CLUJ-NAPOCA, Martie 2026

CONCURSUL OECONOMICUS NAPOCENSIS
Secţiunea XI
Disciplina INFORMATICĂ
BAREM

I. a)

1. Citirea numărului natural N (minim 5 cifre) : 0.25p

Se citește numărul natural **N**
Se memorează în aux<-N

2. Determinarea cifrei minime și maxime : 0.75p

- Se inițializează
  `min ← 9`
  `max ← 0`
- Se parcurg cifrele numărului:
  - `c ← N % 10`
  - dacă `c < min` atunci `min <- c`
  - dacă `c > max` atunci `max <- c`
  - `N <- N / 10`

La final se obțin **cifra minimă** și **cifra maximă**.

3. Înlocuirea cifrei minime cu cifra maximă : 0.5p

Se parcurg din nou cifrele numărului initial salvat în aux:

- dacă cifra curentă `c = min`
  - atunci `c <- max`
- se păstrează celelalte cifre neschimbate.

4. Construirea numărului rezultat : 0.25p

Se formează noul număr din cifrele modificate.

5. Afișarea rezultatului : 0.25p

Se afișează numărul obținut după înlocuirea cifrelor.
Exemplu:
Intrare: `15251`

- min = `1`
- max = `5`

După înlocuire :**55255**

I. b)

1. Citirea datelor :0.25p

- Se citește numărul **n** (numărul de elemente ale vectorului).
- Se citesc cele **n elemente** ale vectorului `v`.

Exemplu:
n = 8
v = 12 5 18 30 7 8 20 11

2. Determinarea numerelor cu exact 3 factori primi : 0.5p

Pentru fiecare element `v[i]`:

- se descompune numărul în **factori primi**
- se numără factorii primi (cu multiplicitate)

Dacă **numărul factorilor primi = 3**
-> elementul trebuie **șters** din vector.
Exemple:

- 12 = 2·2·3 -> **3 factori primi ->se șterge**
- 18 = 2·3·3 -> **3 factori primi -> se șterge**
- 8 = 2·2·2 ->**3 factori primi -> se șterge**
- 5 = 5 ->**1 factor prim -> rămâne**

3. Ștergerea elementelor din vector : 0.5p

Dacă un element trebuie șters:

- se deplasează elementele din dreapta cu o poziție la stânga
- `n` se micșorează cu 1
- se continuă verificarea pentru același index.

4. Ordonarea vectorului prin metoda Bubble Sort : 0.5p

Se aplică algoritmul **Bubble Sort**:
Pentru `i = 1..n-1`

- pentru `j = 1..n-i`
  - dacă `v[j] > v[j+1]`
    - se interschimbă elementele.

Vectorul devine **ordonat crescător**.

5. Afișarea vectorului rezultat : 0.25p

Se afișează elementele rămase în vector.
Exemplu:
Vector inițial:
12 5 18 30 7 8 20 11
Se șterg:
12, 18, 8
Rămâne:
5 30 7 20 11
După **Bubble Sort**:
5 7 11 20 30

II. a)

1. Citirea datelor : 0.25p

- Se citește **n** (dimensiunea matricei).
- Se citesc elementele matricei pătratice `A[n][n]`.

Exemplu:
n = 4
Matricea:

```
1 2 3 4
5 0 1 2
4 1 2 1
2 1 2 1
```

2. Parcurgerea diagonalelor paralele cu diagonala principală : 0.75p

Se parcurg toate diagonalele paralele cu **diagonala principală**.
Numărul total de diagonale este:
2n − 1

Pentru fiecare diagonală:

- se calculează **suma elementelor de pe acea diagonală**
- suma se memorează într-un **vector v**.

3. Calculul sumelor pe diagonale – 0.5p

Pentru fiecare diagonală:

- se parcurg elementele `A[i][j]` unde diferența **j − i** este constantă.
- se adună elementele respective într-o variabilă `s`.

La final:

```
v[k] <- s
```

unde `k` reprezintă poziția în vector.

4. Construirea vectorului rezultat : 0.25p

Vectorul va conține **sumele tuturor diagonalelor paralele cu diagonala principală**, în ordinea parcurgerii.

Numărul de elemente din vector:

2n − 1

5. Afișarea vectorului : 0.25p

Se afișează elementele vectorului `v`.

Exemplu:

Pentru matricea dată:

```
1 2 3 4
5 0 1 2
4 1 2 1
2 1 2 1
```

Sumele diagonalelor paralele cu diagonala principală sunt:

4 5 4 4 8 5 2

Date de intrare:

n = 4

Date de ieșire:

4 5 4 4 8 5 2

II. b)

1. Citirea datelor : 0.25p

- Se citește numărul **n**.
- Se citesc cele **n numere naturale** ale vectorului `v`.

2. Determinarea celui mai mare număr prim din vector : 0.5p

Se definește o **funcție de verificare a numerelor prime**.

Pentru fiecare element `v[i]`:

- dacă `v[i]` este **număr prim**
- și `v[i] > maxim`
    - atunci `maxim <- v[i]`

La final se obține **cel mai mare număr prim din vector**.

3. Extragerea cifrelor numărului prim : 0.25p

Se extrag cifrele numărului **maxim** și se memorează într-un vector `c`.

Se determină și **numărul de cifre k**.

4. Generarea permutărilor prin algoritmul Backtracking : 0.75p

Se generează toate permutările cifrelor folosind **backtracking**.

Ideea algoritmului:

- se construiește recursiv un vector `p`
- `p[i]` reprezintă cifra de pe poziția i în permutare
- se folosește un vector `folosit[]` pentru a nu repeta cifrele

Schema algoritmului:

```
BT(k)
   pentru i <- 1..n
```

```
dacă folosit[i] = 0 atunci
   p[k] <- c[i]
   folosit[i] <- 1
   dacă k = n atunci
      se formează numărul
      suma <- suma + numărul
   altfel
      BT(k+1)
   folosit[i] <- 0
```

5. Afișarea rezultatului : 0.25p

Se afișează **suma numerelor obținute din permutările cifrelor**.
Exemplu:
Cel mai mare număr prim: **123**
Permutări:
123, 132, 213, 231, 312, 321
Sumă:
1332.
Baremul de corectare are caracter orientativ; se punctează orice metodă de rezolvare corectă, în conformitate cu regulamentul concursului.