Tema 3 - Tabele de dispersie

1. **Implementare tabelă de dispersie - liste înlănţuite**. Construiţi o clasă HashTable (sau HashMap) potrivită, care să includă operaţiile de inserţie, căutare şi ştergere. Elementele stocate vor fi de tip (cheie, valoare). Folosiţi pair din stl. Rezolvarea coliziunilor se va realiza prin liste înlănţuite (folosiţi **std::list**). Dacă factorul de încarcare al tabelei depăşeşte 1.0, se cere redimensionarea tabelei (aproximativ dublul dimensiunii iniţiale) şi redistribuirea elementelor în noua tabelă (*rehashing*). În funcţia main citiţi dintr-un fişier n elemente de tip pereche (cheie-valoare) (n>20), repartizaţi elementele în tabelă, apoi permiteţi căutarea, adăugarea sau ştergerea de elemente (meniu). De asemenea permiteţi parcurgerea şi afiąsrea perechilor $< cheie, valoare >$ pentru toate elementele din tabelă. (3p).

   Punctaj suplimentar - pentru implementarea unei funcţii de hashig pentru şiruri de caractere - 0.5p

2. **Implementare tabelă de dispersie - adresare deschisă**. Construiţi o clasă HashTable (sau HashMap) potrivită, care să includă operaţiile de inserţie, căutare şi ştergere. Elementele stocate vor fi de tip (cheie, valoare). Folosiţi pair din stl. Rezolvarea coliziunilor se va realiza prin dublă repartizare. În cazul dublei repartizări, dacă factorul de încarcare al tabelei depăşeşte 0.7, se cere redimensionarea tabelei (aproximativ dublul dimensiunii iniţiale) şi redistribuirea elementelor în noua tabelă (*rehashing*). În funcţia main citiţi dintr-un fişier n elemente de tip pereche (cheie-valoare) (n>20), repartizaţi elementele în tabelă, apoi permiteţi căutarea, adăugarea sau ştergerea de elemente (meniu). De asemenea permiteţi parcurgerea şi afiąsrea perechilor $< cheie, valoare >$ pentru toate elementele din tabelă. (3p).

   Punctaj suplimentar - pentru implementarea unei funcţii de hashig pentru şiruri de caractere - 0.5p

3. **Permutări**. Se consideră două şiruri de caractere (citite din fişier). Să se scrie o funcţie care are ca parametru cele două şiruri şi care returnează *true* dacă al doilea este o permutare a primului şi *false* altfel. Implementaţi folosind **unordered_set** din stl. (1p)

4. **Concurenţi**. Se consideră un număr de competiţii sportive la care s-au inscris concurenţi. Pentru fiecare competiţie există o listă cu numele şi prenumele concurenţilor (pereche de valori de tip std::string). Aceste liste se citesc dintr-un fişier. Să se scrie o funcţie care indică persoanele care participă la mai mult de o singură competiţie. (1p)

5. **Subşiruri de suma dată**. Scrieţi un program care citeşte Nr elemente numere naturale dintr-un fişier şi le plasează într-un vector $Numere$. Având acest vector, se permit oircâte citiri a unei valori k (într-un do-while). Pentru fiecare k se afişază toate perechile de indici $(start, stop)$ $(start \leq stop)$ pentru care subşirul $Numere[start] + Numere[start + 1] + \ldots + Numere[stop] = k$. Rezolvati problema n̂ mod efficient, folosind $unordered\_map$. (1.5p)

6. **Sumă nulă**: Se consideră 4 vectori de numere A, B, C şi D de lungime cel mult 500 fiecare. Să se determine toate cvadruplurile de indici (i, j, k, l) pentru care $A[i] + B[j] + C[k] + D[l] = 0$. Ce complexitate are algoritmul dvs? Care este cea mai bună complexitate pe care o puteţi obţine? (1.5p)

7. **Duplicate apropiate**. Se citeşte dintr-un fişier un număr de valori reale şi se stochează într-un vector. Să se determine în mod eficient, dacă există două numere egale în vector, aflate la o distanţă mai mică sau egală cu o valoare dată $dist$. Puteţi folosi **unordered_set** sau altă structură care permite rezolvare eficientă.(1p)

8. **Magazine** Se consideră nr_mag magazine. Fiecare are un număr de produse. Să se veifice care magazin are cele mai multe produse exclusive (nu apar decât în magazinul respectiv). Citiţi dintr-un fişier în câte un vector de **std::string** produsele pentru fiecare magazin. Afişaţi în final magazinul cu cele mai multe produse exclusive şi care sunt aceste produse. (1.5p)

9. **Anagrame**. Se consideră un şir de cuvinte citite dintr-un fişier. Scrieţi o funcţie, care să grupeze anagramele. Se consideră anagramă un cuvânt obţinut prin rearanjarea literelor altui cuvânt. Folosiţi structurile de date din stl învăţate, aşa încât să obţineţi eficienţa cea mai bună. (1.5p) (punctaj în funcţie de rezolvare)
   Exemplu: Se consideră cuvintele {car, rac, cos, amin, arc, soc, polca, lac, cal, pocal,mina, copil, anim}. Atunci se vor grupa: {car, rac, arc}, {cos, soc}, {amin, mina, anim}, {lac, cal}, {pocal, polca}, {copil}.

10. **Aruncarea zarurilor** Se consideră 3 zaruri care sunt aruncate de un număr N de ori. Practic pentru fiecare aruncare se generează 3 numere aleatoare în mulţimea $\{1, \ldots, 6\}$. Pentru fiecare triplet de numere (n_1, n_2, n_3), $n_i \in \overline{1,6}$

citit de la tastatură să se indice, de câte ori a fost aruncat tripletul. Atentie permutari ale aceluiaşi triplet nu trebuie considerate separat. (2p)

11. **Pătrate**. Se consideră spaţiul cartezian $X0Y$ ce conţine mai multe pătrate ale căror laturi nu se intersectează, dar pot fi incluse unele în celelalte. Pentru fiecare pătrat se cunosc coordonatele centrului său (dat de intersecţia diagonalelor) şi dimensinea laturii. Dat fiind că suntem în spaţiul cartezian, coordonatele pot fi şi negative. Datele se citesc dintr-un fişier. Să se scrie o funcţie, care pentru un punct (x, y) determină pătratul cel mai mic sau cel mai mare (se foloseşte pentru asta unparametru suplimentar in funcţie), pentru care (x, y) este centrul, dacă un astfel de pătrat există. (1p)

ATENŢIE: se punctează NUMAI UNA dintre problemele 1 şi 2!