Structuri de date

Tema 3

David Andreea Raluca – Grupa 10LF221

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 afişarea perechilor $< cheie, valoare >$ pentru toate elementele din tabelă. (3p).

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

```
#include <iostream>
#include <vector>
#include <fstream>
#include <list>

class HashTable
{
private:
	std::list<std::pair<int, int>>* tabel=nullptr;
	int nr_elem = 0;
	int dimensiune=0;
public:
	int hashing(int key)
	{
		return key % dimensiune;
	}
	void redimensionare()
	{
		int dimensiuneNoua = 2 * dimensiune;
		int jndex;
		std::list<std::pair<int, int>>* tabelNou = new std::list<std::pair<int, int>>[dimensiuneNoua];
		for (int index = 0; index < dimensiune; index++)
		{
			for (auto it = tabel[index].begin(); it != tabel[index].end(); it++)
			{
				jndex = it->first % dimensiuneNoua;
```

```
                        tabelNou[jndex].push_back(*it);
                }
        }
        delete[] tabel;
        tabel = tabelNou;
        dimensiune = dimensiuneNoua;
}
void insertie(std::pair<int,int> element)
{
        int index = hashing(element.first);
        tabel[index].push_front(element);
        nr_elem++;
        if (double(nr_elem/dimensiune) > 1.0)
                redimensionare();
}
auto cautare(int key)
{
        int index = hashing(key);
        for (auto it = tabel[index].begin(); it != tabel[index].end(); it++)
                if (it->first == key)
                        return it->second;
        return NULL;
}
void stergere(std::pair<int, int> element)
{
        int index = hashing(element.first);
        for(auto it=tabel[index].begin();it!=tabel[index].end();it++)
                if (it->first == element.first)
                {
                        tabel[index].erase(it);
                        return;
                }
        std::cout << "Elementul pus nu se afla in tabela!" << std::endl;
}
void AlocareMemorie(int nrElemente)
{
        dimensiune = nrElemente;
        tabel = new std::list<std::pair<int, int>>[dimensiune];
}
void DealocareMemorie()
{
        delete[] tabel;
}
void Afisare()
{
        for (int index = 0; index < dimensiune; index++)
        {
                for (auto it = tabel[index].begin(); it != tabel[index].end(); it++)
                        std::cout <<" < "<< it->first <<" , " << it->second <<" > ";
                std::cout << std::endl;
        }
```

```
	}
};

void Interfata()
{
	std::cout << "Comenzile sunt:" << std::endl;
	std::cout << "1 - adaugare element in HashTable" << std::endl;
	std::cout << "2 - cauta un element din HashTable" << std::endl;
	std::cout << "3 - stergere element din HashTable" << std::endl;
	std::cout << "4 - afiseaza HashTable-ul" << std::endl;
	std::cout << " orice numar inafara intervalului [1,4] - EXIT" << std::endl;
	std::cout << std::endl;
}

int main()
{
	int nrElemente;
	HashTable tabela;
	std::ifstream fin("Fisier.in");
	fin >> nrElemente;
	tabela.AlocareMemorie(nrElemente);
	std::pair<int, int> element;
	for (int index = 0; index < nrElemente; index++)
	{
		fin >> element.first >> element.second;
		tabela.insertie(element);
	}
	int comanda=0,key=0,ok=1;
	Interfata();
	while (ok)
	{
		std::cout << "Introduceti comanda dorita:" << std::endl;
		std::cin >> comanda;
		switch (comanda)
		{
		case 1:
			std::cout << "Introduceti elementul pe care vreti sa il adaugati:" << std::endl;
			std::cin >> element.first >> element.second;
			tabela.insertie(element);
			break;
		case 2:
			std::cout << "Introduceti cheia dupa care vreti sa cautati elmentul:" << std::endl;
			std::cin >> key;
			if (tabela.cautare(key) != NULL)
				std::cout << "Elementul cautat dupa cheia " << key << " este " <<
tabela.cautare(key) << std::endl;
			else
				std::cout << "Nu exista niciun element in HashTable cu cheia " << key <<
std::endl;
			break;
		case 3:
```

```
                std::cout << "Introduceti elementul pe care doriti sa il stergeti:" << std::endl;
                std::cin >> element.first >> element.second;
                tabela.stergere(element);
                break;

            case 4:
                std::cout << "HashTable-ul arata asa:" << std::endl;
                tabela.Afisare();
                break;
            default:
                ok = 0;
                std::cout << "EXIT" << std::endl;
                break;
            }
        }
        tabela.DealocareMemorie();
        fin.close();
        return 0;
}
```

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)

```
#include <iostream>
#include <fstream>
#include <unordered_set>

bool VerificarePermutare(std::string sir1,std::string sir2)
{
        if (sir1.size() != sir2.size())
            return false;
        std::unordered_set<char> CaractereSir1;
        for (char caracter : sir1)
            CaractereSir1.insert(caracter);
        for (char caracter : sir2)
        {
            auto iterator = CaractereSir1.find(caracter);
            if (iterator == CaractereSir1.end())
                return false;
            CaractereSir1.erase(caracter);
        }
        return true;
}

int main()
{
```

```
	std::string sir1, sir2;
	std::ifstream fin("Fisier.in");
	fin >> sir1 >> sir2;
	if(VerificarePermutare(sir1,sir2))
		std::cout<<sir2<<" este o permutare al lui "<<sir1;
	else
		std::cout << sir2 << " nu este o permutare al lui " << sir1;
    fin.close();
	return 0;
}
```

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)

```
#include <iostream>
#include <vector>
#include <fstream>
#include <unordered_map>

void CitireDate(int& nr,int&dist, std::vector<float>& vector)
{
	std::ifstream fin("Fisier.in");
	fin >> nr>>dist;
	float element;
	for (int index = 0; index < nr; index++)
	{
		fin >> element;
		vector.push_back(element);
	}
	fin.close();
}

bool VerificareDistanta(std::vector<float> vector,int dist)
{
	std::unordered_map<float, int> tabela;
	for (int index = 0; index < vector.size(); index++)
	{
		if (tabela.find(vector[index]) == tabela.end())
		{
			tabela.insert({ vector[index],index });
		}
		else
		{
			if (index - tabela.find(vector[index])->second <= dist)
				return true;
			else
				tabela.find(vector[index])->second = index;
```

```
                }
        }
        return false;
}

int main()
{
        int nr,dist;
        std::vector<float> vector;
        CitireDate(nr,dist, vector);
        if (VerificareDistanta(vector, dist))
                std::cout << "Exista doua numere egale aflate la o distanta <= " << dist << std::endl;
        else
                std::cout << "Nu exista doua numere egale aflate la o distanta <= " << dist << std::endl;
        return 0;
}
```

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)

```
#include <iostream>
#include <fstream>
#include <unordered_map>
#include <vector>

void CitireDate(std::vector<std::vector<std::string>>& magazine,int& nr_mag)
{
        std::ifstream fin("Fisier.in");
        fin >> nr_mag;
        int nrProduse;
        std::string produs;
        for (int index = 0; index < nr_mag; index++)
        {
                fin >> nrProduse;
                std::vector<std::string> magazin;
                for (int jndex = 0; jndex < nrProduse; jndex++)
                {
                        fin >> produs;
                        magazin.push_back(produs);
                }
                magazine.push_back(magazin);
                magazin.clear();
        }
        fin.close();
}
```

```
int AflareMagazin(std::vector<std::vector<std::string>> magazine, std::unordered_map<std::string, int>
produse, int nr_mag)
{
	int maxim = -1, magMaxim = 0,nr_produse;
	for (int index = 0; index < nr_mag; index++)
	{
		nr_produse = 0;
		for (int jndex = 0; jndex < magazine[index].size(); jndex++)
			if (produse.find(magazine[index][jndex])->second == 1)
				nr_produse++;
		if (nr_produse > maxim)
		{
			maxim = nr_produse;
			magMaxim = index;
		}
	}
	return magMaxim;
}

void AflareProduseMagazin(std::vector<std::vector<std::string>> magazine, std::unordered_map<std::string,
int> produse, int magazin)
{
	std::cout << "Produse exclusive sunt:" << std::endl;
	for (int index = 0; index < magazine[magazin].size(); index++)
		if (produse.find(magazine[magazin][index])->second == 1)
			std::cout << produse.find(magazine[magazin][index])->first << ' ';
	std::cout << std::endl;
}

void AflareProduseExclusive(std::vector<std::vector<std::string>> magazine, int nr_mag)
{
	std::unordered_map<std::string, int> produse;
	for (int index = 0; index < nr_mag; index++)
		for (int jndex = 0; jndex < magazine[index].size(); jndex++)
		{
			if (produse.find(magazine[index][jndex]) == produse.end())
				produse.insert({ magazine[index][jndex],1 });
			else
				produse[magazine[index][jndex]]++;
	  }
	int magazin = AflareMagazin(magazine,produse,nr_mag);
	if (magazin == -1)
		std::cout << "Niciun magazin nu are produse exclusive" << std::endl;
	else
	{
		std::cout << "Magazinul cu cele mai multe produse exclusive este " << magazin+1 <<
std::endl;
		AflareProduseMagazin(magazine,produse,magazin);
	}
}
```

```
int main()
{
	int nr_mag;
	std::vector<std::vector<std::string>> magazine;
	CitireDate(magazine, nr_mag);
	AflareProduseExclusive(magazine, nr_mag);
	return 0;
}
```

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}.

```
#include <iostream>
#include <unordered_map>
#include <vector>
#include <fstream>

void CitireDate(std::vector<std::string>& cuvinte)
{
	std::ifstream fin("Fisier.in");
	std::string cuvant;
	while (!fin.eof())
	{
		fin >> cuvant;
		cuvinte.push_back(cuvant);
	}
	fin.close();
}

void AfisareGrupa(std::vector<std::string> grupa)
{
	std::cout << "{";
	for (int index = 0; index < grupa.size(); index++)
	{
		std::cout << grupa[index];
		if (index != grupa.size() - 1)
			std::cout << ",";
	}
	std::cout << "}";
	std::cout << std::endl;
}
```

```
void CreareHashCuvant(std::string cuvant,std::unordered_map<char,int>& litere)
{
	for (char caracter : cuvant)
	{
		litere.insert({ caracter,1 });
	}
}

void GasireCuvinte(std::vector<std::string>& cuvinte,std::string cuvant)
{
	std::unordered_map<char, int> litere;
	std::vector<std::string> grupa;
	for (int index = 0; index < cuvinte.size(); index++)
	{
		CreareHashCuvant(cuvant,litere);
		for (char caracter : cuvinte[index])
		{
			auto iterator = litere.find(caracter);
			if (iterator != litere.end())
			{
				if (iterator->second == 1)
					litere.erase(iterator);
				else
					iterator->second=iterator->second-1;
			}
			else
				break;
		}
		if (litere.empty())
		{
			grupa.push_back(cuvinte[index]);
			cuvinte.erase(cuvinte.begin() + index);
			index--;
		}
		else
		   litere.clear();
	}
	AfisareGrupa(grupa);
	grupa.clear();
}

void FormareGrupe(std::vector<std::string> cuvinte)
{
	for (int index = 0; index < cuvinte.size(); index++)
	{
		GasireCuvinte(cuvinte, cuvinte[index]);
		index--;
	}
}
```

```
int main()
{
	std::vector<std::string> cuvinte;
	CitireDate(cuvinte);
	FormareGrupe(cuvinte);
}
```

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)

```
#include <iostream>
#include <time.h>
#include <vector>
#include <unordered_map>
#include <stdlib.h>

class Triplete
{
 public:
	int n1,n2, n3;
};

void CitireDate(int& nrAruncari,std::vector<Triplete>& vectorTriplete)
{
	std::cin>> nrAruncari;
	int nrTriplete;
	std::cin >> nrTriplete;
	Triplete numere;
	for (int index = 0; index < nrTriplete; index++)
	{
		std::cin>> numere.n1 >> numere.n2 >> numere.n3;
		vectorTriplete.push_back(numere);
	}
}

void GenerareAruncari(std::vector<Triplete>& aruncari,int nrAruncari)
{
	Triplete aruncare;
	srand(time(NULL));
	while (nrAruncari)
	{
		aruncare.n1 = 1 + rand() % 6;
		aruncare.n2 = 1 + rand() % 6;
```

```
                aruncare.n3 = 1 + rand() % 6;
                aruncari.push_back(aruncare);
                nrAruncari--;
        }
}

void CreareHashTable(std::unordered_map<int, int>& tabela,Triplete numar)
{
        tabela.insert({ numar.n1,1 });
        if (tabela.find(numar.n2) == tabela.end())
                tabela.insert({ numar.n2,1 });
        else
                tabela.find(numar.n2)->second++;
        if (tabela.find(numar.n3) == tabela.end())
                tabela.insert({ numar.n3,1 });
        else
                tabela.find(numar.n3)->second++;
}

bool VerificareAruncare(std::unordered_map<int, int> tabela, Triplete numar)
{
        if (tabela.find(numar.n1) != tabela.end())
        {
                if (tabela.find(numar.n1)->second > 1)
                        tabela.find(numar.n1)->second--;
                else
                        tabela.erase(tabela.find(numar.n1));
        }
        if (tabela.find(numar.n2) != tabela.end())
        {
                if (tabela.find(numar.n2)->second > 1)
                        tabela.find(numar.n2)->second--;
                else
                        tabela.erase(tabela.find(numar.n2));
        }
        if (tabela.find(numar.n3) != tabela.end())
        {
                if (tabela.find(numar.n3)->second > 1)
                        tabela.find(numar.n3)->second--;
                else
                        tabela.erase(tabela.find(numar.n3));
        }
        if (tabela.empty())
                return true;
        return false;
}

void AflareAparitiTriplete(int nrAruncari, std::vector<Triplete> vectorTriplete)
{
        std::vector<Triplete> aruncari;
        std::unordered_map<int,int> tabela;
```

```
	GenerareAruncari(aruncari,nrAruncari);
	int nrOri;
	for (int index = 0; index < vectorTriplete.size(); index++)
	{
		nrOri = 0;
		CreareHashTable(tabela, vectorTriplete[index]);
		for (int index = 0; index < aruncari.size(); index++)
		{
			if (VerificareAruncare(tabela, aruncari[index]))
				nrOri++;
		}
		tabela.clear();
		std::cout << "Tripletul " << vectorTriplete[index].n1 << " " << vectorTriplete[index].n2 << " "
<< vectorTriplete[index].n3 << " a fost aruncat de " << nrOri << std::endl;
	}
}

int main()
{
	int nrAruncari;
	std::vector<Triplete> vectorTriplete;
	CitireDate(nrAruncari, vectorTriplete);
	AflareAparitiTriplete(nrAruncari, vectorTriplete);
	return 0;
}
```