

MODALITATEA DE DESFĂȘURARE A EXAMENULUI LA DISCIPLINA "PROGRAMAREA ALGORITMILOR"

- Examenul la disciplina "Programarea algoritmilor" se va desfășura în ziua de 19.01.2021, între orele 9³⁰ și 12⁰⁰, astfel:
 - 09³⁰ – 10⁰⁰: efectuarea prezenței studenților
 - 10⁰⁰ – 12⁰⁰: desfășurarea examenului
 - 12⁰⁰ – 12³⁰: 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 09³⁰ la ora 12³⁰, 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.
- 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 fiecare program Python se va preciza, pe scurt, sub forma unor comentarii, semnificația variabilelor utilizate.
- Pentru subiectele 1 nu contează complexitățile soluțiilor propuse.
- 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 pe platforma MS Teams folosind un anumit formular.
- Numele fișierului PDF trebuie să respecte șablonul *grupa_nume_prenume_subiect.pdf*. De exemplu, un student cu numele Popescu Ion Mihai din grupa 131 trebuie să denumească fișierul care conține rezolvarea primului subiect astfel: *131_Popescu_Ion_Mihai_1.pdf*.

Subiectul 1 – limbajul Python – 3 p.

a) Scrieți o funcție *frecvențe* care primește un număr variabil de liste formate din numere întregi și returnează un dicționar cu perechi de forma *frecvență: lista valori* conținând toate frecvențele distincte ale tuturor elementelor tuturor listelor, precum și câte o listă cu elementele care au frecvența respectivă. De exemplu, pentru apelul *frecvente*([20, 10, 40, 20], [10, 20, 10], [40, 30, 40]) funcția trebuie să furnizeze dicționarul {3: [20, 10, 40], 1: [30]}. **(1.5 p.)**

b) Înlocuiți punctele de suspensie din instrucțiunea *tupluri* = [...] cu o secvență de inițializare (*list comprehension*) astfel încât, după executarea sa, lista să conțină pentru fiecare număr cuprins între 1 și 20 cu proprietatea că restul împărțirii sale la 6 este diferit de câtul împărțirii sale la 6 câte un tuplu format din numărul respectiv, câtul și restul împărțirii sale la 6. **(0.5 p.)**

c) Considerăm următoarea funcție recursivă:

```
def f(v, p, u):
    if u == p:
        return v[u]
    else:
        m = (p+u)//2
        s1 = f(v, p, m)
        s2 = 0
        for i in range(m+1, u+1):
            s2 = s2 + v[i]
        return s1 + s2
```

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: $\mathcal{O}(n \log_2 n)$

Gigel tocmai a învățat la școală adunarea și înmulțirea numerelor întregi. Pentru a-l ajuta pe Gigel să-și fixeze cunoștințele proaspăt dobândite, precum și pentru a-i testa istețimea algoritmică, bunicul său a scris pe mai multe cartonașe numere întregi nenule și apoi le-a împărțit în două grămezi: o grămadă A formată din m cartonașe și o grămadă B formată din n cartonașe ($1 \leq m \leq n \leq 100000$). Sarcina lui Gigel este să selecteze din grămada B exact m cartonașe pe care apoi să le împerecheze, în orice ordine dorește el, cu cele m cartonașe din grămada A astfel încât prin însumarea produselor celor două numerele scrise pe fiecare pereche de cartonașe să obțină cea mai mare sumă posibilă. Scrieți un program Python care citește de la tastatură valorile scrise pe cartonașele din cele două grămezi și afișează pe ecran cea mai mare sumă pe care o poate obține Gigel respectând restricțiile indicate în enunțul problemei, precum și o modalitate de obținere a sa în forma indicată în exemplu.

Exemplu:

Dacă $A = [3, -2, 5, -1, 4]$ și $B = [7, 8, -5, 2, -4, -1, 5]$, atunci suma maximă pe care o poate obține Gigel este $97 = 3 * 5 + (-2) * (-5) + 5 * 8 + (-1) * (-4) + 4 * 7$.

Subiectul 3 – metoda Programării Dinamice (3 p.)

Complexitatea maximă a soluției: $O(nm)$

O pădure dreptunghiulară este codificată printr-o matrice cu n linii și m coloane cu elementele 0 și 1, în care celulele cu valoarea 0 sunt libere, iar pe cele cu valoarea 1 se află un copac. Pe prima linie se află un pădurar care vrea să ajungă pe ultima linie.

La fiecare pas el se poate deplasa într-una dintre celulele vecine cu cea pe care se află, în una dintre direcțiile SE, S, SV astfel:

- spre S dacă pe această poziție se află un copac
- spre SE dacă poziția este liberă
- spre SV dacă poziția este liberă

Când ajunge pe o poziție care conține un copac, pădurarul trebuie să îl taie.

Scrieți un program Python care citește de la tastatură coloana de pe care pornește pădurarul (prima coloană are numărul 1) și matricea care codifică pădurea și determină un traseu prin care pădurarul poate ajunge pe ultima linie tăind un număr minim de copaci. Se vor afișa numărul de copaci tăiați și coordonatele celulelor unui traseu optim (cu numerotarea liniilor și coloanelor de la 1), ca în exemplul următor. Dacă nu există un traseu pe care pădurarul să ajungă la ultima linie, se va afișa un mesaj corespunzător. Dacă există, determinați dacă traseul optim este unic și afișați un mesaj corespunzător.

Intrare de la tastatură	Ieșire pe ecran
4 0 0 0 0 1 0 1 1 0 1 0 1 0 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1	2 1 4 2 3 3 3 4 2 5 2 traseul optim este unic

Subiectul 4 – metoda Backtracking (3 p.)

a) După o lungă plimbare prin parc cu stăpâna sa, cățelușă Laika se află în fața unei mari provocări: trebuie să urce cele n trepte ($1 \leq n \leq 50$) până la ușa apartamentului în care locuiește. Din cauza oboselii, Laika poate să sară, de fiecare dată, peste un număr de trepte cuprins între 1 și t ($1 \leq t \leq n$). Fiind foarte curioasă, cățelușă Laika se gândește cum ar putea să afle toate modalitățile în care ar putea să urce cele n trepte. Ajutați-o pe Laika, scriindu-i un program Python care să citească de la tastatură numerele naturale n și t , după care să-i afișeze toate modalitățile corecte de urcare (evident, Laika știe să citească) și numărul acestora! **(2.5 p.)**

Exemplu:

Pentru $n = 5, t = 3$ există 13 modalități în care Laika poate urca treptele:

1, 1, 1, 1, 1

1, 1, 1, 2

1, 1, 2, 1

1, 1, 3

1, 2, 1, 1

1, 2, 2

1, 3, 1

2, 1, 1, 1

2, 1, 2

2, 2, 1

2, 3

3, 1, 1

3, 2

b) Precizați cum ar trebui modificată o singură instrucțiune din program astfel încât să afișeze doar soluțiile în care soluțiile în care Laika sare, de fiecare dată, peste un număr de trepte cel mult egal cu cel precedent. Pentru exemplul anterior, aceste soluții sunt cele scrise cu roșu. **(0.5 p.)**