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 **pareimpare** care primește un număr variabil de liste formate doar din numere naturale și returnează un dicționar care conține, pentru fiecare listă primită ca parametru, o pereche de forma **indicele listei: (listă cu numere impare, listă cu numere pare)**. Indicele listei reprezintă poziția la care apare printre parametri (pe poziția 0 apare prima listă primită ca parametru, pe poziția 1 apare a doua listă primită ca parametru etc.). În cazul în care nu există numere pare sau impare în listă, se adaugă lista vidă. De exemplu, pentru apelul **pareimpare([1, 1, 2, 3, 4], [0, 2], [1, 2, 3])** funcția trebuie să furnizeze dicționarul `{0: ([1, 1, 3], [2, 4]), 1: ([], [0, 2]), 2: ([1, 3], [2])}`. **(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 trei cifre care nu sunt divizibile cu 5 și sunt palindromuri. **(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: $\mathcal{O}(n)$

Canalul Pythonic Way este unul foarte strâmt, astfel încât un singur nufăr încape pe lățimea sa. Pe canal sunt înșiruiți, de la capătul din stânga spre capătul din dreapta, mai mulți nuferi, iar pe fiecare nufăr este scris un număr natural nenul reprezentând numărul maxim de nuferi peste care poate să sară o broscuță aflată pe nufărul respectiv. Astfel, dacă pe nufărul cu numărul de ordine i este scris numărul k, atunci o broscuță poate să sară pe oricare dintre nuferii cu numerele de ordine $i+1, i+2, \ldots, i+k$. Primul nufăr, având numărul de ordine 1, se află la capătul din stânga al canalului, iar ultimul nufăr, având numărul de ordine n, se află la capătul din dreapta. Într-o bună zi, Lily, una dintre broscuțele care trăiesc în Pythonic Way, s-a hotărât să iasă din lumea strâmtă a canalului și să plece în lumea largă. Deoarece Lily are o fire melancolică, ea vrea să plece de pe primul nufăr, să ajungă pe ultimul nufăr (pentru a mai vedea încă o dată întreg canalul Pythonic Way) și abia apoi să iasă din canal (fără să mai facă niciun salt). Pentru a nu avea timp să se răzgândească, Lily vrea să ajungă pe ultimul nufăr cât mai repede, adică folosind un număr minim de sărituri care respectă restricția precizată anterior. Scrieți un program în limbajul Python care să citească de la tastatură numerele scrise pe cei n nuferi și să afișeze pe ecran un traseu format din numerele de ordine ale nuferilor pe care trebuie să sară Lily pentru a ieși din canal plecând de pe primul nufăr și efectuând un număr minim de sărituri. Dacă există mai multe trasee cu proprietatea cerută, atunci se va scrie oricare dintre ele. Numerele de ordine ale nuferilor din traseu vor fi despărțite între ele prin câte un spațiu. Fiecare săritură efectuată de Lily trebuie să respecte restricția precizată în enunț.

Exemplu:

Date de intrare	Date de ieșire
2 3 1 5 3 2 2 5	1 2 4 8

Explicații: Lily trebuie să efectueze cel puțin 3 sărituri, plecând de pe primul nufăr, pentru a ieși din canalul Pythonic Way. Un traseu corect pe care îl poate urma Lily este 1, 2, 4, 8. Un alt traseu corect este 1, 3, 4, 8.

Subiectul 3 – metoda Programării Dinamice (3 p.)
Complexitatea maximă a soluției: O(nM)

Pia a hotărât să aloce *M* minute în care să facă doar activitățile ei preferate. Și-a făcut o listă cu *n* activități pe care le-a numerotat *1*,...,*n* și a estimat pentru fiecare activitate *i* =1,...,n durata d_i în minute. Ea ar vrea să își ocupe cât mai mult timp cu activitățile preferate și ar vrea să aleagă ce activități va face astfel încât timpul total dedicat activităților alese (egal cu suma duratelor lor) să fie cât mai apropiat de *M* (poate și depăși M, dar diferența între suma duratelor activităților alese și *M* trebuie să fie cât mai mică). Scrieți un program Python care să citească de la tastatură numărul de minute *M*, numărul de activități *n* și duratele d_1, ... , d_n și afișează ce activități să facă Pia astfel încât durata totală a acestora să fie cât mai apropiată de M.

Intrare de la tastatură	Ieșire pe ecran
21 6 5 10 5 4 10 3	1 3 5

Explicații: suma duratelor activităților 1, 3 și 5 este 5+5+10 = 20 și nu există o mulțime de activități cu suma duratelor 21. Soluția optimă nu este unică, o altă soluție optimă este de exemplu cea formată cu activitățile 2, 5 sau 3, 4, 5, 6 (în acest ultim caz durata totală este 22, cu 1 mai mare decât M, deci la fel de apropiată de M ca și 20)

Intrare de la tastatură	Ieșire pe ecran
20 4 10 11 4 12	1 2

Explicații: suma duratelor activităților 1, 2 este 21 și nu există o mulțime de activități cu suma duratelor 20 (deci cea mai apropiată durată totală de M=20 pe care o putem obține este 21)

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.)**