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) Scrieţi o funcţie **`litere`** care primeşte un număr variabil de cuvinte formate din litere mici ale alfabetului englez şi returnează un dicţionar care conţine pentru fiecare cuvânt primit ca parametru, un dicţionar cu frecvenţa fiecărei litere distincte care apare în cuvânt. De exemplu, pentru apelul **`litere('teste', 'dictionar', 'ele')`** funcţia trebuie să returneze dicţionarul `{'teste': {'e': 2, 's': 1, 't': 2}, 'dictionar': {'a': 1, 'c': 1, 'd': 1, 'i': 2, 'n': 1, 'o': 1, 'r': 1, 't': 1}, 'ele': {'e': 2, 'l': 1}}`. **(1.5 p.)**

b) Folosind un dicţionar cu acelaşi format ca valorile dicţionarului de la punctul a) (i.e., cheile sunt litere, iar valorile frecvenţa literei respective), să se scrie o secvenţă de iniţializare (*list comprehension*) pentru o listă astfel încât aceasta să conţină perechile de forma **`(literă, frecvență)`** cu literele extrase din dicţionar care au frecvenţa pară. De exemplu, pentru dicţionarul `{'e': 2, 's': 1, 't': 2}` lista trebuie să fie `[('e', 2), ('t', 2)]`. **(0.5 p.)**

c) Considerăm următoarea funcţie recursivă:

```
def f(lista, p, u):
    if u-p <= 1:
        return sum(lista[p: u+1])
    k = (u-p+1) // 3
    aux_1 = f(lista, p, p+k)
    aux_2 = f(lista, p+k+1, p+2*k)
    aux_3 = f(lista, p+2*k+1, u)
    return aux_1 + aux_2 + aux_3
```

Determinaţi complexitatea funcţiei 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: $\mathcal{O}(n \log_2 n)$

La ora de sport, profesorul vrea să execute exerciții de gimnastică cu grupe de câte 2 elevi, dar pentru a putea realiza acest lucru trebuie ca valoarea absolută a diferenței dintre înălțimile celor 2 elevi dintr-o grupă să fie strict mai mică decât un număr natural *h*. Scrieți un program Python care citește de la tastatură două numere naturale *n* și *h*, precum și numele și înălțimile a *n* elevi, după care afișează pe ecran, în forma indicată în exemplu, numărul maxim de grupe formate din câte 2 elevi care se pot realiza respectând condiția indicată anterior, precum și numele elevilor din grupele respective. Evident, un elev poate să facă parte din cel mult o grupă! Înălțimile tuturor elevilor și diferența *h* sunt exprimate în centimetri. Nu contează ordinea în care se vor afișa grupele de elevi și nici ordinea numelor elevilor dintr-o grupă.

Exemplu:

Date de intrare	Date de ieșire
8 10 Popescu Ion 172 Mihai Ana 162 Popescu Dana 190 Ionescu Ion 181 Georgescu Ioana 170 Dumitrescu George 188 Constantinescu Radu 165 Georgescu Anca 210	3 Popescu Ion, Georgescu Ioana Mihai Ana, Constantinescu Radu Ionescu Ion, Dumitrescu George

Explicații: Avem *n* = 8 și *h* = 10. Se pot forma maxim 3 grupe de câte 2 elevi cu proprietatea că valoarea absolută a diferenței dintre înălțimile lor este strict mai mică decât 10 centimetri. Soluția nu este unică, o altă soluție corectă obținându-se, de exemplu, înlocuind grupa *Ionescu Ion, Dumitrescu George* cu grupa *Ionescu Ion, Popescu Dana*.

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) Petrişor ar vrea să îşi schimbe parolele şi are o mulţime de litere preferate **L** şi o mulţime **S** de simboluri (caractere care nu sunt litere) pe care ar vrea să le folosească în parole pentru a creşte siguranţa acestora . Pentru a îi fi mai uşor să le ţină evidenţa, el ar vrea să îşi construiască parole de aceeaşi lungime după un anumit tipar. Tiparul este un şir de caractere de lungime **n** format doar cu caracterele **'l'** şi **'s'** cu semnificaţia: dacă în tipar pe poziţia **i** este caracterul **'l'**, atunci în parolă pe poziţia **i** va fi o literă din mulţimea **L**, iar dacă în tipar pe poziţia **i** este caracterul **'s'**, atunci în parolă pe poziţia **i** va fi un simbol din mulţimea **S**. Mai mult, Petrişor şi-ar dori ca orice simbol din **S** şi orice literă din **L** să apară cel mult o dată în parolă. Scrieţi un program Python care să citească de la tastatură numărul **n**, tiparul **T** şi mulţimile **L** şi **S**, după afişează toate parolele care verifică cerinţele lui Petrişor sau mesajul "*Imposibil*" dacă nu există nicio parolă având proprietăţile cerute. **(2.5 p.)**

Exemplu: Pentru **n = 6**, tiparul **'lslsll'**, mulţimea **L** de litere **'a', 'b', 'c', 'D'** şi mulţimea **S** de simboluri **'@','.'** trebuie afişate 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@Dc	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) Modificaţi o singură instrucţiune din program astfel încât să fie afişate doar parolele care încep cu o vocală. Pentru exemplul anterior, aceste soluţii sunt cele scrise cu roşu. **(0.5 p.)**