Algoritmid ja keerukusteooria

Algoritm on eeskiri teatud probleemi lahendamiseks. Algoritm annab lahenduse üldisele ja täpselt sõnastatud probleemile. Algoritmika probleem kirjeldab sisendandmed, mida algoritm saab kasutada ja omadusi, mis peavad olema väljundil pärast algoritmi täitmist. Siin tuleb teha vahet sisendi ja väljundi üldisel kirjeldusel ja sisendi ja väljundi konkreetsetel väärtustel algoritmi käivitamisel. [1]

Üldine kirjeldus: Sisend on positiivne täisarv n. Väljund on loend kõigist paarisarvudest vahemikus $0…n$.

Näited konkreetsete väärtustega:

Sisend	**Väljund**
n=2	*[0, 2]*
n=5	*[0, 2, 4]*

Algoritmi kirjeldamine

Algoritmi kirjeldamisel koodi tasemest veidi kõrgemal tasemel võib kasutada:

1.) Pseudokoodi – tavakeele ja programmeerimiskeele vahepealne tase.

```
Paaris(n):
See algoritm tagastab kõik paarisarvud vahemikus 0..n
    Initsialiseeri Tulem
    Üle kõigi i-de vahemikus 0… n:
        Kui (i jääk jagamisel kahega) == 0:
            lisa i Tulemisse
    tagasta Tulem
```

2.) Pseudokood on lisatud kommentaaridena tavalise koodi juurde. Pseudokood kirjutatakse enne.

```
def paaris(n):
    """
    Tagastab kõigi paarisarvude loendi vahemikus 0..n
    """
    # Initsialiseeri tühi loend Tulem
    Tulem = []
    # Üle iga i vahemikus 0..n
    for i in range(n+1):
        # Kui i jääk jagamisel kahega on null,
        if (i % 2) == 0:
            # siis lisa i Tulemisse
            Tulem.append(i)
    # Väljasta Tulem
    return Tulem
```

3.) Koodi kõrgtaseme keeles mõnede kommentaaridega

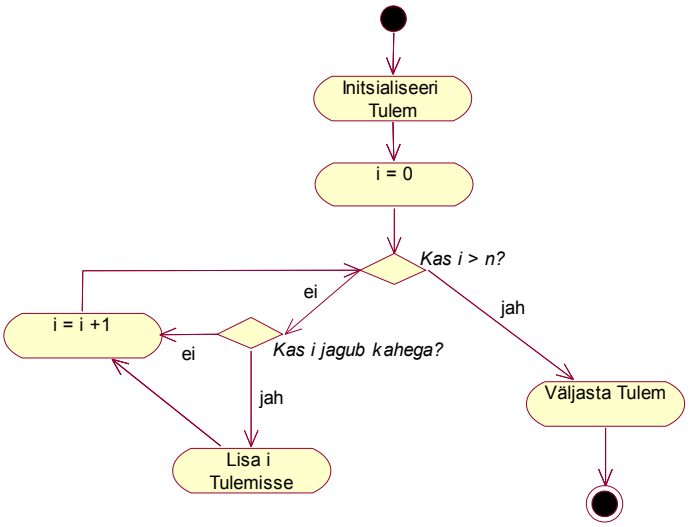
```
def paaris(n):
    """
    Tagastab kõigi paarisarvude loendi vahemikus 0..n
    """
    Tulem = []
    # NB! vahemik 0..n on range(n+1), mitte range(n)
    for i in range(n+1):
        # Kui i jääk jagamisel kahega on null,
        if (i % 2) == 0:
            Tulem.append(i)
     return Tulem
```

Vahel võimaldab kõrgtaseme keel algoritmi asendamist ühe koodireaga ☺

```
def paaris(n):
    """
    Tagastab kõigi paarisarvude loendi vahemikus 0..n
    """
    return [i for i in range(n+1) if (i % 2) == 0]
```

(List Comprehensions http://docs.python.org/2/tutorial/datastructures.html#list-comprehensions)

4.) Diagramme



Algoritmi korrektsus

Algoritm peab olema korrektne st. väljastama kõigi lubatud sisendite korral korrektse väljundi. Tavaliselt on võimalike sisendite arv väga suur või lõpmatu. Sellisel juhul võib testimine konkreetsete näiteandmetega näidata vigade olemasolu, kuid ei tõesta vigade puudumist. Praktikas annab testimine siiski küllaltki olulise kinnituse algoritmi korrektsuse osas, eriti kui testimisel on läbi proovitud ka erand- ja piirjuhud.

Algoritmi korrektsuse näitamiseks võib kasutada ka vähem- või rohkem formaalsemat tõestamist. Näiteks:

Meie algoritm käib läbi kõik täisarvud vahemikus 0..n. Täisarv lisatakse väljundisse siis ja ainult siis kui see on paarisarv. Järelikult peavad väljundis olema kõik paarisarvud vahemikus 0..n ja ei midagi muud.

Algoritmi efektiivsus

Algoritmi efektiivsus võib tähendada aja- või mäluefektiivsust. Efektiivsema algoritmi leidmine võib tähendada hiiglaslikku võitu tööajas või mälukasutuses. Algoritmi keerukust esitatakse tavaliselt funktsioonina sisendandmete suurusest *n*.

n	*f(n) = log n*	*f(n) = n*	*f(n) = n log n*	*f(n) = n^2*	*f(n) = 2^n*	*f(n) = n!*
10	0.003 µs	0.01 µs	0.033 µs	0.1 µs	1 µs	3.63 ms
30	0.005 µs	0.03 µs	0.147 µs	0.9 µs	1 s	$8.4*10^{15}$ aastat
100	0.007 µs	0.1 µs	0.644 µs	10 µs	$4*10^{13}$ aastat	
1000	0.01 µs	1.00 µs	9.966 µs	1 ms		
10000	0.013 µs	10 µs	130 µs	100 ms		

Tabel 1.Kui palju aega võiks kuluda erineva suurusega sisendandmete ja erineva keerukusega algoritmide korral.[1]

Pythoni `cProfile` standardteek

Pythoni *cProfile* standardteek võimaldab mõõta kui palju aega programm mingis funktsioonis veedab ja seega aitab see hinnata algoritmi keerukust ja leida kriitilisi kohti programmis, mis kogu süsteemi aeglaseks teevad.

See on standardteek, selle importimiseks tuleb lihtsalt kirjutada:

```
import cProfile
```

Et näha ajakulu informatsiooni tuleb käivitada mooduli funktsioon *run* ja anda sellele **stringina** ette käivitatav käsk.

```
cProfile.run('anagramm3("suur peet", "pete urus")')
```

Kahte tüüpi jutumärgid (' ja ") võimaldavad käsus ka stringe kasutada.

Väljundtabel näeb välja järgmine:

```
         989519 function calls (883891 primitive calls) in 0.787 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.787    0.787 <string>:1(<module>)
        1    0.000    0.000    0.787    0.787 anagramm.py:38(anagramm3)
 105629/1    0.302    0.000    0.787    0.787 anagramm.py:41(anagramm3rec)
        2    0.000    0.000    0.000    0.000 anagramm.py:9(tyhikuteta)
   211256    0.155    0.000    0.155    0.000 copy.py:117(_copy_with_constructor)
   211256    0.249    0.000    0.434    0.000 copy.py:67(copy)
        1    0.000    0.000    0.787    0.787 {built-in method exec}
   105628    0.012    0.000    0.012    0.000 {method 'append' of 'list' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
   211256    0.031    0.000    0.031    0.000 {method 'get' of 'dict' objects}
    38858    0.010    0.000    0.010    0.000 {method 'join' of 'str' objects}
   105628    0.029    0.000    0.029    0.000 {method 'remove' of 'list' objects}
        2    0.000    0.000    0.000    0.000 {method 'replace' of 'str' objects}
```

Väljundi ridadeks on erinevad funktsioonid ja meetodid, mida programmi töö käigus on välja kutsutud.

Väljundi veegude tähendused:
ncalls: mitu korda funktsiooni välja kutsuti
tottime: aeg, mis funktsioonis veedeti ILMA alamfunktsioonides veedetud ajata
percall: tottime ühe väljakutsete kohta
cumtime: aeg, mis funktsioonis veedeti KOOS alamfunktsioonides veedetud ajaga
percall: cumtime ühe väljakutse kohta

Kasutatud kirjandus

[1] Skiena, S. The Algorithm Design Manual, (1998)