Limbajul Algoritmic: Exerciţii

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

PA 2019/2020

1 Introducere

Scopul seminarului este de familiarizare cu limbajul Alk, scrierea de algorimi simpli în Alk şi de testare a acestora cu intepretorul Alk.

Instalați intepretorul Alk (versiunea 1.1) de la adresa
`https://github.com/alk-language/java-semantics/releases/tag/v1.1`

1. Descărcați arhiva `alki-v1.1.zip`.
2. Dezarhivați `alki-v1.1.zip` într-un director în care aveți pachetele de software; să zicem că acest director este `<alki>`.
3. Pe Windows: adăugați calea `<alki>\alki-v1.1\v1.1\Windows` la variabila sistem PATH;
   Pe Linux: adăugați calea `<alki>/alki-v1.1/v1.1/Linux_Mac` la variabila sistem PATH;
4. Testați interpretorul dând comanda Windows `alki.bat -h` sau comanda Linux `alki.sh`. Sitemul ar trebui să afișeze un mesaj de forma:

```
Missing required option: a
usage: utility-name
 -a,--alk <arg>    algorithm file path
 -i,--init <arg>   initial configuration
```

Observație. În următoarele săptămâni va fi lansată o nouă versiune de interpretor, cu mai multe funcționalități.

2 Exerciţii rezolvate

Exerciţiul 1 Să se proiecteze un algoritm care determină primele n numere prime pentru un n dat. Ce se poate spune despre timpii de execuţie uniform şi logaritmic?

Soluţie. Să reamintim mai întâi definiţia pentru numere prime[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*.

Pe baza acestei definiţii ne putem construi şi primele teste:

n	primele n numere prime
0	
1	2
2	2, 3
3	2, 3, 5
4	2, 3, 5, 7

Structurile de date utilizate: cele n numere prime vor fi memorate într-o listă. Vom scrie o funcţie după următorul şablon (schemă):

```
firstNPrimes(n) {
  // calculează primele $n$ numere prime şi le memorează în lista  l
  return l;
}
```

Şablonul de mai sus poate fi rafinat prin schema de construcţie a unei liste element cu element:

```
firstNPrimes(n) {
  l = emptyList;
  while ( l.size() < n) {
    // calculează următorul număr prim  x
    l.pushBack(x);
  }
  return l;
}
```

Calcularea următorului număr prim poate fi făcută prin parcurgerea numerelor naturale > 1 şi testarea lor dacă sunt prime:

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

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

Algoritmul `isPrime(x)` poate fi scris printr-o schemă iterativă care testează dacă există un număr întreg pozitiv $n \neq 1, n$ care divide pe x; dacă da, atunci algoritmul întoarce *false*, altfel întoarce *true*. Numerele întregi pozitive $n \neq 1, n$ care ar putea divide pe x sunt $2, 3, \ldots, x/2$. Rezultă că algoritmul `isPrime(x)` poate fi scris simplu după cum urmează:

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

Acum scriem cei doi algoritmi, împreună cu instrcţiunea de testare, într-un fişier `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 Execuția algoritmilor cu interpretorul Alk

Algoritmul poate fi testat prin următoarea linie de comandă:

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

sfsol

2.2 Inițializarea variabilelor de intrare direct din linia de comandă

Dacă în loc de `print(firstNPrimes(6));` scrieți `print(firstNPrimes(n));` și executați comanda de mai sus, obțineți o eroare:

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

Motivul este ca valoarea lui `n` nu este cunoscută în starea inițială. Valoarea lui `n` din starea inițială poate fi precizată cu opțiunea `-i`:

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

2.3 Valorile inițiale ale variabilelor de intrare pot fi citite și dintr-un fișier

O altă posibiltate este precizarea stării inițiale într-un fișier, să zicem `prime.in`, și precizarea numelui acestui fișier ca argument al opțiunii `-i`:

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

3 Exerciţii propuse

Exerciţiul 2 Să se scrie un algoritm care generează prime cu ciurul lui Eratosthenes (`http://mathworld.wolfram.com/SieveofEratosthenes.html`).

Exerciţiul 3 Fiecare nou termen din secvența Fibonacci este generat prin adăugarea celor doi termeni precedenți. Începând cu 1 și 2, primii 10 termeni vor fi:

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

Luând în considerare termenii din secvența Fibonacci ale căror valori nu depășesc patru milioane, găsiți suma termenilor pari.

Exerciţiul 4 Limbajul Alk include şi operaţii peste mulţimi:

```
s1 = { 1 .. 5 };
s2 = { 2, 4, 6, 7 };
a = s1 U s2 ;
print(a);
b = s1 ^ s2;
print(b);
c = s1 \ s2;
print(c);
x = 0;
forall y in s2 x = x + y;
print(x);
d = emptySet;
forall y in { 1 .. 6 }
  if (y in s2) d = d U { y };
print(d);
```

1. Scrieți codul de mai sus într-un fișier, apoi lansați-l în execuție.
2. Comparați informațiile afișate cu codul pentru a înțelege operațiile executate.
3. Ce se poate spune despre timpii de execuţie uniform şi logaritmic ai fiecării operaţii?