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 9^{30} și 12^{00}, astfel:
  - 09^{30} – 10^{00}: efectuarea prezenței studenților
  - 10^{00} – 12^{00}: desfășurarea examenului
  - 12^{00} – 12^{30}: 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^{30} la ora 12^{30}, 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 *ștergere* care primește o valoare *x* și un număr variabil de liste formate din numere întregi și returnează o listă care conține, pentru fiecare listă primită ca parametru, câte un tuplu de forma (*numărul aparițiilor lui x în listă*, *lista rămasă după ștergerea tuturor aparițiilor lui x*). De exemplu, pentru apelul *stergere*(4, [2, 1, 4, 4], [-7, 3], [4, 4, 4]) funcția trebuie să furnizeze lista [(2, [2, 1]), (0, [-7, 3]), (3, [])]. **(1.5 p.)**

b) Înlocuiți punctele de suspensie din instrucțiunea *perechi* = [...] cu o secvență de inițializare (*list comprehension*) astfel încât, după executarea sa, lista să conțină toate tuplurile de forma (*a*, *b*) cu proprietatea că *a* și *b* sunt cifre nenule cu parități diferite. **(0.5 p.)**

c) Considerăm următoarea funcție recursivă:

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

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

O tablă de joc are de-a lungul ei, pe mijloc, un șanț cu poziții numerotate cu numere consecutive, începând de la 1 la 10000. Pe tablă sunt plasate, în șanț, bețișoare cu capătul stâng într-o anumită poziție dată din șanț. Bețișoarele au lungimi date. Un jucător trebuie să aleagă bețișoarele pe care trebuie să le elimine pentru a rămâne pe tablă doar bețișoare care nu se ating între ele. Se consideră că două bețișoare care ocupă spațiul din șanț între pozițiile p_1 și p_2, respectiv u_1 și u_2 se ating dacă intervalele $[p_1, p_2]$ și $[u_1, u_2]$ se intersectează. Un jucător câștigă dacă propune un număr minim de astfel de bețișoare care trebuie eliminate. Fiind date numărul n de bețișoare ($1 \leq n \leq 100000$) și pentru fiecare beţisor poziția capătului din stânga și lungimea lui, scrieți un program Python care să determine o mulțime de bețișoare care trebuie eliminate pentru a câștiga jocul (adică pe tablă să rămână un număr maxim de bețișoare care nu se ating între ele). Programul va afișa indicii bețișoarelor care trebuie eliminate, considerate ca fiind numerotate de la 1.

Exemplu:

Intrare de la tastatură	**Ieşire pe ecran**
5 3 70 1 9 1 19 11 20 40 40	1 3

Explicaţii:
După eliminarea bețișoarelor 1 și 3 rămân: bețișorul 2 (care ocupă pozițiile de la 1 la 10), bețișorul 4 (care ocupă pozițiile de la 11 la 31) și bețișorul 5 (care ocupă pozițiile de la 40 la 80). Cele 3 bețișoare rămase nu se ating și nu există 4 bețișoare care să nu se atingă!

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 line) 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 3 4 2 0 2 3 2 2 1 1 1 1 1 3 1	1 3 1 2 1 1 0 0 0 traseul optim nu este unic

Subiectul 4 – metoda Backtracking (3 p.)

a) După o lungă plimbare prin parc cu stăpâna sa, cățelușa 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, cățelușa 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.)**