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 **cmmdc** care primește un număr variabil de tupluri formate din câte două numere naturale nenule și returnează pentru fiecare tuplu primit ca parametru cel mai mare divizor comun al elementelor sale, sub forma unui dicționar cu perechi de tipul **cel mai mare divizor: lista tuplurilor care au cel mai mare divizor comun egal cu cheia**. De exemplu, pentru apelul **cmmdc((12, 50), (3, 4), (11, 13), (14, 2))** funcția trebuie să furnizeze dicționarul `{2: [(12, 50), (14, 2)], 1: [(3, 4), (11, 13)]}`. **(1.5 p.)**

b) Înlocuiți punctele de suspensie din instrucțiunea **numere = […]** cu o secvență de inițializare (*list comprehension*) astfel încât, după executarea sa, lista să conțină numerele naturale formate din exact două cifre care nu sunt pătrate perfecte și nici divizibile cu 7. **(0.5 p.)**

c) Considerăm următoarea funcție recursivă:

```
def f(lista):
    if len(lista) <= 2:
        return min(lista)
    k = len(lista) // 2
    aux_1 = lista[:k]
    aux_2 = lista[k+1:]
    return min(f(aux_1), f(aux_2), lista[k])
```

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

Subiectul 2 – metoda Greedy (3 p.)

Complexitatea maximă a soluției: $\mathcal{O}(n \log_2 n)$

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 greutățile 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 greutățile 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 8.5 21.25 12 6.05 20.7 23.8 22 33.25 21 48.15 62.20	3 3 + 2 10 + 4 1 + 8

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: $O(n^2)$

După o nouă plimbare lungă prin parc cu stăpâna sa, cățelușa Laika se află în fața unei mari provocări: trebuie iar să urce cele n trepte până la ușa apartamentului în care locuiește. De data aceasta ea mai are însă energie și poate să sară de pe o treaptă i direct pe una dintre treptele i+1, i+2, ..., n . Totuși pentru a sari *i* trepte trebuie sa plătească un cost c_i , pentru că face gălăgie. De asemenea, pentru că treptele au fost spălate și Laika vine după o joacă prin noroi, pentru fiecare treaptă *i* (i=1,...,n) pe care calcă Laika există un cost t_i (număr natural pozitiv) pe care trebuie să îl plătească. Ajutați-o pe Laika scriind un program Python care afișează o modalitate de a urca treptele de la baza scării (considerată treapta 0, pentru care taxa este $t_0 = 0$) la treapta *n* cu cost total minim. Se citesc de la tastatură n și taxele pentru cele n trepte $t_1, t_2, ..., t_n$ și costul pentru a sări treptele $c_1, c_2, ..., c_n$ (c_i este costul pe care trebuie să îl plătească dacă sare *i* trepte). Laika nu e un câine normal așa că s-ar putea costul să sară 3 trepte să fie mai mic decât să sară 2.

Intrare de la tastatură	Ieșire pe ecran
5 1 2 11 1 1 1 7 4 9 5	taxa totala 6 pentru traseul cu scările 0 5

Explicații: n=5, deci scara are 5 trepte numerotate 1,2, ... ,5 (pe lângă baza scării care se consideră treapta 0 și pentru care nu se plătește taxă); Laika preferă să sară din prima 5 trepte pentru că poate :), deci costul plătit va fi $c_5 + t_5 = 5 + 1 = 6$ (c_5 pentru că a sărit 5 trepte și t_5 pentru că a călcat pe treapta 5)

Intrare de la tastatură	Ieșire pe ecran
5 1 2 11 1 1 1 7 4 9 15	taxa totala 9 pentru traseul cu scările 0 1 4 5

Explicații: Laika sare la treapta 1 cu costul 2 (c_1=1 costul de a sări o treaptă, t_1=1 costul de a ajunge pe treapta 1), apoi Laika sare la treapta 4 cu costul 5 (c_3=4 costul de a sări 3 trepte, t_4=1 costul de călca pe treapta 4), apoi sare la treapta 5 cu costul 2 (1+1).

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.)**