Programarea Algoritmilor
– SEMINAR NR. 4 –
(grupa 131)

1. Element majoritar

   Se dă o listă cu *n* numere naturale nenule reprezentând opțiunile de vot ale celor *n* alegători pentru *k* candidați. Să se afișeze, dacă există, candidatul câștigător al alegerilor, adică acel element majoritar din listă (care are frecvența mai mare de 50%, adică nr_apariții $\geq \left[\frac{n}{2}\right] + 1$).

   Exemplu:

   v = [1, 2, 2, 1, 1, 2] => nu câștigă nimeni

   v = [2, 3, 2, 1, 2, 2] => câștigă 2

```
v = [int(x) for x in input("Introduceti numere naturale cu spatiu intre ele
si Enter la final:\n").split()]
print(v)

# Solutia (a): numarare directa -> O(n*n)
for i in set(v):
    if v.count(i) >= len(v)//2 + 1:
        print("DA", i)
        break
else:
    print("NU")

# Solutia (b): sortare + platou maximal -> O(n*log(n))
# platou = secventa de elemente consecutive egale
v2 = sorted(v); print(v2)
L_max = 0; elem_maj = None
L = 1; elem = v2[0]
for k in range(1,len(v2)):
    if v2[k] == elem:
        L += 1
        if k == len(v2) and L_max < L:
            L_max = L; elem_maj = elem
    else:
        if L_max < L:
            L_max = L; elem_maj = elem
        L = 1; elem = v2[k]
print(f"Nr maxim aparitii este {L_max} pentru elementul {elem_maj}.")

if L_max >= len(v)//2 + 1:
    print("DA", elem_maj)
else:
    print("NU")

# Solutia (c): sortare + numar aparitii element v2[n//2] -> O(n*log(n))
# Obs: elementul din mijlocul vectorului sortat este singurul
# care poate fi cel majoritar.
v2 = sorted(v); print(v2)
n = len(v2)
if v2.count(v2[n//2]) >= n//2 + 1:
    print("DA", v2[n//2])
else:
    print("NU")

# Solutia (d): cu dictionar frecvente -> O(n)

## Obs: cu dictionary comprehension ar fi O(n*n)
## d = {x : v.count(x) for x in set(v)}
```

```
d = {}
for x in v:
    if x not in d:
        d[x] = 1
    else:
        d[x] += 1
print(d)

for x in d:
    if d[x] > len(v)//2:
        print("DA", x)
        break
else:
    print("NU")

# Solutia (e): algoritmul optim = Algoritmul Boyer-Moore (1981) -> O(n)
# v = [1,2,1,2,3] => nu castiga nimeni
s = 0; c = None
for x in v:
    if s == 0:
        s = 1; c = x
    elif x == c:
        s += 1
    else:
        s -= 1

if s == 0:
    print("NU")
elif v.count(c) > len(v)//2:
    print("DA", c)
else: print("NU")
```

2. a) Scrieți o funcție care returnează o matrice triunghiulară de dimensiune n, având forma următoare:
   - prima coloană conține numerele 1, 2, 3, …, n.
   - ultima linie conține numerele n, n-1, …, 2, 1.
   - restul elementelor aflate în triunghiul de sub diagonala principală se calculează ca suma elementelor vecine de la vest, sud și sud-vest.

   Exemplu: pentru n = 4 => M=[[1], [2,15], [3,10,15], [4,3,2,1]]

```
1
2  15
3  10  15
4   3   2   1
```

   Indicație de rezolvare: întâi creăm o matrice triunghiulară cu elementele cerute pe prima coloană și ultima linie, iar în restul triunghiului punem 0. Apoi calculam de jos în sus, fiecare linie de la stânga la dreapta, elementele din locul zerourilor.

```
def matrice(n):
    m = [[i] + [0]*(i-1) for i in range(1, n)]
    m.append([i for i in range(n, 0, -1)])

    for i in range(n-2, 0, -1):
        for j in range(1, len(m[i])):
            m[i][j] = m[i][j-1] + m[i+1][j-1] + m[i+1][j]

    return m
```

   TEMĂ: Dacă liniile ar fi ordonate invers, matricea ar putea fi creată direct, de sus în jos, fără a pune întâi 0-uri?

```
4   3   2   1
3   10  15
2   15
1
```

……..

b) Scrieți o funcție care afișează o listă de liste sub formă de matrice, cu coloanele aliniate la dreapta.

```
def afisare(m):
    ncmax = len(str(max([max(linie) for linie in m])))

    for linie in m:
        for elem in linie:
            print(str(elem).rjust(ncmax), end=" ")
        print()
```

3. Scrieți o funcție cu număr variabil de parametri care furnizează toate listele care conțin un x dat ca prim parametru.
   Exemplu: cauta_x(7, [5,1,7,3,7], [2,3], [-3,7,1]) => [5,1,7,3,7], [-3,7,1]

   a) Funcție
```
def cauta(x, *liste):
    rez = []
    for lcrt in liste:
        if x in lcrt:
            rez.append(lcrt)
    return rez

x = 7
r = cauta(x, [5,1,7,3,7], [2,3], [-3,7,1])

if(r == []):
    print("Valoarea " + str(x) + " nu se gaseste in nicio lista!\n")
else:
    print("Valoarea " + str(x) + " se gaseste in listele urmatoare:")
    print(*r, sep="\n")
```

   b) Generator
```
def cauta(x, *liste):
    for lcrt in liste:
        if x in lcrt:
            yield lcrt

x = 7
r = cauta(x, [5,1,7,3,7], [2,3], [-3,7,1])

for lcrt in r:
    print(lcrt)
```

4. Sortări multicriteriale
   a) numerele pare înaintea celor impare (ordinea între valorile cu aceeași paritate nu contează)

```
def cmpParitate_1(x):
    return x % 2

v = [2, 100, 65, -10, 13, 14, 99, 27, -100, 70, -2, -20]
v.sort(key=cmpParitate_1)
print(v)
```

b) numerele pare înaintea celor impare, cele pare în ordine crescătoare și cele impare în ordine descrescătoare

```
def cmpParitate_2(x):
    return x % 2, x if x % 2 == 0 else -x

v = [2, 100, 65, -10, 13, 14, 99, 27, -100, 70, -2, -20]
v.sort(key=cmpParitate_2)
print(v)
```

5. a) Pentru un student se cunosc următoarele informații: numele, grupa și o listă cu creditele obținute la toate examenele din anul respectiv. Considerând o listă de studenți, scrieți o funcție care să adauge la fiecare student situația sa școlară: promovat (True) sau nepromovat (False). Pentru a fi considerat promovat, un student trebuie să nu aibă nici un examen nepromovat (adică un 0 în lista cu creditele), iar suma creditelor obținute să fie mai mare sau egală decât un anumit număr minim de credite dat.

```
ls = [("Popescu Ion", 135, [5,7,3,0,4]), ("Popa Anca", 131, [5,7,3]),
      ("Mihai Ana", 135, [7,3,3,2]), ("Mihai Ana", 132, [7,0,3,0]),
      ("Ionescu Clara", 131, [5,5]), ("Adam Iulia", 135, [5,5,5])]

def situatie(ls, nrminc):
    for i in range(len(ls)):
        if ls[i][2].count(0) == 0 and sum(ls[i][2]) >= nrminc:
            ls[i] += (True,)
        else:
            ls[i] += (False,)

situatie(ls, 15)
print(*ls, sep="\n", end="\n\n")
```

b) Scrieți câte o funcție comparator care să sorteze studenții după următoarele criterii:

- crescător după grupă și în fiecare grupă în ordine alfabetică
```
def cmpStudenti_1(s):
    return s[1], s[0]
```

- întâi cei promovați, apoi cei nepromovați și în fiecare categorie în ordine alfabetică
```
def cmpStudenti_2(s):
    return -s[3], s[0]
```

- descrescător după suma creditelor, iar în cazul unor sume egale în ordinea crescătoare a grupei și în ordine alfabetică în cadrul grupei
```
def cmpStudenti_3(s):
    return -sum(s[2]), s[1], s[0]
```

- în ordinea crescătoare a grupelor, în cadrul fiecărei grupe mai întâi studenții promovați, iar apoi cei nepromovați, în fiecare categorie (promovat/nepromovat) în ordinea descrescătoare a sumei creditelor și, în cazul unor sume egale, în ordine alfabetică
```
def cmpStudenti_4(s):
    return s[1], -s[3], -sum(s[2]), s[0]

L1 = sorted(ls, key=cmpStudenti_1); print(*L1, sep=" ", end="\n\n")
L2 = sorted(ls, key=cmpStudenti_2); print(*L2, sep=" ", end="\n\n")
L3 = sorted(ls, key=cmpStudenti_3); print(*L3, sep=" ", end="\n\n")
L4 = sorted(ls, key=cmpStudenti_4); print(*L4, sep=" ", end="\n\n")
```