

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 *litere* care primește un număr variabil de cuvinte formate doar din litere mici și returnează un dicționar care conține, pentru fiecare cuvânt primit ca parametru, o pereche de forma *cuvânt*: (*mulțime vocale*, *mulțime consoane*). De exemplu, pentru apelul *litere*("teste", "programare") funcția trebuie să furnizeze dicționarul {'teste': ({'e'}, {'t', 's'}), 'programare': ({'e', 'o', 'a'}, {'g', 'p', 'r', 'm'})}. **(1.5 p.)**

b) Înlocuiți punctele de suspensie din instrucțiunea *litere* = [...] cu o secvență de inițializare (*list comprehension*) astfel încât, după executarea sa, lista să conțină toate literele mici din alfabetul englez al căror index este divizibil cu 2 sau 3, dar nu este divizibil cu 6. Vom considera faptul că litera 'a' are indexul 1. **(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 = f(lista, p, p+k)  
    aux_2 = f(lista, p+k+1, p+2*k-1)  
    aux_3 = f(lista, p+2*k, u)  
    return 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: $\mathcal{O}(n \log_2 n)$

Considerăm n găleți pline cu apă ($1 \leq n \leq 10000$) cu proprietatea că în orice găleată, mai puțin în cea având capacitatea minimă, se găsește o cantitate de apă strict mai mare decât suma capacităților găleților mai mici. Oricare dintre cele n găleți poate fi deșertată într-un bazin cu capacitatea C , însă doar complet. Scrieți un program Python care să citească de la tastatură capacitățile celor n găleți și capacitatea C a bazinului (toate numere naturale nenule), după care să afișeze capacitățile găleților care ar trebui să fie deșertate în bazin pentru a-l umple complet sau un mesaj corespunzător dacă acest lucru nu este posibil.

Exemplu:

Considerăm 6 găleți având capacitățile [30, 6, 62, 5, 133, 14] și capacitatea bazinului $C = 153$. În acest caz, bazinul poate fi umplut complet deșertând în el gălețile cu capacitățile 14, 6 și 133. Observați faptul că toate capacitățile găleților au proprietatea specificată în enunț: $30 > 14 + 6 + 5$, $6 > 5$, $62 > 30 + 14 + 6 + 5$, $133 > 30 + 6 + 62 + 5 + 14$ și $14 > 6 + 5$!

Subiectul 3 – metoda Programării Dinamice (3 p.)

Complexitatea maximă a soluției: $O(nm)$

O regiune deșertică poate fi reprezentată printr-un tablou cu m linii și n coloane. Elementele tabloului reprezintă diferențele de nivel față de nivelul mării măsurate în metri. Un beduin trebuie să traverseze deșertul de la un punct dat din sud (un element dat de pe ultima linie) la nord (orice element de pe prima linie). La fiecare pas el se poate deplasa într-unul dintre elementele vecine cu cel care se află, în una dintre direcțiile NE, N, NV. Un traseu este considerat optimal dacă numărul total de metri urcați de beduin este minim. Scrieți un program Python care citește de la tastatură indicele coloanei de unde pleacă beduinul (cu numerotare de la 0; amintim că beduinul pornește de pe ultima linie) și matricea care reprezintă deșertul, după care determină o modalitate optimală de a traversa deșertul. Se vor afișa numărul minim de metri urcați și un traseu optim, cu liniile și coloanele numerotate de la 0. În plus, determinați dacă traseul optim este unic și afișați un mesaj corespunzător.

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

Subiectul 4 - metoda Backtracking (3 p.)

a) Un grup format din n excursioniști ($1 \leq n \leq 100$) au m corturi ($1 \leq m \leq 100$) cu capacitățile c_1, c_2, \dots, c_m astfel încât $c_1 + c_2 + \dots + c_m \geq n$. Capacitatea unui cort reprezintă numărul maxim de excursioniști care pot sta în cortul respectiv. Scrieți un program Python care să citească de la tastatură numerele naturale $n, m, c_1, c_2, \dots, c_m$ și afișează toate posibilitățile de a repartiza cei n excursioniști în cele m corturi astfel încât niciun cort să nu rămână gol, precum și numărul acestora. **(2.5 p.)**

Exemplu:

Pentru $n = 9, m = 3, c_1 = 5, c_2 = 2, c_3 = 4$ există 5 modalități corecte de repartizare a excursioniștilor în corturi:

3, 2, 4
4, 1, 4
4, 2, 3
5, 1, 3
5, 2, 2

b) Precizați cum ar trebui modificată o singură instrucțiune din program astfel încât să afișeze doar soluțiile în care primul cort este plin (se presupune faptul că acest lucru este posibil). Pentru exemplul anterior, aceste soluții sunt ultimele două. **(0.5 p.)**