numbers greater than $\sqrt{n}$ then only one of them can divide n.

For $n = 124992$ the output must be: `2*2*2*2*2*2*3*3*7*31`

For $n = 19$, the output is `19`

For $n = 807200$, the output is `2*2*2*2*2*5*5*1009`

Note that `1009` is not in the array of prime numbers we give, but $\sqrt{1009} < 11$ and therefore, if no prime less than or equal to $\sqrt{1009}$ divides `1009`, then `1009` is a prime number.

In order to **be efficient**, your function must use the properties of prime numbers stated above.

b) Justify the **correctness** of your function.

5. Suppose that the *array* x contains in the positions 1 to $3k - 2$ a sequence of integers

$$\boxed{v_1}\ p_1\ d_1\ \boxed{v_2}\ p_2\ d_2\ \boxed{v_3}\ \ \ldots\ \boxed{v_{k-1}}\ p_{k-1}\ d_{k-1}\ \boxed{v_k},$$

that represents information about a tour of a vehicle, for pick-up and delivery of a product. In that sequence, the v_i's are locations and **all distinct**, $p_i \in \{0, 1\}$ indicates whether the link (v_i, v_{i+1}) has problems or not that can delay the services (being 1 if it has), d_i is the maximum number of units that the vehicle can carry in that link, for all i.

We want to send a minimum of a and a maximum of b units from a location s to a location t, being $1 \leq a \leq b$. Can the vehicle pick up and deliver them? If it can, which is the maximum number of units that we can send and which is the number of links with problems from v_1 to t?

In order to solve the problem, we have implemented a function ANALISE(x, k, s, t, a, b), that must return $(0, 0)$ iff it is not possible to use that vehicle. Otherwise, it must return a pair with the values we asked for.

```
ANALISE(x, k, s, t, a, b)
1  | p ← 0;
2  | j ← 1;
3  | while k > 1 ∧ x[j] ≠ s ∧ x[j] ≠ t do
4  |     j ← j + 3;
5  |     p ← p + x[j − 2];
6  |     k ← k − 1;
7  | if x[j] ≠ s then return (0, 0);
8  | c ← b;
9  | k ← k − 1;
10 | while x[j] ≠ t ∧ c ≥ a ∧ k > 0 do
11 |     j ← j + 3;
12 |     p ← p + x[j − 2];
13 |     if x[j − 1] < c then c ← x[j − 1];
14 |     k ← k − 1;
15 | if x[j] ≠ t ∨ c < a then return (0, 0);
16 | return (c, p);
```

Remark that, the position 0 of the array is not used! The problem statement says "Suppose that the *array* x contains in the positions 1 to $3k - 2 \ldots$". We can assume that, $k \geq 2$ initially, although the function returns $(0, 0)$ if $k = 1$.

a) Let k_0 be the value of k in line 1 and assume that the content of the array x is as stated above. Show the following **invariant for loop 3-6** (justify the properties that we refer to as **Initialization** and **Maintenance**):

> *For $i \geq 1$, when we are testing the condition in line 3 for the ith time, $j = 3(i-1)+1 = 3i-2$, $p = \sum_{r=1}^{i-1} p_r$ (that is, the number of links with problems from v_1 to $v_i = x[j]$), s and t did not occur in $v_1 \ldots v_{i-1}$, $k = k_0 - (i-1) \geq 1$, and it remains to analyse $k-1$ links, which correspond to the positions of x starting at the one with index j, which contain* $\boxed{v_i}\ p_i\ d_i\ \boxed{v_{i+1}}\ \ldots\ \boxed{v_{k_0-1}}\ p_{k_0-1}\ d_{k_0-1}\ \boxed{v_{k_0}}$.

b) Complete the analysis of the **correctness** of the function. For that purpose:

- write the **invariant of loop 10-14** and justify concisely how it is kept in each iteration;
- check and explain whether the values returned in lines 7, 15 and 16 are correct.