

## 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<sup>00</sup> și 11<sup>30</sup>, astfel:
  - 09<sup>00</sup> – 09<sup>30</sup>: efectuarea prezenței studenților
  - 09<sup>30</sup> – 11<sup>30</sup>: desfășurarea examenului
  - 11<sup>30</sup> – 12<sup>00</sup>: 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<sup>00</sup> la ora 12<sup>00</sup>, 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) Unei liste formată din numere de la 1 la 26 i se asociază în mod unic un cuvânt astfel: numărul 1 este înlocuit de litera 'a', numărul 2 este înlocuit de litera 'b', ..., numărul 26 este înlocuit de litera 'z'. De exemplu, pentru lista [5, 23, 1, 13, 5, 14], cuvântul asociat este 'examen'. Să se scrie o funcție **cuvinte** care primește un număr variabil de liste formate din numere de la 1 la 26 și returnează un dicționar care conține, pentru fiecare listă, cuvântul asociat acesteia. De exemplu, pentru apelul **cuvinte([1, 14, 1], [1, 18, 5], [13, 5, 18, 5])**, funcția returnează dicționarul {'ana': [1, 14, 1], 'are': [1, 18, 5], 'mere': [13, 5, 18, 5]}. **(1.5 p.)**

b) Înlocuiți punctele de suspensie din instrucțiunea `lung = [...]` cu o secvență de inițializare (*list comprehension*) astfel încât, după executarea sa, lista să conțină toate tuplurile de forma (cuvânt, lungime) pentru fiecare cuvânt care începe cu o vocală dintr-o propoziție dată **p**. Propoziția este formată doar din litere mici ale alfabetului englez, iar cuvintele care o formează sunt distincte și despărțite între ele prin câte un spațiu. De exemplu, pentru propoziția `p = 'un exemplu de propozitie'`, lista rezultată va fi `lung = [('un', 2), ('exemplu', 7)]`. **(0.5 p.)**

c) Considerăm următoarea funcție recursivă:

```
def f(lista, p, u):
    if u - p <= 1:
        return lista[0]
    k = (p + u) // 2
    if k % 2 == 1:
        return f(lista, p, k-1) + f(lista, k+1, u)
    else:
        return f(lista, p, k-2) + f(lista, k+1, u)
```

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:  $O(n \log_2 n)$

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

**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  $i+1, i+2, \dots, n$ . Totuși pentru a sari  $i$  trepte trebuie să 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  $i$  ( $i=1, \dots, n$ ) 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  $t_1, t_2, \dots, t_n$  și costul pentru a sări treptele  $c_1, c_2, \dots, c_n$  ( $c_i$  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) Petrișor ar vrea să își schimbe parolele și are o mulțime de litere preferate **L** și o mulțime **S** de simboluri (caractere care nu sunt litere) pe care ar vrea să le folosească în parole pentru a crește siguranța acestora. Pentru a îi fi mai ușor să le țină evidența, el ar vrea să își construiască parole de aceeași lungime după un anumit tipar. Tiparul este un șir de caractere de lungime **n** format doar cu caracterele '**l**' și '**s**' cu semnificația: dacă în tipar pe poziția **i** este caracterul '**l**', atunci în parolă pe poziția **i** va fi o literă din mulțimea **L**, iar dacă în tipar pe poziția **i** este caracterul '**s**', atunci în parolă pe poziția **i** va fi un simbol din mulțimea **S**. Mai mult, Petrișor și-ar dori ca orice simbol din **S** și orice literă din **L** să apară cel mult o dată în parolă. Scrieți un program Python care să citească de la tastatură numărul **n**, tiparul **T** și mulțimile **L** și **S**, după afișează toate parolele care verifică cerințele lui Petrișor sau mesajul "*Imposibil*" dacă nu există nicio parolă având proprietățile cerute. (2.5 p.)

**Exemplu:** Pentru **n = 6**, tiparul '**lslsll**', mulțimea **L** de litere '**a**', '**b**', '**c**', '**D**' și mulțimea **S** de simboluri '@', '.' trebuie afișate următoarele 48 de parole (nu neapărat în această ordine):

a@b.cD	b@D.ca	c.D@ab
a@b.Dc	b.a@cD	c.D@ba
a@c.bD	b.a@Dc	D@a.bc
a@c.Db	b.c@aD	D@a.cb
a@D.bc	b.c@Da	D@b.ac
a@D.cb	b.D@ac	D@b.ca
a.b@cD	b.D@ca	D@c.ab
a.b@cD	c@a.bD	D@c.ba
a.c@bD	c@a.Db	D.a@bc
a.c@Db	c@b.aD	D.a@cb
a.D@bc	c@b.Da	D.b@ac
a.D@cb	c@D.ab	D.b@ca
b@a.cD	c@D.ba	D.c@ab
b@a.Dc	c.a@bD	D.c@ba
b@c.aD	c.a@Db	
b@c.Da	c.b@aD	
b@D.ac	c.b@Da	

b) Modificați o singură instrucțiune din program astfel încât să fie afișate doar parolele care încep cu o vocală. Pentru exemplul anterior, aceste soluții sunt cele scrise cu roșu. (0.5 p.)