Programarea Algoritmilor
– LABORATOR NR. 3 –

Liste, tupluri, seturi, dicționare; Fișiere text

1. Să se determine cele mai mari două valori distincte dintr-un șir de numere întregi.

2. Să se determine cuvântul care apare cel mai des într-o propoziție dată, precum și cuvântul care apare cel mai rar. Dacă sunt mai multe posibilități, se vor afișa cuvintele cele mai mici din punct de vedere lexicografic.

3. Folosind un dicționar, să se numere de câte ori apare fiecare caracter într-un text de tip "lorem ipsum". Folosiți generatorul de text: (https://loremipsum.io/).

4. Să se unifice două dicționare în care cheile sunt șiruri de caractere, iar valorile sunt de tip numeric. Astfel, în rezultat va apărea fiecare cheie distinctă, iar pentru o cheie care apare în ambele dicționare inițiale valoarea corespunzătoare va fi egală cu suma valorilor asociate cheii respective în cele două dicționare.

5. Scrieți un program care să furnizeze toate cuvintele dintr-un șir de caractere formate din aceleași litere cu ale unui cuvânt dat (fără a fi neapărat anagrame!). Dacă șirul nu conține nici un cuvânt cu proprietatea cerută, programul trebuie să afișeze un mesaj corespunzător. Cuvintele din șir sunt despărțite între ele prin spații și semnele de punctuație uzuale. De exemplu, pentru șirul “Langa o cabana, stand pe o banca, un bacan a spus un banc bun.” și cuvântul “bacan” funcția trebuie să furnizeze cuvintele “cabana”, “banca”, “bacan” și “banc”.

6. Grupuri de cuvinte care rimează

   Se citește un text format din cuvinte de lungime minim 3, despărțite prin spațiu. Să se creeze un dicționar “rime” care are ca chei sufixe de lungime 2 ale cuvintelor din text, iar ca valori liste cu cuvintele din text care au ca acel sufix. Apoi să se creeze un alt dicționar “final”, care preia din dicționarul “rime” doar acele perechi pentru care lista conține cel puțin 2 cuvinte pentru respectivul sufix.

7. Total cheltuieli

   De la tastatură se citește numele unui fișier. Acel fișier conține un text în care sunt specificate cheltuielile efectuate de Ana într-o zi. Scrieți un program care să afișeze suma totală cheltuită de Ana în ziua respectivă.

   Exemplu: fișierul “cheltuieli.txt” conține textul:
   “Azi am cumpărat 5.5 kg de mere cu 2.2 RON kilogramul și 3 pâini a câte 5 RON fiecare. ”

   Observații:
   - Se știe că există un număr par de numere întregi/reale în text, fiecare pereche reprezentând cantitatea și prețul (nu neapărat în această ordine) pentru un produs.
   - Orice număr este între două spații. După ce împărțiți textul în cuvinte, pentru a verifica dacă un cuvânt `cuv` este un număr întreg folosiți metoda `cuv.isdecimal()`, care returnează True dacă toate caracterele din `cuv` sunt cifre în baza 10. Pentru a verifica dacă cuvântul este un număr real, spargeți (metoda “split”) cuvântul după “.”, iar dacă obțineți *două* părți, verificați pentru fiecare dacă `isdecimal()`.

8. Magazine și produse

Fișierul text “inventar.txt” conține pe fiecare rând, cu spațiu între ele, un șir de litere reprezentând numele unui magazin, urmat de numere naturale reprezentând codurile produselor aflate în stoc la acel magazin.

a) Stocați informațiile citite din fișier într-un dicționar având ca chei numele magazinelor și ca valori un set cu codurile produselor din acel magazin.

b) Afișați codurile acelor produse care există în stocul tuturor magazinelor *(intersecție de seturi).*

c) Afișați codurile tuturor produselor *(reuniune de seturi).*

d) Afișați pentru fiecare magazin: numele magazinului și codurile produselor exclusiviste (produse care se află doar în acel magazin, nu și în altele) *(diferență de seturi)*.

9. Definiții

Un student vrea, pentru ora de engleză de la FMI, să găsească cele mai expresive cuvinte. Pentru aceasta a făcut o poză la astfel de cuvinte dintr-un dicționar, iar apoi a folosit un program ca să extragă un string cu textul din poze. Textul respecta regulile:

- Este împărțit în paragrafe, fiecare paragraf terminându-se cu "\n".
- Fiecare paragraf corespunde unei intrări din dicționar, adică conține un cuvânt și definițiile pentru acesta
- Cuvântul unui paragraf este la începutul lui și este mereu urmat de ":" și apoi de toate definițiile cuvântului

Acum, studentul vrea sa decidă care sunt cele mai expresive cuvinte. Pentru asta, numără pentru fiecare cuvânt în câte expresii apare (Ex: **run** away, **run** over, **run** a company), în acesta mod:

- Numără de câte ori apare cuvântul în paragraf
- Numără de câte ori apare caracterul tilda (~) în paragraf
- Și însumează aceste două numere

- Se dă stringul complet, format din paragrafe și dat ca input pe un singur rând.
- Se cere să se obțină o listă de tupluri formate dintr-un cuvânt aflat la început de paragraf și numărul care reprezintă expresivitatea sa calculată în acel paragraf.

De exemplu, pentru următorul string:

"run: to go faster than a walk : to go steadily by springing steps : to take part into a contest - **~** a marathon : to move at a fast gallop - he may occasionally **run** to and from work : flee, retreat, escape - drop the gun and **run** : to go without restraint : move freely about at will - let chickens **~** loose : consort - we **run** with our group \n" +
"dog: canid wolves, foxes, and other dogs especially : a highly variable domestic mammal : a pet **~** : fellow, chap, a lazy person - you lucky **dog** \n" +
"break: **break** a/the record to do something better than the best known speed, time, number, etc. previously achieved : to fail to keep a law, rule, or promise = **~** the law : These enzymes **break** down food in the stomach (= cause food to separate into smaller pieces). I needed something to **break** the monotony of my typing job. The phone rang, as to **break** my concentration. To **~** (of a storm) = to start suddenly: We arrived just as a storm was breaking. \n"

Se obține:
[("run", 5), ("dog", 2), ("break", 6)]
de exemplu, pentru "run" numărăm de două ori pe **~** și de 3 ori pe **run,** deci expresivitatea este 5, iar tuplul este ("run", 5)

10. Departajare

La facultate se ține un concurs de programare. Organizatorii au aranjat ca punctajele să fie calculate automat, din codul scris. Totuși, ei nu au implementat vreo metodă de a departaja punctajele egale, așa că pe ultimul moment s-au decis ca persoana care a trimis codul prima să fie clasată mai sus decât cel de-al doilea ca timp, dar cu punctaj identic șamd.

Până la introducerea lui "-1", să se citească de la tastatură perechi de două elemente: (punctaj, nume_student), cu punctajul între 0 și 100. Perechile se consideră a fi introduse în ordinea predării codului de către participanți.

a) Salvați inputul într-o listă de tupluri (punctaj, nume_student, nr_ordine), astfel încât să se poată deduce și numărul de ordine pentru fiecare participant (al câtelea a predat codul).

b) Să se afișeze lista tuturor punctajelor distincte obținute de participanți (folosiți un set).

c) Într-un dicționar, pentru fiecare punctaj să se asocieze o listă de tupluri ce conțin numele participantului și numărul său de ordine. Apoi, să se afișeze clasamentul concursului (descrescător după punctaje, pe câte un rând: punctaj, nume_participant și nr_ordine).

De exemplu, pentru următorul input:

```
64 Danil Marius
70 Derek Alexandru
100 Pirpiric Claudiu
18 Alexandrescu Matias
64 Popescu Catalin
100 Cozia Daniel
82 Stefan Dinca
-1
```

Vom obtine:

a) Lista [(64, Danil Marius, 1), (70, Derek Alexandru, 2), (100, Pirpiric Claudiu, 3), (18, Alexandrescu Matias, 4), (64, Popescu Catalin, 5), (100, Cozia Daniel, 6), (82, Stefan Dinca, 7)]

b) Mulțimea {100, 82, 70, 64, 18} (nu neapărat în această ordine)

c) Și dicționarul (având perechile nu neapărat în această ordine):

```
{
  100: [("Pirpiric Claudiu", 3), ("Cozia Daniel", 6)],
   82: [("Dinca Stefan", 7)],
   70: [("Derek Alexandru", 2)],
   64: [("Danil Marius", 1), ("Popescu Catalin", 5)],
   18: [("Alexandrescu Matias", 4)]
}
```

11. Se citește un număr natural N.

a) Să se genereze și afișeze o matrice de dimensiune NxN, cu elementele de la 1 la N*N - în ordine crescătoare, de la stânga la dreapta și de sus în jos.

b) Pentru a parcurge elementele matricei în spirală, pornind din colțul din stânga-sus (spre dreapta, în jos, spre stânga, în sus, …), să se obțină întâi o listă având elemente de tip tuplu (linie, coloană) care să reprezinte pozițiile ce trebuie parcurse în această spirală.

c) Folosind lista de tupluri de mai sus, să se afișeze elementele din matrice aflate la acele poziții.

L\C	0	1	2	3	4
0	1	2	3	4	5
1	6	7	8	9	10
2	11	12	13	14	15
3	16	17	18	19	20
4	21	22	23	24	25

```
lista_poz = [(0, 0), (0, 1), ..., (0, N-2),
             (0, N-1), (1, N-1), ..., (N-2, N-1),
             (N-1, N-1), (N-1, N-2), ..., (N-1, 1),
             (N-1, 0), (N-2, 0), ..., (1, 0),
             (1, 1), (1, 2), ...]
```

Pentru N = 5:

```
spirala = [1, 2, 3, 4, 5, 10, 15, 20, 25, 24, 23, 22,
21, 16, 11, 6, 7, 8, 9, 14, 19, 18, 17, 12, 13]
```

12. Se citește din fișierul "graf_in.txt" un graf (ne)orientat astfel:

- cuvântul "orientat" sau "neorientat"
- n - numărul de noduri (cu nodurile indexate de la 1 la n)
- m - numărul de arce (muchii)
- m perechi de forma (xi,yi) reprezentând lista de arce (muchii)
- două noduri s și f (start și final)

Pentru toate cerințele faceți afișările într-un fișier "graf_out.txt".
Nu uitați să închideți fișierele la final!

a) Să se afișeze *lista de arce (muchii)*: o listă de tupluri cu elemente (i,j) pentru muchie de la nodul i la nodul j, pt. graf orientat, respectiv pentru muchie între nodurile i și j, pt. graf neorientat.

b) Să se genereze *listele de adiacență* pentru graf: folosiți un dicționar D în care cheia este nodul curent i, iar valoarea este o listă de noduri j, adiacente cu nodul curent (există muchie de la nodul i la nodul j, pt. graf orientat, respectiv există muchie între nodurile i și j, pt graf neorientat).

c) Să se genereze *matricea de adiacență* a grafului: o matrice M de dimensiune n*n, în care M[i][j] = 1 dacă există muchia de la nodul i la nodul j (pt. graf orientat) sau M[i][j] = M[j][i] = 1 dacă există muchie între nodurile i și j (pt. graf neorientat), și valoarea 0 în restul matricei.

d) Să se obțină *parcurgerea în lățime (breadth first)* pornind din nodul s.
1) Luați o listă BF, inițial vidă (`BF=[]`), care se va comporta ca o *coadă* și va avea ca elemente tupluri (nod, tata_nod). Nu ștergeți elementele introduse în lista BF, dar aveți două variabile st(ânga) și dr(eapta) care vor reține poziția curentă a primului, respectiv ultimului element al cozii. Spunem despre nodurile aflate în lista BF că sunt "vizitate" deja.
2) Adăugați primul element (tuplul format din nodul s și -1 pt. tatăl lui inexistent) în coadă (adică la finalul listei BF): `BF.append((s,-1)); st=dr=0`.
3) Cât timp coada nu este vidă (`while st<=dr:`) executați:
-- pentru primul nod din coadă (`tata=BF[st][0]`), găsiți toți vecinii (din dicționarul obținut la punctul b) (`for fiul in D[tata]:`) nevizitați (care nu se află în lista BF) (`if fiul not in [nod for (nod, parinte) in BF]:`) și îi introduceți la finalul cozii împreună cu părintele lor (`BF.append((fiul,tata)); dr+=1`); apoi "scoateți" primul nod din coadă (`st+=1`).
4) Afișați parcurgerea în lățime: doar primul element al fiecărui tuplu din lista BF.

e) Să se obțină *parcurgerea în adâncime (depth first)* pornind din nodul s.
1) Luați o listă DF, initial vidă (`DF=[]`), care se va comporta ca o *stivă* și va avea ca elemente noduri ale grafului; și o listă "vizitat", inițial vidă (`vizitat=[]`), în care vom adăuga nodurile pe măsură ce sunt vizitate în cadrul parcurgerii în adâncime.
2) Adăugați în stivă primul element (nodul s) și îl marcăm ca fiind vizitat (`DF.append(s); vizitat.append(s)`).

3) Cât timp stiva nu este vidă (`while DF!=[]:`) executați:
-- pentru nodul din vârful stivei (ultimul element al listei DF) (`tata=DF[-1]`) se caută în dicționarul D un vecin (`for fiul in D[tata]:`) nevizitat (`if fiul not vizitat: break`).
i) dacă acest fiu există, atunci îl adăugăm în vârful stivei (la finalul listei DF) și îl vizităm (`DF.append(fiul); vizitat.append(fiul)`), apoi reluăm pasul 3).
ii) dacă acest fiu nu există (nodul nu are fii sau toți fiii sunt deja vizitați), atunci scoatem nodul din vârful stivei (de la finalul listei DF) (`DF.pop(-1)`), apoi reluăm pasul 3).
4) Afișați parcurgerea în adâncime: lista "vizitat".

f) Să se găsească drumul de lungime minimă între nodurile s și f, dacă acesta există. În caz că nu există drum, să se afișeze un mesaj corespunzător. Folosind lista BF (obținută la punctul d), obțineți nodurile care formează drumul, plecând din nodul f și trecând din tată în tată până ajungeți la nodul s (atenție la ordinea în care afișați nodurile drumului, trebuie de la s către f).