

Tema - Arhitectura Sistemelor de Calcul

Seriile 13, 14, 15

Noiembrie 2023

Cuprins

1	Detalii administrative	2
1.1	Deadline	2
1.2	Reamintirea punctajului pe tema	2
1.3	Transmitere	2
1.4	Ce se va transmite	2
1.5	Cum se va face evaluarea	2
1.6	Alte observatii	3
2	Formularea temei	4
3	Cerinte	7
3.1	Cerinta 0x00 - 5p	7
3.2	Cerinta 0x01 - 2.5p	8
3.3	Cerinta 0x02 - 2.5p	9
4	Inputuri concrete	10
4.1	Cerinta 0x00	10
4.2	Cerinta 0x01	10
4.3	Cerinta 0x02	11

1 Detalii administrative

1.1 Deadline

Puteți trimite soluțiile cel târziu pe 6 Ianuarie 2024, ora 23:59.

1.2 Reamintirea punctajului pe tema

Tema valorează 40% din nota la acest laborator (conform Cursului 0x00), și este necesară obținerea notei 5 pentru promovare.

1.3 Transmitere

Veti trimite soluțiile în următoarele formulare, în funcție de serie:

- seria 13 (grupele 131 și 132): TBD
- seria 13 (grupele 133 și 134): TBD
- seria 14: TBD
- seria 15: TBD
- restanțieri: TBD

Link-urile către formulare vor fi completate în perioada imediat următoare. Va rugăm să reveniți în zilele următoare asupra acestui document.

1.4 Ce se va transmite

Se vor încărca în formular **trei surse** (cate una pentru fiecare cerință) cu denumirile **grupa_-nume_prenume_0.s**, **grupa_-nume_prenume_1.s**, respectiv **grupa_-nume_prenume_2.s**. Dacă aveți mai multe nume / prenume, veți încărca surse denumite, de exemplu, **172_GeorgescuXulescu_IonVasile_0.s**. Este **important** să încărcați surse cu denumirea corectă, deoarece testarea va fi **automată**.

1.5 Cum se va face evaluarea

Există doi pași pentru obținerea notei:

- se verifică toate sursele să nu fie cazuri de plagiat. În cazul în care se detectează plagiat, se face automat sesizare către *Comisia de Etică a Universității din București*;
- sursele care au trecut de verificarea anti-plagiat, vor fi testate automat.

Important! Studenții care au alte configurări față de cele pe care lucram la laborator, trebuie să precizeze acest lucru în formularul în care transmit tema, pentru a putea efectua evaluarea și pentru a nu primi 0 implicit.

1.6 Alte observatii

1. Nu va interzicem sa discutati idei intre voi, dar aveti grija, deoarece este o diferenta importanta intre a da o idee si a da codul direct.
2. Nu folositi convertoare automate din C/C++/alte limbaje in x86, le-am folosit si noi si recunoastem fara dificultate un cod care nu este scris de voi.

2 Formularea temei

Conway's Game of Life este un *zero-player game* bidimensional, inventat de matematicianul John Horton Conway in anul 1970. Scopul acestui joc este de a observa evolutia unui sistem de celule, pornind de la o configuratie initiala, introducand reguli referitoare la moartea, respectiv crearea unei noi celule in sistem. Acest sistem evolutiv este *Turing-complete*.

Starea unui sistem este descrisa de starea cumulata a celulelor componente, iar pentru acestea avem urmatoarele patru reguli:

1. **Subpopulare.** Fiecare celula (care este in viata in generatia curenta) cu mai putin de doi vecini in viata, moare in generatia urmatoare.
2. **Continuitate celule vii.** Fiecare celula (care este in viata in generatia curenta), cu doi sau trei vecini in viata, va exista si in generatia urmatoare.
3. **Ultrapopulare.** Fiecare celula (care este in viata in generatia curenta), care are mai mult de trei vecini in viata, moare in generatia urmatoare.
4. **Creare.** O celula moarta care are exact trei vecini in viata, va fi creata in generatia urmatoare.
5. **Continuitate celule moarte.** Orice alta celula moarta, care nu se incadreaza in regula de creare, ramane o celula moarta.

Vecinii unei celule se considera urmatorii 8, intr-o matrice bidimensionala:

a_{00}	a_{01}	a_{02}
a_{10}	celula curenta	a_{12}
a_{20}	a_{21}	a_{22}

Definim starea unui sistem la generatia n ca fiind o matrice $\mathcal{S}_n \in \mathcal{M}_{\mathfrak{m} \times \mathfrak{n}}(\{0, 1\})$ (\mathfrak{m} - numarul de linii, respectiv \mathfrak{n} - numarul de coloane), unde elementul 0 reprezinta o celula moarta, respectiv 1 reprezinta o celula in viata (in generatia curenta).

Definim o k -evolutie ($k \geq 0$) a sistemului o iteratie $\mathcal{S}_0 \rightarrow \mathcal{S}_1 \rightarrow \dots \rightarrow \mathcal{S}_k$, unde fiecare \mathcal{S}_{i+1} se obtine din \mathcal{S}_i , aplicand cele cinci reguli enuntate mai sus.

Observatie. Pentru celulele aflate pe prima linie, prima coloana, ultima linie, respectiv ultima coloana, se considera extinderea la 8 vecini, prin considerarea celor care **nu** se afla in matrice ca fiind celule moarte.

Exemplificare. Fie urmatoarea configuratie initiala \mathcal{S}_0 :

0	1	1	0
1	0	0	0
0	0	1	1

In primul rand, vom considera extinderea acestei matrice \mathcal{S}_0 de dimensiuni 3×4 intr-o matrice extinsa $\overline{\mathcal{S}}_0$ de dimensiuni 5×6 , astfel:

0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	0	0	0
0	0	0	1	1	0
0	0	0	0	0	0

În cele ce urmează, vom lucra doar în interiorul matricei principale, dar considerând extinderea pentru procesarea corectă a vecinilor. Vom parcurge fiecare element, și vom vedea ce regulă evolutivă putem aplica. De exemplu, pentru elementul de pe poziția (0,0) în matricea inițială, vom aplica regula de continuitate a celulelor moarte, deoarece este o celulă moartă și nu are exact trei vecini în viață.

Următoarea celulă este în viață, și are exact doi vecini în viață, astfel ca se aplică regula continuității celulelor în viață.

Pentru celulă de pe poziția (0,2) în \mathcal{S}_0 , observăm că are un singur vecin, astfel ca se aplică regula de subpopulare - celulă va muri în generația următoare.

Urmand același raționament pentru toate celulele, configurația sistemului într-o iteratie (în $\overline{\mathcal{S}}_1$) va fi:

0	0	0	0	0	0
0	0	1	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
0	0	0	0	0	0

Schema de criptare simetrică. Definim o cheie de criptare (pornind de la o configurație inițială \mathcal{S}_0 și o k -evoluție) ca fiind operația $\langle \mathcal{S}_0, k \rangle$, care reprezintă tabloul **unidimensional** de date (înțeles ca sir de biți) obținut în urma concatenării liniilor din matrice din matricea extinsă obținută, $\overline{\mathcal{S}}_k$.

De exemplu, pornind de la configurația anterioară \mathcal{S}_0 , și aplicând doar o 1-evoluție, se obține matricea extinsă $\overline{\mathcal{S}}_1$ descrisă anterior, care va avea ca efect al aplicării operației $\langle \mathcal{S}_0, 1 \rangle$ obținerea următorului tablou unidimensional (înțeles ca sir de biți):

0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0

Considerăm m un mesaj în clar (un sir de caractere fără spații). Criptarea $\{m\}_{\langle \mathcal{S}_0, k \rangle}$ va însemna XOR-rea mesajului în clar m cu rezultatul dat de $\langle \mathcal{S}_0, k \rangle$. Sunt următoarele cazuri:

- dacă mesajul și cheia au aceeași lungime, se XOR-ează element cu element, până se obține rezultatul;
- dacă mesajul este mai scurt decât cheia, se folosește doar prima parte din cheie, corespunzătoare lungimii mesajului;
- dacă mesajul este mai lung decât cheia, se consideră replicarea cheii de oricâte ori este nevoie pentru a cripta întreg mesajul.

Considerăm ca $m = \text{parola}$, și utilizăm drept cheie $\langle \mathcal{S}_0, 1 \rangle$, unde \mathcal{S}_0 este configurația inițială descrisă anterior. Am văzut că rezultatul obținut este sirul de biți:

0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0

pe care îl vom considera fără spații:

000000001000000010000000000000

Pentru a efectua criptarea, trebuie să analizăm sirul de criptat, și anume **parola**. Vom vedea care este codificarea ASCII (binară) a fiecărui caracter din acest sir:

p	01110000
a	01100001
r	01110010
o	01101111
l	01101100
a	01100001

Sirul **parola** va fi, astfel, sirul binar

011100000110000101110010011011110110110001100001

Observam, in acest caz, ca sirul de criptat este mai lung decat cheia de criptare, astfel ca daca incercam acum o XOR-are, am avea urmatoarea situatie:

```
mesaj = 011100000110000101110010011011110110110001100001
cheie = 000000001000000010000000000000
```

Vom considera, in acest caz, ca vom concatena iar cheia la cheia initiala:

```
mesaj = 011100000110000101110010011011110110110001100001
cheie = 00000000100000001000000000000000000000001000000010000000000000
```

Iar apoi vom pastra din noua cheie doar cat ne este suficient pentru a cripta mesajul:

```
mesaj = 011100000110000101110010011011110110110001100001
cheie = 00000000100000001000000000000000000000001000000010
```

Mesajul criptat se va obtine prin XOR-are element cu element, stiind ca $0 \text{ XOR } 0 = 1 \text{ XOR } 1 = 0$, respectiv $0 \text{ XOR } 1 = 1 \text{ XOR } 0 = 1$. In acest caz,

```
mesaj = 011100000110000101110010011011110110110001100001
cheie = 00000000100000001000000000000000000000001000000010
cript = 011100001110000111110010011011110110111001100011
```

Mesajul criptat afisat va fi in hexadecimal (pentru a nu fi probleme de afisare a caracterelor), iar in acest caz vom avea:

```
cript = 0111 0000 1110 0001 1111 0010 0110 1111 0110 1110 0110 0011
      =  7   0   E   1   F   2   6   F   6   E   6   3
      = 0x70E1F26F6E63
```

Pentru decriptare se aplica acelasi mecanism, mesajul decriptat se va XOR-a cu cheia calculata, si vom avea in final $m \text{ XOR } k \text{ XOR } k = m$. ($k \text{ XOR } k = 0$, iar $m \text{ XOR } 0 = m$, din asociativitatea lui XOR, respectiv din regulile de calcul). La decriptare, mesajul nu va fi afisat in hexadecimal, ci in clar.

3 Cerinte

In cadrul acestei teme aveti trei cerinte - o cerinta pentru 5p, una pentru 3p, respectiv o cerinta pentru alte 2p.

Important! NU dati inputul manual la fiecare retestare a programului! Sunt inputuri lungi, care va vor costa timp. Creati-va un fisier, de exemplu `input.txt`, in care scrieti inputul dorit, iar dupa ce aveti un executabil, de exemplu `task00`, pe care in mod normal l-ati fi rulat cu `./task00`, rulati comanda `./task00 < input.txt`. Astfel, continutul din fisier va fi redirectat la **STDIN**, exact ca atunci cand ati fi introdus manual valorile. Folositi aceasta informatie si pentru a va testa mai multe inputuri, creandu-va fisiere `input0.txt`, `input1.txt` etc., si testandu-le cu `./task00 < input0.txt`, `./task00 < input1.txt` etc.

Important! Toate sirurile de caractere (utilizate pentru afisare) pe care le definiti in sectiunea `.data` vor avea, la final, caracterul `\n`!

Important! Cerintele 0x01, respectiv 0x02 **NU** sunt cascadeate, astfel ca ele pot fi rezolvate independent.

3.1 Cerinta 0x00 - 5p

Se citesc de la tastatura (**STDIN**) numarul de linii m , numarul de coloane n , numarul de celule vii p , pozitiile celulelor vii din matrice, respectiv un numar intreg k . **Atentie!** In citirea inputului se considera matricea initiala, **neextinsa**: se citește configuratia initiala \mathcal{S}_0 , si **NU** $\overline{\mathcal{S}_0}$! De exemplu, pentru matricea din prezentarea cerintei, inputul ar fi urmatorul:

```
3          // m - numarul de linii
4          // n - numarul de coloane
5          // p - numarul celulelor vii
0
1          // prima celula vie este in (0,1)
0
2          // a doua celula vie este in (0,2)
1
0          // a treia celula vie este in (1,0)
2
2          // a patra celula vie este in (2,2)
2
3          // a cincea celula vie este in (2,3)
5          // numarul intreg k
```

Se cere, la acest pas, afisarea la **STDOUT** a configuratiei sistemului dupa o k -evolutie. **Atentie!** Se va afisa starea sistemului \mathcal{S}_{k-1} si **NU** matricea extinsa $\overline{\mathcal{S}_{k-1}}$!

Matricea va fi afisata uzual, iar in acest caz, rezultatul este:

```
0 0 0 0
0 0 0 0
0 0 0 0
```

(toate celulele mor dupa cea de-a doua iteratie).

Observatie: Elementele de pe linie vor fi afisate cu un spatiu intre ele, iar la finalul fiecarei linii, veti afisa un caracter `\n`. **Si dupa ultima linie veti afisa acel caracter \n!**

Se garanteaza urmatoarele:

- $1 \leq m \leq 18$
- $1 \leq n \leq 18$
- $p \leq n \cdot m$
- $k \leq 15$

3.2 Cerinta 0x01 - 2.5p

Se citesc de la tastatura (**STDIN**) numarul de linii **m**, numarul de coloane **n**, numarul de celule vii **p**, pozitiile celulelor vii din matrice, un numar intreg **k**, un intreg **o** care poate fi 0 sau 1 (0 pentru criptare, 1 pentru decriptare), respectiv un mesaj **m** care poate fi in clar, pentru criptare (un sir de forma **0x...**, pentru decriptare). Se cere criptarea/decriptarea mesajului primit, conform cheii care trebuie calculate, conform specificatiilor din formularea temei.

Un input ar putea fi urmatorul:

```
3          // m - numarul de linii
4          // n - numarul de coloane
5          // p - numarul celulelor vii
0
1          // prima celula vie este in (0,1)
0
2          // a doua celula vie este in (0,2)
1
0          // a treia celula vie este in (1,0)
2
2          // a patra celula vie este in (2,2)
2
3          // a cincea celula vie este in (2,3)
1          // numarul intreg k
0          // se va efectua CRIPTARE
parola     // mesajul in clar de criptat
```

In acest caz, rezultatul este 0x70E1F26F6E63.

Pentru inputul:

```
3          // m - numarul de linii
4          // n - numarul de coloane
5          // p - numarul celulelor vii
0
1          // prima celula vie este in (0,1)
0
2          // a doua celula vie este in (0,2)
1
0          // a treia celula vie este in (1,0)
2
2          // a patra celula vie este in (2,2)
2
3          // a cincea celula vie este in (2,3)
1          // numarul intreg k
1          // se va efectua DECRYPTARE
0x70E1F26F6E63 // sirul hexa de decriptat
```


rezultatul este `parola`.

Observatie. Se garanteaza aceleasi conditii ca in cazul primei cerinte, respectiv se garanteaza corectitudinea datelor de intrare, mesajele de criptat vor fi mesaje fara spatii, formate din caractere alpha-numerice (cifre, litere mici, litere mari), iar mesajele de decriptat vor fi siruri hexadecimale care vor incepe cu `0x` si vor contine simboluri din multimea $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$. Mesajele (considerate in clar) vor avea maxim 10 caractere!

3.3 Cerinta 0x02 - 2.5p

Sa se refaca, intr-un fisier sursa separat (denumit conform sectiunii 1.4.), cerinta 0x00, astfel incat inputul sa fie citit dintr-un fisier `in.txt`, iar outputul sa fie scris intr-un fisier `out.txt`, utilizand exclusiv **apeluri de sistem** (NU apelarea functiilor din C/C++)!

4 Inputuri concrete

În această secțiune, vom rescrie inputurile fără comentariile asociate, pentru a fi clară forma în care le veți primi.

4.1 Cerinta 0x00

Input	Output
3	0 0 0 0
4	0 0 0 0
5	0 0 0 0
0	
1	
0	
2	
1	
0	
2	
2	
2	
3	
5	

Tabela 1: Input Cerinta 0x00

4.2 Cerinta 0x01

Input	Output
3	0x70E1F26F6E63
4	
5	
0	
1	
0	
2	
1	
0	
2	
2	
2	
3	
1	
0	
parola	

Tabela 2: Input Cerinta 0x01 - criptare

Input	Output
3	parola
4	
5	
0	
1	
0	
2	
1	
0	
2	
2	
2	
3	
1	
1	
0x70E1F26F6E63	

Tabela 3: Input Cerinta 0x01 - decriptare

4.3 Cerinta 0x02

Sunt aceleasi inputuri ca la Cerinta 0x00, doar ca vor fi citite de program din fisierul `in.txt` si apoi afisate in `out.txt`, utilizand exclusiv apeluri de sistem.