Portofoliu Algoritmi și complexitate

Anul I, semestrul 2

Pavel Andrei
Grupa M116

Facultatea de Matematică
Universitatea "Alexandru Ioan Cuza"
Iași
2018 - 2019

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.

```python
import random, math

# The values aren't important, what matters is that fake < real
realWeight = 40
fakeWeight = 35

print("Please enter length: ", end="")
length = int(input())

coins = [realWeight] * length
coins[random.randrange(length)] = fakeWeight

bigPile = coins
steps = 0

while len(bigPile) > 1:
    steps += 1
    print("Step %d:" % steps)

    # we split the big pile in 3 piles with the same size ±1
    pileSize = len(bigPile) // 3 + (len(bigPile) % 3 == 2)
    leftPile = bigPile[0:pileSize]
    rightPile = bigPile[pileSize:pileSize*2]
    leftoverPile = bigPile[pileSize*2:]

    print("left", leftPile)
    print("right", rightPile)
    print("leftover", leftoverPile)

    difference = 0
    for i in range(pileSize):
        difference += rightPile[i] - leftPile[i]

    if difference > 0: bigPile = leftPile
    elif difference < 0: bigPile = rightPile
    else: bigPile = leftoverPile

    print()

print("Took", steps, "steps to find the fake coin, expected",
      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?

```python
print("Please insert a space separated list of pancake diameters:\n(",
     end="")
strs = input().split(' ')
# index 0 represents the topmost pancake
pancakes = [int(num) for num in reversed(strs) if num != ""]

def draw(msg, vec, spatulaIndex = -1):
    print(msg + "(", end="")
    i = len(vec)
    if spatulaIndex > 0:
        for i in range(i-1, spatulaIndex-1, -1): print(vec[i], end=" ")
        print("|", end="")

    for i in range(i-1, -1, -1): print(vec[i], end=" ")
    print()

flips = 0
def flip(vec, index):
    global flips
    flips += 1
    draw("flip %2d: " % (flips), pancakes, index)
    for i in range(0, index // 2):
        t = vec[i]
        vec[i] = vec[index-i-1]
        vec[index-i-1] = t

```

```python
for bottom in range(len(pancakes), 0, -1):
    spatulaIndex = 0
    for i in range(0, bottom):
        if pancakes[i] >= pancakes[spatulaIndex]: spatulaIndex = i
    spatulaIndex += 1
    # if the biggest is at the bottom we do nothing
    if spatulaIndex == bottom: continue
    # if the biggest is already at the top we don't have to flip it
    if spatulaIndex != 1: flip(pancakes, spatulaIndex)

    flip(pancakes, bottom)

draw("", pancakes)
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.

```python
n = int(input("Please insert n: "))

res = 0
i = 0

while n != 0:
    res *= 10
    res += n % 10
    n //= 10
    i += 1

print("Result:", res)
```

```
$ python3 reversed.py
Please insert n: 1234
Result: 4321

$ python3 reversed.py
Please insert n: 2400
Result: 42
```

I. Parțial corectitudinea

Considerăm serțiunile 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 in buclă:

$i = 0$

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

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

Arătăm că propoziția I este invariantă.

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

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

```
    res *= 10
```

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

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

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

```
    n //= 10
```

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

Cum $0 \leq c_i \leq 9 \implies 0 \leq c_i 10^{-1} \leq 0.9 \implies \left[c_i 10^{-1}\right] = 0$.

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

```
    i += 1
```

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 adevărata și la sfârșitul iterației.

La ieșirea din buclă:

$i = k+1$

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

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

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

În concluzie algoritmului este parțial corect.

II. Total corectitudinea

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

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

În concluzie algoritmului este total corect.

Laborator 3

10. Considerăm o secvență $x = (x_0, ..., x_{n-1})$ de n numere întregi, cu măcar un element pozitiv. O subsecvență a șirului este de forma $(x_i, x_{i+1}, ..., x_j)$, 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.

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

```python
print("Please insert the sequence: ", end="")
strs = input().split(' ')
x = [int(num) for num in strs if num != ""]
n = len(x)
f = [0 for i in range(n)]

f[0] = x[0]
for i in range(1, n):
    f[i] = f[i-1] + x[i]

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 $x = (x_0, ..., x_{n-1})$ și o valoare dată, s. Să se verifice daca există cel puțin doi indici i și j (nu neapărat distincți) cu proprietatea că $x_i = x_j = s$. Analizați complexitatea algoritmului propus.

```python
print("Please insert the sequence: ", end="")
strs = input().split(' ')
x = [int(num) for num in strs if str != ""]
print("Please insert s: ", end="")
s = int(input())

def f(x):
    for i in range(0, len(x)):
        for j in range(i, len(x)):
            if x[i] + x[j] == s:
                print ("Found %d + %d = %d " % (x[i] , x[j], s))
                return True
    print("Not found")
    return False
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 maximul din subtabloul parcurs la pasul respectiv.

```python
print("Please insert the array: ", end="")
strs = input().split(' ')
v = [int(num) for num in strs if num != ""]

def impl(start, end, step):
    sorted = True
    for i in range(start, end, step):
        if v[i] > v[i+1]:
            t = v[i]
            v[i] = v[i+1]
            v[i+1] = t
            sorted = False
    return sorted

begin = 0
end = len(v) - 1

while True:
    if impl(begin, end, 1): break
    if impl(end-1, begin-1, -1): break

    end -= 1
    begin += 1

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 $\{0, 1, 2, ..., m\}$. 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 algoritmul 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, $i = 0, ..., m$);

(b) se calculează tabloul frecvențelor cumulate $fc[0..m]$, $fc_i = \sum_{j=0}^{i} f_j$, $i = 0, ..., m$;

(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, 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 $k = [log10(max(x))] + 1$; $T_l :=$ timpul total de execuție al liniei l; $T(n,k) :=$ timpul de execuție total.

$T_{13} = 10k$; $T_{14} = kn$; $T_{16} = 9k$; $T_{20} = kn$; $T_{21} = kn$;
$T(n,k) = 3kn + 19k$.

$T \in O(kn)$.

Laborator 6

6. Se poate demonstra că plecând de la numărul 4, se poate obține orice număr natural diferit de zero, printr-o succesiune de operații de tipul:

- se adaugă cifra 4 la sfârșitul numărului curent;
- se adaugă cifra 0 la sfârșitul numărului curent;
- numărul curent de împarte la 2.

Propuneți un subalgoritm recursiv care să descrie cum se poate obține un număr natural $n \neq 0$, pornind de la numărul 4, aplicând operațiile de mai sus. De exemplu, pentru $n = 435$, drumul parcurs pornind de la numărul 4 este:
$4 \to 2 \to 24 \to 12 \to 6 \to 3 \to 34 \to 17 \to 174 \to 87 \to 870 \to 435$
sau pentru $n = 5$,
$4 \to 2 \to 1 \to 10 \to 5$
Indicație. Drumul până la numarul n se poate determina prin construirea drumului invers de la n la numărul 4, folosind operațiile inverse.

```python
def from4(n):
    assert(n > 0)

    def impl(n, steps):
        steps.append(n)
        if n == 4: return steps
        if n % 10 == 4 or n % 10 == 0: return impl(n // 10, steps)
        return impl(n * 2, steps)

    steps = impl(n, [])
    for num in reversed(steps[1:]):
        print(num, "-> ", end="")
    print(n)

print("Please insert a number (> 0): ", end="")
n = int(input())
from4(n)
```

```
$ python3 src.py
Please insert a number (> 0): 435
4 -> 2 -> 24 -> 12 -> 6 -> 3 -> 34 -> 17 -> 174 -> 87 -> 870 -> 435

$ python3 src.py
Please insert a number (> 0): 5
4 -> 2 -> 1 -> 10 -> 5

$ python3 src.py
Please insert a number (> 0): 231
4 -> 2 -> 1 -> 14 -> 144 -> 72 -> 36 -> 18 -> 184 -> 92 -> 924 -> 462 -> 231

$ python3 src.py
Please insert a number (> 0): 178
4 -> 44 -> 22 -> 224 -> 112 -> 56 -> 28 -> 284 -> 142 -> 1424 -> 712 -> 356 -> 178
```

11. Considerăm o scară cu $n \in \mathbb{N}^*$ trepte. Determinați numărul de moduri în care poate fi urcată scara efectuând pași de una, două sau trei trepte. Descrieți algoritmul corespunzător.

Numarul de moduri în care poate fi urcată scara este dat de următoarea relație de recurență:

$f : \mathbb{N}^* \to \mathbb{N}$,

$$f(n) = \begin{cases} 0, & n = 0 \\ 1, & n = 1 \\ 2, & n = 2 \\ f(n-1) + f(n-2) + f(n-3), & n > 2 \end{cases}$$

Observăm că $f(n-1)$ este al n-lea termen din sirul Tribonacci. Deci putem folosi următoarea formulă pentru $f(n)$:

$f(n-1) = round\left(\frac{3b}{b^2-2b+4}\left(\frac{a_+ + a_- + 1}{3}\right)^n\right)$, unde $a_\pm = \sqrt[3]{19 \pm 3\sqrt{33}}$, $b = \sqrt[3]{586 + 102\sqrt{3}}$.

```python
from math import pow, sqrt

t = (pow(19 - 3*sqrt(33), 1/3) + pow(19 + 3*sqrt(33), 1/3) + 1) / 3
b = pow(586 + 102*sqrt(33), 1/3)
left = (3*b) / (b*b - 2*b + 4)

def f(n):
    # this gives the correct answer from 1 to 53
    if n < 54: return round(left * pow(t, n+1))
    t1 = f(53)
    t2 = f(52)
    t3 = f(51)
    n -= 53

    while n > 0:
        n -= 1
        r = t1 + t2 +t3
        t3 = t2
        t2 = t1
        t1 = r
    return t1

print("Please insert a number (> 0): ", end="")
n = int(input())
print(f(n))
```

```
$ python3 src.py
Please insert a number (> 0): 4
7

$ python3 src.py
Please insert a number (> 0): 5
13

$ python3 src.py
Please insert a number (> 0): 55
222332455004452
```

Laborator 7

6. Un vector ordonat crescător are componentele în progresie aritmetică. Un singur element lipsește din progresie (sigur acesta nu este nici primul și nici ultimul). Folosind tehnica reducerii, identificați elementul lipsă.

```python
def find_missing(v):
    size = len(v)

    second = (v[0] + v[2]) // 2
    if second != v[1]: return v[2] - v[1] + v[0]
    ratio = v[1] - v[0]

    def impl(v):
        if len(v) == 1: return v[0] + ratio
        middle = len(v) // 2
        if v[0] + ratio * middle == v[middle]:
            return impl(v[middle:])
        return impl(v[:middle])

    res = impl(v)
    if res != v[-1] + ratio:
        return res
    return None

print("Please insert the vector: ", end="")
strs = input().split(' ')
v = [int(str) for str in strs if str != ""]

res = find_missing(v)
if res == None:
    res = "Nothing"

print(res, "is missing")
```

```
$ python3 src.py
Please insert the vector: 1 2 3 4 5 6 7 9 10 11
8 is missing

$ python3 src.py
Please insert the vector: 10 8 6 2
4 is missing

$ python3 src.py
Please insert the vector: 1 2 3
Nothing is missing

$ python3 src.py
Please insert the vector: 3 2 1
Nothing is missing
```

9. (*Generarea permutărior folosind algoritmul lui Heap*) Utilizați următorul algoritm pentru a genera toate permutările de ordin n ale mulțimii $\{1, 2, ..., n\}$, $n \in \mathbb{N}^*$: fiecare permutare este generată pornind de la precedenta, interschimbând o singură pereche de valori, celelalte $n-2$ valori ramânând pe loc. Pornind cu un $i = 0$, pașii algoritmul se repetă până când i devine egal cu n:

- se generează cele $(n-1)!$ permutări ale primelor $n-1$ elemente, alăturând ultimului element fiecărei dintre acestea. Asfel se generează toate permutările cu n pe ultima poziție.
- dacă n este impar, se interschimbă primul și ultimul element; dacă n este par, se interschimbă elementul de indice i și ultimul element; se incrementează i și se reiau pașii algoritmului;
- după fiecare iterație, algoritmul produce toate permutările care se termină cu elementul care tocmai a fost mutat pe ultima poziție.

```python
def permutations(v):
    def impl(v, n):
        if n == 1:
            yield v
            return
        for i in range(n):
            for p in impl(v, n-1):
                yield p
            if n % 2 == 1:
                v[0], v[n-1] = v[n-1], v[0]
            else:
                v[i], v[n-1] = v[n-1], v[i]
    return impl(v, len(v))

print("Please insert n: ", end="")
n = int(input())
v = [i for i in range(1, n+1)]

i = 0
for p in permutations(v):
    print("(", end="")
    for v in p[:-1]: print(v, end=" ")
    print(p[-1], end="")
    print(")", end=", ")
    if i % 6 == 5: print()
    i += 1
```

```
$ python3 perm.py
Please insert n: 4
(1 2 3 4), (2 1 3 4), (3 1 2 4), (1 3 2 4), (2 3 1 4), (3 2 1 4),
(4 2 3 1), (2 4 3 1), (3 4 2 1), (4 3 2 1), (2 3 4 1), (3 2 4 1),
(4 1 3 2), (1 4 3 2), (3 4 1 2), (4 3 1 2), (1 3 4 2), (3 1 4 2),
(4 1 2 3), (1 4 2 3), (2 4 1 3), (4 2 1 3), (1 2 4 3), (2 1 4 3),
```

Laborator 8

5. Propuneți un algoritm de complexitate $O(n)$ care transformă (pe loc) un tablou cu valori întregi astfel încât toate valorile negative să fie înaintea celor pozitive (partiționare cu pivot $= 0$).

```python
def partition(v):
    size = len(v)
    i = -1
    j = size

    while True:
        i += 1
        while i < size and v[i] <= 0:
            i += 1
        j -= 1
        while j >= 0 and v[j] >= 0:
            j -= 1

        if i < j:
            v[i], v[j] = v[j], v[i]
        else:
            return


print("Please insert the vector: ", end="")
strs = input().split(' ')
v = [int(str) for str in strs if str != ""]

partition(v)
print(v)
```

```
$ python3 src.py
Please insert the vector: 1 2 3 4 -1 -2 -3 4 -6 -8 1
[-8, -6, -3, -2, -1, 4, 3, 4, 2, 1, 1]

$ python3 src.py
Please insert the vector: 1 2 -3 4 -1 -2 0 1
[-2, -1, -3, 4, 2, 1, 0, 1]

$ python3 src.py
Please insert the vector: 1 2 3
[1, 2, 3]
```

6. (*Aproximarea numerică a unei integrale folosind formula trapezului*) Considerăm o funcție reală f continuă pe intervalul $[a,b]$. (...) valoarea integralei definite a lui f între limitele $[a,b]$ se aproximează prin aria trapezului cu vârfurile $(a,0),(b,0),(a,f(a)),(b,f(b))$. Deci

$$\int_a^b f(x)\,\mathrm{d}x = \begin{cases} \dfrac{b-a}{2}(f(a)+f(b)), & b-a<\varepsilon \\ \displaystyle\int_a^c f(x)\,\mathrm{d}x + \int_c^b f(x)\,\mathrm{d}x, \quad c=\dfrac{a+b}{2}, & b-a\geq\varepsilon \end{cases}$$

```python
def exp(x, iterations=20):
    factorial = 1
    pow = 1
    res = 1.0
    for i in range(1, iterations):
        factorial *= i
        pow *= x
        res += pow/factorial
    return res

def cos(x, iterations=20):
    factorial = 1
    pow = 1
    res = 1.0
    for i in range(1, iterations, 2):
        factorial *= i * (1+i)
        pow *= - x * x
        res += pow/factorial
    return res


def integrate(f, a, b, eps=1e-4):
    delta = b - a
    if delta >= eps:
        c = (a+b)/2
        return integrate(f, a, c, eps) + integrate(f, c, b, eps)
    return delta * (f(a) + f(b)) / 2

from math import pi
print("e^x:", integrate(exp, 0, 1))# should be e - 1
print("x^2:", integrate(lambda x: x*x, -1, 1))# should be 2/3
print("cos:", integrate(cos, -pi/2, pi/2))# should be 2
```

```
$ python3 integrals.py
e^x: 1.7182818289924702
x^2: 0.6666666679084301
cos: 1.9999999984680359
```

Laborator 9

9. a) Avem la dispoziție 6 culori: alb, negru, galben, verde, roșu și albastru. Afișați toate modalitățile de realizare a unui drapel tricolor folosind aceste trei culori astfel încât cele 3 culori ale drapelului să fie distincte și culoarea din mijloc este ori galben, ori verde.
b) Avem la dispoziție, în plus, șase steme de aceleași șase culori. Fiecare steag poate să aibă sau nu o stemă, dar dacă are, atunci aceasta trebuie să aibă o culoare diferită de cele trei culori deja existente în steag. Afișați toate modalitățile de realizare a steagului.

```python
#galben, verde, negru, alb, rosu, albastru, algbastru, -
colors = ['g','v','n','a','r', 'b', '-']
index = 0

flag = [-1,0,0,0]
k = 0
maxK = 3
print("Long mode (y/N):", end="")
if input()[0].lower() == 'y': maxK = 3

while k >= 0:
    maxVal = 6
    if k == 1: maxVal = 3
    elif k == 3: maxVal = 7
    if flag[k] < maxVal-1:
        flag[k] += 1
        ok = True
        for i in range(0, k):
            if flag[k] == flag[i]:
                ok = False
                break
        if not ok: continue

        if k == maxK:
            for f in flag[:maxK]: print(colors[f], end="")
            print(colors[flag[maxK]], end=" ")
            if index % 20 == 19: print()
            index += 1
        else:
           k += 1
           flag[k] = -1
    else:
        k -= 1
```

```
$ python3 flags.py
Long mode (y/N): n
gvn gva gvr gvb gnv gna gnr gnb vgn vga vgr vgb vng vna vnr vnb ngv nga ngr ngb
nvg nva nvr nvb agv agn agr agb avg avn avr avb ang anv anr anb rgv rgn rga rgb
rvg rvn rva rvb rng rnv rna rnb bgv bgn bga bgr bvg bvn bva bvr bng bnv bna bnr
```

11. Determinați toate submulțimile mulțimii $A = \{a_1, a_2, ..., a_n\}, n \in \mathbb{N}^*$, cu proprietatea că suma elementelor unei submulțimi este s.

```python
print("Please insert the set: ", end="")
strs = input().split(' ')
A = [int(num) for num in strs if num != ""]
print("Please insert s: ", end="")
s = int(input())

size = len(A)
x = [-1 for i in range(size+1)]
k = 0
while k >= 0:
    if x[k] < size-1:
        x[k] += 1
        sum = A[x[k]]

        def valid():
            global sum
            for i in range(0, k):
                if x[k] == x[i]:
                    return False
                sum += A[x[i]]
            return True

        if not valid() or sum > s: continue

        if sum == s:
            print("{", end="")
            for i in x[:k]: print(A[i], end=", ")
            print(A[x[k]], end="}, ")

        if k != size:
            x[k+1] = x[k]-1
            k += 1
    else:
        k -= 1
print()
```

```
Please insert the set: 1 2 3 4 5
Please insert s: 5
{1, 4}, {2, 3}, {5},

Please insert the set: 0 1 2 3 4 5
Please insert s: 5
{0, 1, 4}, {0, 2, 3}, {0, 5}, {1, 4}, {2, 3}, {5},

Please insert the set: 1 2 3 4
Please insert s: 44
```