

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 9⁰⁰ și 11³⁰, astfel:
 - 09⁰⁰ – 09³⁰: efectuarea prezenței studenților
 - 09³⁰ – 11³⁰: desfășurarea examenului
 - 11³⁰ – 12⁰⁰: 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 09⁰⁰ la ora 12⁰⁰, 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) Considerând un dicționar cu același format ca dicționarul de la punctul a), scrieți o secvență de inițializare pentru o listă (*list comprehension*) care să conțină cheile din dicționar care au cel puțin 3 cifre distincte. De exemplu, pentru dicționarul anterior, se va obține lista [234, 8158558]. **(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: $O(n \log_2 n)$

O balanță veche s-a defectat și acum se echilibrează nu doar pentru două obiecte având aceeași greutate, ci pentru orice două obiecte cu proprietatea că modulul diferenței dintre greutatea lor este mai mic sau egal decât un număr real g . Scrieți un program Python care citește de la tastatură un număr natural n , un număr real g și greutatea a n obiecte, după care afișează pe ecran numărul maxim de perechi de obiecte care echilibrează balanța defectă, precum și perechile respective, știind că orice obiect poate să facă parte din cel mult o pereche. Fiecare pereche afișată trebuie să fie de forma $x + y$, unde x și y sunt numerele de ordine ale celor două obiecte din pereche (obiectele sunt numerotate începând de la 1). Greutățile tuturor obiectelor și diferența g sunt exprimate prin numere reale strict pozitive, reprezentând grame. Nu contează ordinea în care se vor afișa perechile de obiecte pe ecran și nici ordinea numerelor de ordine ale obiectelor dintr-o pereche.

Exemplu:

Date de intrare	Date de ieșire
10	3
8.5	3 + 2
21.25	10 + 4
12	1 + 8
6.05	
20.7	
23.8	
22	
33.25	
21	
48.15	
62.20	

Explicații: Avem $n = 10$ și $g = 8.5$. Se pot forma maxim 3 perechi de obiecte care pot echilibra balanța defectă. Soluția nu este unică, o altă soluție corectă obținându-se, de exemplu, înlocuind perechea 1 + 6 cu perechea 1 + 5.

Subiectul 3 – metoda Programării Dinamice (3 p.)

Complexitatea maximă a soluției: $O(nM)$

Pia a hotărât să aloce M minute în care să facă doar activitățile ei preferate. Și-a făcut o listă cu n activități pe care le-a numerotat $1, \dots, n$ și a estimat pentru fiecare activitate $i = 1, \dots, n$ durata d_i în minute. Ea ar vrea să își ocupe cât mai mult timp cu activitățile preferate și ar vrea să aleagă ce activități va face astfel încât timpul total dedicat activităților alese (egal cu suma duratelor lor) să fie cât mai apropiat de M (poate și depăși M , dar diferența între suma duratelor activităților alese și M trebuie să fie cât mai mică). Scrieți un program Python care să citească de la tastatură numărul de minute M , numărul de activități n și duratele d_1, \dots, d_n și afișează ce activități să facă Pia astfel încât durata totală a acestora să fie cât mai apropiată de M .

Intrare de la tastatură	Ieșire pe ecran
21 6 5 10 5 4 10 3	1 3 5

Explicații: suma duratelor activităților 1, 3 și 5 este $5+5+10 = 20$ și nu există o mulțime de activități cu suma duratelor 21. Soluția optimă nu este unică, o altă soluție optimă este de exemplu cea formată cu activitățile 2, 5 sau 3, 4, 5, 6 (în acest ultim caz durata totală este 22, cu 1 mai mare decât M , deci la fel de apropiată de M ca și 20)

Intrare de la tastatură	Ieșire pe ecran
20 4 10 11 4 12	1 2

Explicații: suma duratelor activităților 1, 2 este 21 și nu există o mulțime de activități cu suma duratelor 20 (deci cea mai apropiată durată totală de $M=20$ pe care o putem obține este 21)

Subiectul 4 – metoda Backtracking (3 p.)

a) Moș Crăciun are nevoie de m mașinuțe și apelează din nou la cei n spiriduși ai săi. El îl roagă pe fiecare spiriduș i ($1 \leq i \leq n$) să-i spună care este numărul minim a_i și numărul maxim b_i de mașinuțe pe care ar vrea să le facă. Moș Crăciun ar vrea să îi pună pe spiriduși să facă cele m mașinuțe care-i trebuie pentru Crăciun, dar respectând opțiunile fiecărui spiriduș. Dacă vreți să primiți și voi una dintre mașinuțe, trebuie să-l ajutați pe Moș Crăciun scriind un program Python care să citească de la tastatură numerele m, n și opțiunile a_i și b_i ale fiecăruia dintre cei n spiriduși, după care să afișeze pe ecran toate modalitățile în care Moș Crăciun poate distribui producția de mașinuțe spiridușilor astfel încât spiridușul i să producă un număr de mașinuțe cuprins între a_i și b_i de mașinuțe sau mesajul "*Imposibil*" dacă nu există nicio modalitate de distribuție care să respecte cerințele precizate anterior. **(2.5 p.)**

Exemplu:

Pentru $m = 16, n = 3, a_1 = 1, b_1 = 6, a_2 = 0, b_2 = 7, a_3 = 4, b_3 = 8$ trebuie să fie afișate următoarele 20 de modalități de distribuție (nu neapărat în această ordine):

```
1 7 8
2 6 8
2 7 7
3 5 8
3 6 7
3 7 6
4 4 8
4 5 7
4 6 6
4 7 5
5 3 8
5 4 7
5 5 6
5 6 5
5 7 4
6 2 8
6 3 7
6 4 6
6 5 5
6 6 4
```

b) Modificați o singură instrucțiune din program astfel încât să fie afișate doar soluțiile în care spiridușul 1 și spiridușul 2 produc același număr de mașini. **(0.5 p.)**