Algorithmic Language: Exercises

Ştefan Ciobâcă, Dorel Lucanu
Faculty of Computer Science
Alexandru Ioan Cuza University, Iaşi, Romania

PA 2021/2022

1 Introducere

The aim of the seminar is to get acquainted with the Alk language, to write simple algorithms in Alk and to test them with the Alk interpreter.

Install the Alk interpreter and follow the installation instructions at

`https://github.com/alk-language/java-semantics/`

New! Also install the Visual Studio Code extension for Alk: `https://github.com/Andreizabo/alk-vscode-extension#readme` From now on, we consider that the Alk interpreter can be called using the symbolic name `alki` (created, for example, with a symbolic "link").

2 Solved exercises

Exerciţiul 1 Try the Alk interpreter on the examples in the `examples/miscelanea` folder.

Design an algorithm that computes the first n prime numbers for a given n.

Soluţie. Please note the definition of prime numbers [1]

> A *prime number* (or prime integer, often simply called a "prime" for short) is a positive integer $p > 1$ that has no positive integer divisors other than 1 and p itself. More concisely, a prime number p is a positive integer having exactly one positive divisor other than 1, meaning it is a number that cannot be factored. For example, the only divisors of 13 are 1 and 13, making 13 a prime number, while the number 24 has divisors $1, 2, 3, 4, 6, 8, 12$, and 24 (corresponding to the factorization $24 = 2^3 \cdot 3$), making 24 not a prime number. Positive integers other than 1 which are not prime are called *composite numbers.*

[1]Source: `http://mathworld.wolfram.com/PrimeNumber.html`

Based on this definition, we can build our first tests:

n	first n prime numbers
0	
1	2
2	2, 3
3	2, 3, 5
4	2, 3, 5, 7

Data structures used: The n prime numbers will be stored in a list.
We will write a function after the following template (schematic):

```
firstNPrimes(n) {
  // compute first n primes and save them in list l
  return l;
}
```

The above template can be refined by constructing an item-by-item list:

```
firstNPrimes(n) {
  l = emptyList;
  while ( l.size() < n) {
    // compute next prime x
    l.pushBack(x);
  }
  return l;
}
```

The computation of the next prime number can be done by scrolling through the natural numbers > 1 and testing them for primality:

```
firstNPrimes(n) {
  x = 2;
  l = emptyList;
  while ( l.size() < n) {
    if (isPrime(x)) {
      l.pushBack(x);
    }
    ++ x;
  }
  return l;
}
```

The `isPrime(x)` algorithm can be written using an iterative method that tests if a positive integer $n \neq 1, n$ which divides x; if so, then the algorithm returns $false$, otherwise $true$. Positive integers $n \neq 1, n$ that could divide x are $2, 3, \ldots, x/2$. As a result, the `isSprime(x)` algorithm can be written simply like in the following:

```
isPrime(x) {
  if (x < 2) return false;
  for (i= 2; i <= x / 2; ++i)
    if (x % i == 0) return false;
  return true;
}
```

Now we write the two algorithms, together with the test instruction, in a text file. `prime.alk`:

```
isPrime(x) {
  if (x < 2) return false;
  for (i= 2; i <= x / 2; ++i)
    if (x % i == 0) return false;
  return true;
}

firstNPrimes(n) {
  x = 2;
  l = emptyList;
  while ( l.size() < n) {
    if (isPrime(x)) {
      l.pushBack(x);
    }
    ++ x;
  }
  return l;
}

print(firstNPrimes(6));
```

2.1 Execution of algorithms with the Alk interpreter

The algorithm can be tested by the following command line:

```
$ alki -a prime.alk
<2, 3, 5, 7, 11, 13>
```

sfsol

2.2 Initializing Input Variables from Command Line

If instead of `print(firstNPrimes(6));` you type `print(firstNPrimes(n));` and run the above command, you get an error:

```
$alki -a prime.alk
Error at line 25: The reference is invalid: n
```

The reason is that the value of `n` is not known in its original state. The value of `n` in the initial state can be specified with the `-i` option:

```
$ alki -a prime.alk -i "n|->7"
<2, 3, 5, 7, 11, 13, 17>
n |-> 7
```

2.3 Reading of input variables from a file

Another possibility is to specify the initial state in a file, say `prime.in`, and to specify the name of this file as an argument to the `-i` option:

```
alki.bat -a prime.alk -i prime.in
<2, 3, 5, 7, 11>
n |-> 5
```

3 Proposed exercises

Exerciţiul 2 To get acquainted with the ALK language, follow the examples: `https://github.com/alk-language/java-semantics/wiki/Alk-by-examples`

Exerciţiul 3 Write an algorithm that generates the primes with Eratosthenes sieve (`http://mathworld.wolfram.com/SieveofEratosthenes.html`)

Exerciţiul 4 Each new term in the Fibonacci sequence is generated by adding the two previous terms. Starting with 1 and 2, the first 10 terms will be:

$$1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...$$

Considering the terms in the Fibonacci sequence whose values do not exceed four million, find the sum of the even terms.

Exercițiul 5 Consider the following implementation of the DFS algorithm.

```
/* This example includes the recursive version of the DFS algorithm.
    @input: a digraf D and a vertex i0
    @output: the list S of the verices reachable from i0 */

dfsRec(out D, i, out S) {   // the recursive function
  if (S[i] == 0) {
    // visit i
    S[i] = 1;
    p = D.a[i];
    while (p.size() > 0) {
      j = p.topFront();
      p.popFront();
      dfsRec(D, j, S);
} } }

dfs(out D, i0) {   // the calling algorithm
  i = 0;
  while (i < D.n) {
    S[i] = 0;
    i = i + 1;
  }
  dfsRec(D, i0, S);
  return S;
}
reached = dfs(D, i0);
print(reached);

/* Example of running command line
   Assuming that the digraph D is given by
     D.n = 3
     D.a[0] = <1, 2>
     D.a[1] = <2, 0>
     D.a[2] = <0>
   and the vertex i0 is 1, then create a file dfs.in with the following contents:
D |-> { n -> 3
        a -> [ < 1, 2 >, < 2, 0 >,  < 0 > ] }
i0 |-> 1

and execute the command
alki -a dfsrec.alk -i dfs.in
should print "[1, 1, 1]" */
```

(a) Implement the iterative version of the DFS algorithm starting from the example above.

(b) Change the implementation to make it BFS.

(c) Run the implemented algorithms on a few examples with larger graphs.

[RO] - exerciții din anii anteriori:

1. Scrieți trei algoritmi (Euclid prin scăderi repetate, Euclid prin împărțiri repetate, căutare explicită) pentru următoarea problemă:

   `Input`: două numere naturele $a, b \in \mathbb{N}$ cu proprietatea că $a + b \neq 0$;

   `Output`: cel mai mare divizor comun al numerelor a, b.

2. Dați exemple de câteva instanțe ale problemei de mai sus și de răspunsurile aferente.

3. Problema de mai sus este aceeași cu problema ce urmează?

   `Input`: două numere naturale $a, b \in \mathbb{N}$;

   `Output`: "da" și $cmmdc(a, b)$, dacă a și b au un cel mai mare divizor comun și "nu" altfel.

4. Enunțați problema celor 8 regine. Dați exemplu de instanța a problemei.

5. Enuntați problema satisfiabilității pentru formule din `LP` ca pereche `Input-Output`.

6. Enunțați problema satisfiabilității pentru formule din `LP1` ca pereche `Input-Output`.

7. Gândiți-vă la trei algoritmi pe care i-ați învățat la liceu și enunțați ce probleme rezolva algoritmii.

8. Care dintre problemele discutate la acest seminar sunt probleme de decizie? Care sunt probleme rezolvabile?

9. Formalizați următoarea problemă și scrieți un algoritm pentru ea: să se scrie un natural n ca sumă de numere naturale consecutive.

10. Scrieți în Alk algoritmul căutării binare, care rezolvă problema:

    `Input:` a - număr întreg și v - un vector de n întregi, ordonat crescător.

    `Output` o poziție i pe care se găsește a în v sau -1, dacă nu se găsește.

    Testați algoritmul pe câteva exemple.