

# Portofoliu Algoritmi și complexitate

Pavel Andrei

## Laborator 1

15. O colecție de  $n$  monede sunt identice, cu excepția uneia, falsă, care are greutatea mai mică decât a celorlalte. Propuneți o strategie pentru a identifica moneda falsă folosind o balanță simplă și cât mai puține cântăriri.

```
1 import random, math
2 # The values aren't important, what matters is that fake < real
3 realWeight = 40
4 fakeWeight = 35
5
6 length = int(input("Please enter length: "))
7
8 coins = [realWeight] * length
9 coins[random.randrange(length)] = fakeWeight
10
11 bigPile = coins
12 steps = 0
13
14 while len(bigPile) > 1:
15     steps += 1
16     print("Step %d:" % steps)
17     #we split the big pile in 3 piles with the same size ±1
18     pileSize = len(bigPile) // 3 + (len(bigPile) % 3 == 2)
19     leftPile = bigPile[0:pileSize]
20     rightPile = bigPile[pileSize:pileSize*2]
21     leftoverPile = bigPile[pileSize*2:]
22     print("left", leftPile)
23     print("right", rightPile)
24     print("leftover", leftoverPile)
25     difference = 0
26     for i in range(pileSize):
27         difference += rightPile[i] - leftPile[i]
28
29     if difference > 0: bigPile = leftPile
30     elif difference < 0: bigPile = rightPile
31     else: bigPile = leftoverPile
32     print()
33
34 print("Took", steps, "steps to find the fake coin, expected",
35       math.ceil(math.log(length, 3)))
```

```
$ python3 src.py
Please enter length: 15
Step 1:
left [40, 40, 40, 40, 40]
right [40, 40, 40, 40, 40]
leftover [40, 40, 35, 40, 40]
```

```
Step 2:
left [40, 40]
right [35, 40]
leftover [40]
```

```
Step 3:
left [35]
right [40]
leftover []
```

Took 3 steps to find the fake coin, expected 3

16. Bucătarul unui restaurant a pregătit clătite și le-a stivuit pe o farfurie. Fiind începător, clătitele nu au ieșit la fel, având diametre diferite, iar farfuria arată destul de rău. Bucătarul șef a vrut să-i dea o lecție și i-a dat sarcina de a rearanja (cea cu diametrul cel mai mare să fie prima pe farfurie, apoi cea cu diametrul imediat următor ca mărime ș.a.m.d.) folosind doar o spatulă. Ce strategie să adopte?

```
1 print("Please insert a space separated list of pancake diameters:\n(",
2     end="")
3 strs = input().split(' ')
4 # index 0 represents the topmost pancake
5 pancakes = [int(num) for num in reversed(strs) if num != ""]
6
7 def draw(msg, vec, spatulaIndex = -1):
8     print(msg + "(", end="")
9     i = len(vec)
10    if spatulaIndex > 0:
11        for i in range(i-1, spatulaIndex-1, -1): print(vec[i], end=" ")
12        print("|", end="")
13
14    for i in range(i-1, -1, -1): print(vec[i], end=" ")
15    print()
16
17 flips = 0
18 def flip(vec, index):
19     global flips
20     flips += 1
21     draw("flip %2d: " % (flips), pancakes, index)
22     for i in range(0, index//2):
23         t = vec[i]
24         vec[i] = vec[index-i-1]
25         vec[index-i-1] = t
26
```

```

27 for bottom in range(len(pancakes), 0, -1):
28     spatulaIndex = 0
29     for i in range(0, bottom):
30         if pancakes[i] >= pancakes[spatulaIndex]: spatulaIndex = i
31     spatulaIndex += 1
32     # if the biggest is at the bottom we do nothing
33     if spatulaIndex == bottom: continue
34     # if the biggest is already at the top we don't have to flip it
35     if spatulaIndex != 1: flip(pancakes, spatulaIndex)
36
37     flip(pancakes, bottom)
38
39 draw("", pancakes)
40 print("Done in %d flips" % flips)

```

```
$ python3 src.py
```

Please insert a space separated list of pancake diameters:

```

(5 9 4 3 7 2 8 1
flip 1: (5 |9 4 3 7 2 8 1
flip 2: (|5 1 8 2 7 3 4 9
flip 3: (9 4 3 7 2 |8 1 5
flip 4: (9 |4 3 7 2 5 1 8
flip 5: (9 8 1 5 2 |7 3 4
flip 6: (9 8 |1 5 2 4 3 7
flip 7: (9 8 7 3 4 2 |5 1
flip 8: (9 8 7 |3 4 2 1 5
flip 9: (9 8 7 5 1 2 |4 3
flip 10: (9 8 7 5 |1 2 3 4
(9 8 7 5 4 3 2 1
Done in 10 flips

```

```
$ python3 src.py
```

Please insert a space separated list of pancake diameters:

```

(4 3 2 1
(4 3 2 1
Done in 0 flips

```

```
$ python3 src.py
```

Please insert a space separated list of pancake diameters:

```

(3 3 1 4 3
flip 1: (3 3 1 |4 3
flip 2: (|3 3 1 3 4
flip 3: (4 3 1 |3 3
flip 4: (4 3 |1 3 3
(4 3 3 3 1
Done in 4 flips

```

## Laborator 2

10. Demonstrați corectitudinea algoritmului de determinare a valorii obținute prin inversarea ordinii cifrelor unui număr natural.

```
1  #include <iostream>
2
3  int main() {
4      unsigned n;
5      std::cout << "Please insert n: ";
6      std::cin >> n;
7
8      unsigned res = 0;
9      int i = 0;
10     while (n != 0) {
11         res *= 10;
12         res += n % 10;
13         n /= 10;
14         ++i;
15     }
16
17     std::cout << res << " I: " << i << "\n";
18     return 0;
19 }
```

### I. Parțial corectitudinea

Considerăm secvențele de intrare și ieșire:

$$P_{in} = \left\{ n = \sum_{j=0}^k c_j 10^j; c_j \in \overline{0,9}, \forall j \in \overline{0,k}; c_k \neq 0 \right\},$$

$$P_{out} = \left\{ res = \sum_{j=0}^k c_{k-j} 10^j \right\}.$$

Alegem proprietatea:

$$I = \left\{ n = \sum_{j=0}^{k-i} c_{i+j} 10^j; res = \sum_{j=0}^{i-1} c_{i-1-j} 10^j \right\}.$$

La intrarea în buclă:

$$i = 0$$

$$n = \sum_{j=0}^k c_j 10^j$$

$$\text{Deci propoziția } I = \left\{ n = \sum_{j=0}^k c_j 10^j; res = \sum_{j=0}^{-1} c_{-1-j} 10^j = 0 \right\} \text{ este adevărată.}$$

Arătăm că propoziția  $I$  este invariantă.

Presupunem  $I$  adevărată la începutul iterației și  $n \neq 0$ ; demonstrăm  $I$  adevărată la sfârșitul iterației.

$$n = \sum_{j=0}^{n-i} c_{i+j} 10^j; res = \sum_{j=0}^{i-1} c_{i-1-j} 10^j$$

9

```
res *= 10;
```

$$res = \left( \sum_{j=0}^{i-1} c_{i-1-j} 10^j \right) \cdot 10 = \sum_{j=0}^{i-1} c_{i-1-j} 10^{j+1} = \sum_{j=1}^i c_{i-j} 10^j$$

10

```
res += n % 10;
```

$$res = \left( \sum_{j=1}^i c_{i-j} 10^j \right) + c_i = \left( \sum_{j=1}^i c_{i-j} 10^j \right) + c_{i-0} 10^0 = \sum_{j=0}^i c_{i-j} 10^j$$

11

```
n /= 10;
```

$$n = \left\lfloor \left( \sum_{j=0}^{k-i} c_{i+j} 10^j \right) / 10 \right\rfloor = \left\lfloor \sum_{j=0}^{k-i} c_{i+j} 10^{j-1} \right\rfloor = \left\lfloor \sum_{j=1}^{k-i} c_{i+j} 10^{j-1} \right\rfloor + \lfloor c_i 10^{-1} \rfloor$$

$$\text{Cum } 0 \leq c_i \leq 9 \implies 0 \leq c_i 10^{-1} \leq 0.9 \implies \lfloor c_i 10^{-1} \rfloor = 0.$$

$$\text{Deci } n = \left\lfloor \sum_{j=1}^{k-i} c_{i+j} 10^{j-1} \right\rfloor = \sum_{j=1}^{k-i} c_{i+j} 10^{j-1} = \sum_{j=0}^{k-i-1} c_{i+j+1} 10^j.$$

12

```
++i;
```

Scriem  $res$  și  $n$  în funcție de noul  $i$ . Deci  $i$  devine  $i - 1$ .

$$res = \sum_{j=0}^{i-1} c_{i-1-j} 10^j$$

$$n = \sum_{j=0}^{k-(i-1)-1} c_{i-1+j+1} 10^j = \sum_{j=0}^{k-i} c_{i+j} 10^j$$

Deci  $I$  adevărată și la sfârșitul iterației.

La ieșirea din buclă:

$$i = k + 1$$

$$n = \sum_{j=0}^{k-(k+1)} c_{k+1+j} 10^j = \sum_{j=0}^{-1} c_{k+1+j} 10^j = 0$$

$$res = \sum_{j=0}^{k+1-1} c_{k+1-1-j} 10^j = \sum_{j=0}^k c_{k-j} 10^j$$

$$\text{Deci } P_{out} = \left\{ res = \sum_{j=0}^k c_{k-j} 10^j \right\} \text{ adevărată.}$$

În concluzie algoritmului este parțial corect.

## II. Total corectitudinea

Considerăm funcția  $t : \mathbb{N} \rightarrow \mathbb{N}$ ,  $t(i) = k+1-i$ ,  $t(i+1) - t(i) = k+1-(i+1) - (k+1-i) = -1 < 0$ , deci  $t$  monoton strict descrescătoare.

$$t(i) = 0 \iff i = k + 1 \iff n = \sum_{j=0}^{-1} c_{k+1+j} 10^j = 0 \iff \text{condiția de ieșire din buclă.}$$

În concluzie algoritmului este total corect.

## Laborator 3

10. Considerăm o secvență  $x = (x_0, \dots, x_{n-1})$  de  $n$  numere întregi, cu măcar un element pozitiv. O subsecvență a șirului este de forma  $(x_i, x_{i+1}, \dots, x_j)$ , cu  $0 \leq i \leq j \leq n-1$ , iar suma subsecvenței este suma elementelor componentelor sale. Descrieți un algoritm pentru a determina subsecvența de sumă maximă. Estimați timpul de execuție al algoritmului, precizând operația dominantă.

```
1 print("Please insert the sequence: ", end="")
2 strs = input().split(' ')
3 v = [int(num) for num in strs if num != ""]
4 n = len(v)
5 # python way of defining a n-dimensional list initialized to 0
6 sub_sums = [0 for i in range(0, n)]
7
8 best = (0, 0)
9 best_sum = 0
10 for i in range(0, n):
11     sub_sums[i] = v[i]
12     best_end_index = i
13     # after this loop v[j] = (sum from k=i to j of v[k])
14     for j in range(i+1, n):
15         sub_sums[j] = sub_sums[j-1] + v[j]
16         if sub_sums[j] > sub_sums[best_end_index]:
17             best_end_index = j
18     if sub_sums[best_end_index] > best_sum:
19         best_sum = sub_sums[best_end_index]
20         best = (i, best_end_index)
21
22 print("Best with a sum of", best_sum, "is: (x%d,...,x%d)" % best)
```

```
$ python3 src.py
Please insert the sequence: 1 2 3 4
Best with a sum of 10 is: (x0,...,x3)
```

```
$ python3 src.py
Please insert the sequence: 1 -2 3 4
Best with a sum of 7 is: (x2,...,x3)
```

```
$ python3 src.py
Please insert the sequence: 1 2 -3 4
Best with a sum of 4 is: (x0,...,x3)
```

Considerăm operația de bază ca fiind compararea elementelor tabloului  $v$  (liniile 16 și 18). Notăm  $T_l(n) :=$  timpul total de execuție al liniei  $l$ ;  $T(n) :=$  timpul de execuție total.

$$T_{16}(n) = \sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-1} ((n-1) - i) = n(n-1) - \sum_{i=0}^{n-1} i = n(n-1) - \frac{n(n-1)}{2} = \frac{n(n-1)}{2}$$

$$T_{18}(n) = \sum_{i=0}^{n-1} 1 = n$$

$$T(n) = \frac{n(n-1)}{2} + n = \frac{n(n+1)}{2}$$

## Laborator 4

8. Considerăm o secvență  $x = (x_0, \dots, x_{n-1})$  de  $n$  numere întregi. Generați tabloul  $f = (f_0, \dots, f_{n-1})$ , cu  $f_i = \sum_{j=0}^i x_j$ , printr-un algoritm de complexitate liniară.

```
1 print("Please insert the sequence: ", end="")
2 strs = input().split(' ')
3 x = [int(num) for num in strs if num != ""]
4 n = len(x)
5 f = [0 for i in range(n)]
6
7 f[0] = x[0]
8 for i in range(1, n):
9     f[i] = f[i-1] + x[i]
10
11 print(f)
```

```
$ python3 src.py
Please insert the sequence: 1 2 3 0 -1 5
[1, 3, 6, 6, 5, 10]
```

9. Considerăm un tablou de valori întregi  $x = (x_0, \dots, x_{n-1})$  și o valoare dată,  $s$ . Să se verifice dacă există cel puțin doi indici  $i$  și  $j$  (nu neapărat distincți) cu proprietatea că  $x_i = x_j = s$ . Analizați complexitatea algoritmului propus.

```
1 print("Please insert the sequence: ", end="")
2 strs = input().split(' ')
3 x = [int(num) for num in strs if str != ""]
4 print("Please insert s: ", end="")
5 s = int(input())
6
7 def f(x):
8     for i in range(0, len(x)):
9         for j in range(i, len(x)):
10             if x[i] + x[j] == s:
11                 print ("Found %d + %d = %d " % (x[i] , x[j], s))
12                 return True
13     print("Not found")
14     return False
15 f(x)
```

```
$ python3 src.py
Please insert the sequence: 1 2 3 0 -1 5
Please insert s: 9
Not found
```

```
$ python3 src.py
Please insert the sequence: 1 2 3 0 5 -1
Please insert s: 7
Found 2 + 5 = 7
```

## Laborator 5

4. (*Shaker sort*) modificând algoritmul de sortare prin interschimbarea elementelor vecine, sortați elementele unui tablou, astfel încât, la fiecare pas, să se plaseze pe pozițiile finale câte două elemente: minimul, respectiv maximum din subtabloul parcurs la pasul respectiv.

```
1 print("Please insert the array: ", end="")
2 strs = input().split(' ')
3 v = [int(num) for num in strs if num != ""]
4
5 def impl(start, end, step):
6     sorted = True
7     for i in range(start, end, step):
8         if v[i] > v[i+1]:
9             t = v[i]
10            v[i] = v[i+1]
11            v[i+1] = t
12            sorted = False
13    return sorted
14
15 begin = 0
16 end = len(v) - 1
17
18 while True:
19     if impl(begin, end, 1): break
20     if impl(end-1, begin-1, -1): break
21
22     end -= 1
23     begin += 1
24
25 print(v)
```

```
$ python3 shaker_sort.py
Please insert the array: 6 5 3 1 8 7 2 4 0 9
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
$ python3 shaker_sort.py
Please insert the array: 4 1 0 2 7 3 9 8 5 6
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
$ python3 shaker_sort.py
Please insert the array: 9 4 3 0 5 6 1 8 7 0
[0, 0, 1, 3, 4, 5, 6, 7, 8, 9]
```

```
$ python3 shaker_sort.py
Please insert the array: 40 3 43 95 9 2 4 0
[0, 2, 3, 4, 9, 40, 43, 95]
```



5. (*Counting sort* - sortare prin numărare) Considerăm un tablou  $x$  de dimensiune  $n$ , cu elemente din mulțimea  $\{0, 1, 2, \dots, m\}$ . Pentru sortarea unui astfel de tablou poate fi descris un algoritm de sortare de complexitate liniară, dacă  $m$  nu este semnificativ mai mare ca  $n$ . Pașii algoritmului sunt:

- (a) se construiește tabloul  $f[0..m]$  al frecvențelor de apariție a elementelor tabloului  $x$  ( $f_i$  reprezintă de câte ori apare valoarea  $i$  în tabloul  $x$ ,  $i = 0, \dots, m$ );
- (b) se calculează tabloul frecvențelor cumulate  $fc[0..m]$ ,  $fc_i = \sum_{j=0}^i f_j$ ,  $i = 0, \dots, m$ ;
- (c) se folosește tabloul frecvențelor cumulate pentru a construi tabloul ordonat.

Descrieți algoritmul de sortare prin numărare. Care este complexitatea acestuia?

```

1 print("Please insert the array: ", end="")
2 x = [int(num) for num in (input().split()) if num != ""]
3 n = len(x)
4 m = max(x) + 1
5
6 f = [0 for i in range(m)]
7 output = [0 for i in range(n)]
8
9 for i in x: f[i] += 1
10 print("f:", f)
11
12 for i in range(1, m): f[i] = f[i-1] + f[i]
13
14 print("fc:", f)
15 for i in range(n):
16     val = x[i]
17     f[val] -= 1
18     output[f[val]] = val
19
20 print("output:", output)

```

```

$ python3 counting_sort.py
Please insert the array: 0 1 2 2 1 0 2 1 2 4
f: [2, 3, 4, 0, 1]
fc: [2, 5, 9, 9, 10]
output: [0, 0, 1, 1, 1, 1, 2, 2, 2, 2, 4]

```

```

$ python3 counting_sort.py
Please insert the array: 1 2 2 1 2 1 2 4
f: [0, 3, 4, 0, 1]
fc: [0, 3, 7, 7, 8]
output: [1, 1, 1, 2, 2, 2, 2, 4]

```

Considerăm atribuirile în vectori ca fiind operațiile de bază (ignorăm inițializările).

Notăm  $T_l :=$  timpul total de execuție al liniei  $l$ ;  $T(n, m) :=$  timpul de execuție total.

$T_9 = n$ ;  $T_{12} = m - 1$ ;  $T_{17} = n$ ;  $T_{18} = n$ ;

$T(n, m) = 3n + m - 1$ .

$T \in O(n + m)$ .

6. (*Radix sort* - sortare pe baza cifrelor) Considerăm un tablou  $x$  de dimensiune  $n$ , cu elemente numere naturale de cel mult  $k$  cifre. Algoritmul de sortare este bazat pe următoarea idee: folosind counting sort, se ordonează tabloul în raport cu cifra cea mai puțin semnificativă a fiecărui număr, apoi se sortează în raport cu cifra de rang imediat superior ș.a.m.d., până de ajunge la cifra cea mai semnificativă.

Descrieți algoritmul radix sort. Care este complexitatea acestuia?

```

1 print("Please insert the array: ", end="")
2 x = [int(num) for num in (input().split()) if num != ""]
3 n = len(x)
4 max_x = max(x)
5
6 f = [0 for i in range(10)]
7 output = [0 for i in range(n)]
8
9 pow10 = 1
10 while max_x > 0:
11     def getDigit(num): return (num // pow10) % 10
12
13     for i in range(10): f[i] = 0
14     for i in x: f[getDigit(i)] += 1
15
16     for i in range(1, 10): f[i] += f[i-1]
17
18     for i in range(n - 1, -1, -1):
19         index = getDigit(x[i])
20         f[index] -= 1
21         output[f[index]] = x[i]
22
23     #output becomes new input
24     for i in range(n): x[i] = output[i]
25
26     pow10 *= 10
27     max_x //= 10
28
29 print("output:", output)

```

```
$ python3 radix_sort.py
```

```
Please insert the array: 3 2 4 23 427 459 56 90
```

```
output: [2, 3, 4, 23, 56, 90, 427, 459]
```

```
$ python3 radix_sort.py
```

```
Please insert the array: 89568 23 123 2 1 4 45 499
```

```
output: [1, 2, 4, 23, 45, 123, 499, 89568]
```

Considerăm atribuirile în vectori ca fiind operațiile de bază (ignorăm inițializările).

Notăm  $k = \log_{10}(\max(x)) + 1$ ;  $T_l :=$  timpul total de execuție al liniei  $l$ ;  $T(n, k) :=$  timpul de execuție total.

$T_{13} = 10k$ ;  $T_{14} = kn$ ;  $T_{16} = 9k$ ;  $T_{20} = kn$ ;  $T_{21} = kn$ ;

$T(n, k) = 3kn + 19k$ .

$T \in O(kn)$ .