MODALITATEA DE DESFĂȘURARE A EXAMENULUI LA DISCIPLINA "PROGRAMAREA ALGORITMILOR"

- Examenul la disciplina "Programarea algoritmilor" se va desfășura în ziua de 20.01.2022, între orele 9^{00} și 11^{30}, astfel:
  - 09^{00} – 09^{30}: efectuarea prezenței studenților
  - 09^{30} – 11^{30}: desfășurarea examenului
  - 11^{30} – 12^{00}: verificarea faptului că sursele trimise de către studenți au fost salvate pe platforma MS Teams
- Testul se va desfășura pe platforma MS Teams, iar pe tot parcursul desfășurării sale, de la ora 09^{00} la ora 12^{00}, studenții trebuie să fie conectați pe canalul dedicat cursului de "Programarea algoritmilor" corespunzător seriei lor.
- În momentul efectuării prezenței, fiecare student trebuie să aibă pornită camera video în MS Teams și să prezinte buletinul sau cartea de identitate. Dacă dorește să-și protejeze datele personale, studentul poate să acopere codul numeric personal și/sau adresa!
- În timpul desfășurării testului studenții pot să închidă camera video, dar trebuie să o deschidă dacă li se solicită acest lucru de către un cadru didactic!

- Toate subiectele se vor rezolva folosind limbajul Python.
- Subiectul 1 este obligatoriu, iar dintre subiectele 2, 3 și 4 se vor rezolva CEL MULT DOUĂ, la alegere.
- Citirea datelor de intrare se va realiza de la tastatură, iar rezultatele vor fi afișate pe ecran.
- Se garantează faptul că datele de intrare sunt corecte.
- Operațiile de sortare se vor efectua folosind funcții sau metode predefinite din limbajul Python.
- Pentru subiectul 1 nu contează complexitatea soluției propuse.
- Rezolvările subiectelor alese dintre subiectele 2, 3 și 4 trebuie să conțină:
  - o scurtă descriere a algoritmului și o argumentare a faptului că acesta se încadrează într-o anumită tehnică de programare;
  - în cazul problemelor rezolvate folosind metoda Greedy sau metoda programării dinamice se va argumenta corectitudinea criteriului de selecție sau a relațiilor de calcul;
  - în cazul subiectelor unde se precizează complexitatea maximă pe care trebuie să o aibă soluția, se va argumenta complexitatea soluției propuse și vor primi punctaj maxim doar soluțiile corecte care se încadrează în complexitatea cerută;
  - în cazul problemei rezolvate folosind metoda backtracking nu contează complexitatea soluției propuse, dar se va ține cont de eficiența condițiilor de continuare;
  - în fiecare program Python se va preciza, pe scurt, sub forma unor comentarii, semnificația variabilelor utilizate.
- Rezolvările corecte care nu respectă restricțiile indicate vor primi punctaje parțiale.
- Se acordă 1 punct din oficiu.

- Rezolvările tuturor subiectelor se vor scrie de mână, folosind pix/stilou cu culoarea pastei/cernelii albastră sau neagră. Pe fiecare pagina studentul își va scrie numele și grupa, iar paginile trebuie să fie numerotate.
- Înainte de expirarea timpului alocat examenului, toate paginile vor fi fotografiate/scanate clar, în ordinea corectă, și transformate într-un singur fișier PDF care va fi încărcat în Google Drive folosind un anumit formular.

- Numele fișierului PDF trebuie să respecte șablonul *grupa_nume_prenume.pdf*. De exemplu, un student cu numele Popescu Ion Mihai din grupa 131 trebuie să denumească fișierul care conține rezolvările tuturor subiectelor astfel: *131_Popescu_Ion_Mihai.pdf*.

Subiectul 1 – limbajul Python – 3 p.

a) Scrieți o funcție **aparitii** care primește un număr variabil de numere naturale și returnează un dicționar care conține pentru fiecare număr primit ca parametru o listă de perechi în care pentru fiecare cifră distinctă a numărului avem perechea formată din valoarea cifrei și frecvența sa în acel număr. De exemplu, pentru apelul **aparitii(1011, 234, 8158558)** funcția trebuie să returneze dicționarul {1011: [(1,3), (0,1)], 234: [(2,1), (3,1), (4,1)], 8158558: [(1,1), (5,3), (8,3)]}. **(1.5 p.)**

b) Știind că matricea pătratică **m** este memorată sub forma unei liste de liste, înlocuiți punctele de suspensie din instrucțiunea **numere = […]** cu o secvență de inițializare (*list comprehension*) astfel încât, după executarea sa, lista să conțină pătratul elementelor aflate pe diagonala principală a matricei **m**. De exemplu, pentru matricea `m = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]` trebuie ca lista **numere** să fie `[1, 25, 81]`. **(0.5 p.)**

c) Considerăm următoarea funcție recursivă:

```
def f(lista, p, u):
    if u - p <= 2:
        return sum(lista[p: u+1])
    k = (u - p + 1) // 3
    aux_1 = sum(lista[p: p + k])
    aux_2 = f(lista, p + k + 1, p + 2 * k - 1)
    aux_3 = sum(lista[p+2*k+1: u+1])
    return sum([aux_1, aux_2, aux_3])
```

Determinați complexitatea funcției apelată pentru o listă **L** formată din **n** numere întregi astfel: **f(L, 0, n-1)**. **(1 p.)**

Subiectul 2 – metoda Greedy (3 p.)
Complexitatea maximă a soluției: $\mathcal{O}(n)$

Canalul Pythonic Way este unul foarte strâmt, astfel încât un singur nufăr încape pe lățimea sa. Pe canal sunt înșiruiți, de la capătul din stânga spre capătul din dreapta, mai mulți nuferi, iar pe fiecare nufăr este scris un număr natural nenul reprezentând numărul maxim de nuferi peste care poate să sară o broscuță aflată pe nufărul respectiv. Astfel, dacă pe nufărul cu numărul de ordine i este scris numărul k, atunci o broscuță poate să sară pe oricare dintre nuferii cu numerele de ordine $i+1, i+2, \ldots, i+k$. Primul nufăr, având numărul de ordine 1, se află la capătul din stânga al canalului, iar ultimul nufăr, având numărul de ordine n, se află la capătul din dreapta. Într-o bună zi, Lily, una dintre broscuțele care trăiesc în Pythonic Way, s-a hotărât să iasă din lumea strâmtă a canalului și să plece în lumea largă. Deoarece Lily are o fire melancolică, ea vrea să plece de pe primul nufăr, să ajungă pe ultimul nufăr (pentru a mai vedea încă o dată întreg canalul Pythonic Way) și abia apoi să iasă din canal (fără să mai facă niciun salt). Pentru a nu avea timp să se răzgândească, Lily vrea să ajungă pe ultimul nufăr cât mai repede, adică folosind un număr minim de sărituri care respectă restricția precizată anterior. Scrieți un program în limbajul Python care să citească de la tastatură numerele scrise pe cei n nuferi și să afișeze pe ecran un traseu format din numerele de ordine ale nuferilor pe care trebuie să sară Lily pentru a ieși din canal plecând de pe primul nufăr și efectuând un număr minim de sărituri. Dacă există mai multe trasee cu proprietatea cerută, atunci se va scrie oricare dintre ele. Numerele de ordine ale nuferilor din traseu vor fi despărțite între ele prin câte un spațiu. Fiecare săritură efectuată de Lily trebuie să respecte restricția precizată în enunț.

Exemplu:

Date de intrare	Date de ieșire
2 3 1 5 3 2 2 5	1 2 4 8

Explicații: Lily trebuie să efectueze cel puțin 3 sărituri, plecând de pe primul nufăr, pentru a ieși din canalul Pythonic Way. Un traseu corect pe care îl poate urma Lily este 1, 2, 4, 8. Un alt traseu corect este 1, 3, 4, 8.

Subiectul 3 – metoda Programării Dinamice (3 p.)
Complexitatea maximă a soluției: $O(n^2)$

Schiorel a urcat cu telecabina până în vârful stațiunii și își dorește să ajungă cât mai obosit la una din cabanele stațiunii ca să se poată hidrata cât mai intens. Stațiunea e reprezentată ca o matrice pătratică de dimensiune n în care în fiecare pătrat avem gradul de oboseală pe care îl va acumula Schiorel dacă trece prin acel câmp sau -1, însemnând că în acel câmp avem o cabană. Schiorel poate începe traseul de oriunde de pe linia de sus și se poate opri la orice cabană, voi trebuie să-l ajutați să ajungă cât mai obosit! Schiorel poate coborî drept sau în diagonală, adică din (i,j) în {(i+1,j-1), (i+1, j), (i+1, j+1)} evident fără a părăsi stațiunea.
Scrieți un program Python care citește de la tastatură dimensiunea tablei *n* și pentru fiecare pătrățică de coordonate (i,j) (cu i=1,..,n, j=1,...,n) o valoare c_{ij} cu semnificația:

- dacă c_{ij} este număr natural, el reprezintă gradul de oboseală acumulat de Schiorel când trece prin acea zona a stațiunii.
- dacă c_{ij} este -1, atunci în acea zonă se află o cabană!

și afișează un traseu al lui Schiorel până la o cabană, astfel încât să ajungă cât mai obosit (odată ajuns la o cabană, Schiorel se oprește și nu mai continuă drumul).

Intrare de la tastatură	Ieșire pe ecran
4 5 2 6 11 -1 7 1 -1 4 10 3 5 1 6 -1 2	Gradul de oboseala maxim 23 1 3 2 2 3 2 4 3

Explicații: Pârtia este o matrice de dimensiuni 4x4 în care elementele reprezintă oboseala acumulată trecând prin acel punct, respectiv -1 în locul în care avem cabană. Pe traseul (1,3), (2,2), (3,2), (4,3) acumulează oboseala 23 (!traseul trebuie să înceapă pe prima linie și să se termine cu o pătrățică de valoare -1).

Intrare de la tastatură	Ieșire pe ecran
4 5 2 6 31 -1 7 -1 -1 4 10 3 5 1 6 -1 2	Gradul de oboseala maxim 31 1 4 2 3

Odată ajuns la o cabana, Schiorel se oprește din schiat.
P.S. Lui Schiorel îi place oboseala.

Subiectul 4 – metoda Backtracking (3 p.)

a) Un număr natural se numeşte *p-mărginit* ($0 \leq p \leq 9$) dacă valoarea absolută a diferenţei dintre oricare două cifre ale sale este cel mult egală cu *p*. De exemplu, numărul 27383 este 6-*mărginit*, iar numărul 2022 este 2-*mărginit*. Scrieţi un program Python care să citească de la tastatură numerele naturale *p* şi *c*, după care afişează toate numerele naturale *p-mărginite* formate din cifre nenule având suma cifrelor egală cu *c* sau mesajul "*Imposibil*" dacă nu există niciun astfel de număr. **(2.5 p.)**

Exemplu:
Pentru $p = 3$ şi $c = 6$ trebuie afişate următoarele 30 de numere (nu neapărat în această ordine):

111111	**1221**	**222**
11112	**123**	**231**
11121	**1311**	**24**
1113	**132**	**3111**
11211	**141**	**312**
1122	**21111**	**321**
1131	**2112**	**33**
114	**2121**	**411**
12111	**213**	**42**
1212	**2211**	**6**

b) Precizaţi cum ar trebui modificată o singură instrucţiune din program astfel încât să fie afişate doar numerele *p-mărginite* formate din cifre nenule având suma cifrelor egală cu *c* şi prima cifră egală cu ultima. Pentru exemplul anterior, aceste soluţii sunt cele scrise cu roşu. **(0.5 p.)**