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 **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) Știind că matricea pătratică **m** este memorată sub forma unei liste de liste, înlocuiți punctele de suspensie din instrucțiunea **numere = […]** cu o secvență de inițializare (*list comprehension*) astfel încât, după executarea sa, lista să conțină pătratul elementelor aflate pe diagonala principală a matricei **m**. De exemplu, pentru matricea `m = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]` trebuie ca lista **numere** să fie `[1, 25, 81]`. **(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)

Se organizează un concurs de tip trivia în limbajul Python la care participă nea Vasile împreună cu alți N oameni. Fiecărui om din cei N i se asociază o valoare strict pozitivă X_i, mai puțin lui Vasile căruia i se asociază implicit valoarea 0 (practic, Vasile este complet subestimat atât de către organizatori, cât și de ceilalți participanți). Concursul se desfășoară în mai multe runde eliminatorii, până când rămâne un singur om care este declarat câștigătorul. Într-o rundă la care participă K oameni (inclusiv Vasile!) sunt eliminați toți cei care nu știu răspunsul corect la întrebarea respectivă, iar scorul rundei se calculează ca fiind $\frac{\sum X_i}{K}$ pentru toți indicii i ai oamenilor eliminați în runda respectivă. Câștigătorul concursului va primi o sumă de bani egală cu suma scorurilor tuturor rundelor. Nea Vasile știe tot ceea ce s-ar putea ști despre limbajul Python (adică este imposibil ca el să piardă concursul!), deci îl interesează doar să afle suma maximă de bani pe care ar putea să o câștige.

Scrieți un program Python care să citească de la tastatură numărul natural nenul N, reprezentând numărul de concurenți (în afară de nea Vasile) și cele N numere strict pozitive asociate celor N participanți (separate între ele prin câte un spațiu), după care afișează un singur număr real (cu 3 zecimale) reprezentând suma maximă de bani pe care ar putea să o câștige nea Vasile.

Exemplu:

Date de intrare	Date de ieșire
2 6 6	5

Explicații: Suma maximă de bani pe care o poate câștiga Vasile este 5 = 2 + 3. În prima rundă ar trebui să iasă un singur adversar de-ai lui nea Vasile, scorul rundei fiind 6 / 3 = 2. În a doua rundă va ieși și ultimul adversar, scorul rundei fiind 6 / 2 = 3.

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) O țeavă cu lungimea de p metri ($1 \leq p \leq 50$) trebuie să fie tăiată în cel puțin două bucăți ale căror lungimi să fie divizori ai lungimii sale. De exemplu, o țeavă cu lungimea de 4 metri poate fi tăiată în 4 bucăți de câte 1 metru, 2 bucăți de câte 2 metri sau 2 bucăți de câte 1 metru și 1 bucată de 2 metri, dar nu poate fi tăiată într-o bucată de 1 metru și o bucată de 3 metri (deoarece 3 nu este un divizor al lui 4). Scrieți un program Python care să citească de la tastatură numărul natural p și afișează toate modalitățile distincte în care poate fi tăiată corect o bară de lungime p metri, precum și numărul acestora. Două modalități de tăiere se consideră identice dacă sunt formate din aceleași bucăți de țeavă, dar în altă ordine. De exemplu, pentru o țeavă cu lungimea de 4 metri, modalitățile de tăiere 1+1+2, 1+2+1 și 2+1+1 sunt considerate identice. **(2.5 p.)**

Exemplu:
Pentru $p = 6$ trebuie afișate următoarele 7 modalități de tăiere (nu neapărat în această ordine):

```
1+1+1+1+1+1
1+1+1+1+2
1+1+1+3
1+1+2+2
1+2+3
2+2+2
3+3
Nr. modalitati: 7
```

b) Precizați cum ar trebui adăugată o singură instrucțiune în program astfel încât să fie afișate doar modalitățile de tăiere în care au fost utilizate exact două tipuri distincte de bucăți de țeavă. Pentru exemplul anterior, aceste soluții sunt cele scrise cu roșu. **(0.5 p.)**