Structuri de date – Probleme

1. Se consideră doi vectori v_1 cu nr_1 elemente şi v_2 cu nr_2 elemente. Vectorul v_1 este sortat crescător şi vectorul v_2 este sortat descrescător. Să se obţină un al treilea vector v_3 care conţine atât elementele lui v_1 cât şi elementele lui v_2 şi care este sortat crescător. Folosiţi o metodă eficientă. (1p)

```
#include <iostream>
//O(n1+n2)
void alocareMemorie(int& n, int*& vector) {
	std::cout << "n=";
	std::cin >> n;
	vector = new int[n];
}

void alocareVect3(int n1, int n2, int& n3, int*& vector) {
	n3 = n1 + n2;
	vector = new int[n3];
}

void citire(int n, int*& vector) {
	for (int i = 0; i < n; i++) {
		std::cin >> vector[i];
	}
}

void afisare(int n, int*& vector) {
	for (int i = 0; i < n; i++) {
		std::cout << vector[i] << " ";
	}
}

void interclasare(int n1, int n2, int*& v1, int*& v2, int*& v3) {
	int i = 0, j = n2 - 1, k = 0;
	while (i < n1 && j >-1) {
		if (v1[i] < v2[j]) {
			v3[k] = v1[i];
			i++;
		}
		else {
			v3[k] = v2[j];
			j--;
		}
		k++;
	}
	if (i == n1)
		for (i = j; i >= 0; i--) {
			v3[k] = v2[i];
			k++;
		}
	else
		if (j == -1) {
			for (j = i; j < n1; j++)
```

```
                {
                        v3[k] = v1[j];
                        k++;
                }
        }
}

void dealocare(int n, int*& v) {
        delete[] v;
}

int main()
{
        int* v1, * v2, * v3;
        int n1 = 0, n2 = 0, n3 = 0;
        alocareMemorie(n1, v1);
        alocareMemorie(n2, v2);
        alocareVect3(n1, n2, n3, v3);
        citire(n1, v1);
        citire(n2, v2);
        interclasare(n1, n2, v1, v2, v3);
        afisare(n3, v3);
        dealocare(n1, v1);
        dealocare(n2, v2);
        dealocare(n3, v3);
        return 0;
}
```

2. Se consideră un vector conţinând nr numere naturale. Scrieţi o funcţie care are ca parametru vectorul şi dimensiunea acestuia şi care returnează cel mai mare număr natural care se poate forma cu toate cifrele pare ale numerelor existente în vector. Folosiţi un algoritm eficient. (1p)
   Exemplu: pentru $v = \{369, 113, 2, 0, 33, 1354, 42\}$ funcţia va return 644220.

```
#include <iostream>
//O(nr)
void alocareMemorie(int& n, int*& vector) {
        std::cout << "n=";
        std::cin >> n;
        vector = new int[n];
}

void alocareFr(int*& fr) {
        fr = new int[5];
}

void citire(int n, int*& vector) {
        for (int i = 0; i < n; i++) {
                std::cin >> vector[i];
        }
}

int cifPar(int nr, int*& v) {
        int* fr;
```

```
	int p = 1, n = 0;
	alocareFr(fr);
	for (int i = 0; i < 5; i++) {
		fr[i] = 0;
	}
	for (int i = 0; i < nr; i++) {
		if (v[i] == 0)
			fr[0]++;
		while (v[i]) {
			if (v[i] % 2 == 0)
				fr[(v[i] % 10) / 2]++;
			v[i] /= 10;
		}
	}
	for (int i = 4; i >= 0; i--) {
		while (fr[i]) {
			n = n * 10 + i * 2; fr[i]--;
		}
	}
	delete[] fr;
	return n;
}

void dealocare(int n, int*& v) {
	delete[] v;
}

int main()
{
	int* v;
	int nr = 0;
	alocareMemorie(nr, v);
	citire(nr, v);
	std::cout << "Nr cautat " << cifPar(nr, v);
	dealocare(nr, v);

	return 0;
}
```

3. Se citesc dintr-un fişier un număr de elevi. Fiecare elev are un nume, un prenume şi 3 note, numere naturale. Se va folosi pentru un elev un **tuple** cu câmpurile nume şi prenume de tip **string** (căutati pe net documentaţie) şi cu trei câmpuri de note de tip **int**. Elevii vor fi memoraţi într-un obiect de tip **std::vector<std::tuple<std::string, std::string, int, int, int> >**. Să se sorteze vectorul de elevi descrescător după medie şi să se afişeze frumos, punând în evidenţă elevii cu media mai mică decât 5. (2p)

```
#include <iostream>
#include <tuple>
#include <vector>
#include <string>
#include <algorithm>
#include <fstream>
//O(nlogn)
```

```
void citireVector(int& n, std::vector<std::tuple<std::string, std::string, int, int, int> >& vector) {
	std::ifstream fin("date.in");
	fin >> n;
	for (int i = 0; i < n; i++){
		std::tuple<std::string,std::string, int, int, int> elev;
		fin >> std::get<0>(elev);
		fin >> std::get<1>(elev);
		fin >> std::get<2>(elev);
		fin >> std::get<3>(elev);
		fin >> std::get<4>(elev);
		vector.push_back(elev);
	}
}

void afisare(std::vector<std::tuple<std::string, std::string, int, int, int> > vector) {
	float medie;
	for (int i = 0; i < vector.size(); i++) {
		medie = (std::get<2>(vector[i]) + std::get<3>(vector[i]) + std::get<4>(vector[i])) / 3;
		if (medie < 5)
			std::cout << "Copil cu medie mai mica decat 5" << std::endl;
		std::cout<< std::get<0>(vector[i])<<" ";
		std::cout<< std::get<1>(vector[i])<<" ";
		std::cout<< std::get<2>(vector[i])<<" ";
		std::cout<< std::get<3>(vector[i])<<" ";
		std::cout<< std::get<4>(vector[i])<<" ";
		std::cout << std::endl;
	}
}

bool sortare(std::tuple<std::string, std::string, int, int, int> elev_1, std::tuple<std::string, std::string, int, int, int> elev_2) {
	int medie_1 = (std::get<2>(elev_1) + std::get<3>(elev_1) + std::get<4>(elev_1))/3;
	int medie_2 = (std::get<2>(elev_2) + std::get<3>(elev_2) + std::get<4>(elev_2))/3;
	if (medie_1 < medie_2)
		return false;
	return true;
}
int main()
{
	std::vector<std::tuple<std::string, std::string, int, int, int> > vector;
	int n = 0;
	citireVector(n, vector);
	std::sort(vector.begin(), vector.end(), sortare);
	afisare(vector);
	return 0;
}
```

7. Alex are un joc cu blocuri din lemn de forma unor paralelelipipede dreptunghice cu baza pătrată de dimensiune fixă. Blocurile au înălţimi diferite. Alex aranjază aceste blocuri în pătrate concentrice.

   a) Să se verifice dacă un astfel de aranjament, reprezentat printr-o matrice, are formă piramidală, adică: toate blocurile dintr-un pătrat sunt mai mici decât cele aflate în regiunea din interiorul acestui pătrat (1p).

   b) Dacă pe baza fiecărui pătrat este gravat un număr pozitiv (ce reprezintă înălţimea blocului în cm), să se scrie o funcţie care calculează pentru un pătrat k înălţimea medie a cuburilor de pe acel pătrat. (1p).

```
#include <iostream>
#include <fstream>
//O(n)
void alocMem(int& n, float**& matrice) {
	std::cout << "Nr linii si colane: ";
	std::cin >> n;
	matrice = new float* [n];
	for (int i = 0; i < n; i++) {
		matrice[i] = new float[n];
	}
}

void citireMatrice(int n, float**& matrice) {
	std::ifstream fin("date.in");
	for (int i = 0; i < n; i++)
		for (int j = 0; j < n; j++)
			fin >> matrice[i][j];
}

float maximPrimulPatrat(int n, float**& matrice) {
	int max = 0;
	for (int j = 0; j < n; j++) {
		if (matrice[0][j] > max)
			max = matrice[0][j];
		if (matrice[n - 1][j] > max)
			max = matrice[n - 1][j];
	}
	for (int i = 1; i < n - 1; i++) {
		if (matrice[i][0] > max)
			max = matrice[i][0];
		if (matrice[i][n - 1] > max)
			max = matrice[i][n - 1];
	}
	return max;
}

int piramida(int n, float**& matrice) {
	int ok = 1, k = 1, max = 0, min = 99999;
	float varf = matrice[n / 2][n / 2];
	float maxim_anterior = maximPrimulPatrat(n, matrice);
	while (k < n - k - 2) {
		min = 99999;
```

```
                max = 0;
                for (int j = k; j < n - k; j++) {
                        if (matrice[k][j] > max)
                                max = matrice[k][j];
                        if (matrice[n - k - 1][j] > max)
                                max = matrice[n - k - 1][j];
                        if (matrice[k][j] < min)
                                min = matrice[k][j];
                        if (matrice[n - k - 1][j] < min)
                                min = matrice[n - k - 1][j];
                }
                for (int i = k + 1; i < n - k - 1; i++) {
                        if (matrice[i][k] > max)
                                max = matrice[i][k];
                        if (matrice[i][n - k - 1] > max)
                                max = matrice[i][n - k - 1];
                        if (matrice[i][k] < min)
                                min = matrice[i][k];
                        if (matrice[i][n - k - 1] < min)
                                min = matrice[i][n - k - 1];
                }
                if (maxim_anterior >= min) {
                        ok = 0;
                        return ok;
                }
                else {
                        maxim_anterior = max;
                        k++;
                }
        }
        if (maxim_anterior >= varf)
                ok = 0;
        return ok;
}

void medie(int n, float**& matrice) {
        int k, c = 0;
        float med = 0;
        std::cout << "Numarul k este: ";
        std::cin >> k;
        for (int j = k - 1; j < n - k + 1; j++) {
                c += 2;
                med += matrice[k - 1][j] + matrice[n - k][j];
        }
        for (int i = k; i < n - k; i++) {
                c += 2;
                med += matrice[i][k - 1] + matrice[i][n - k];
        }
        med /= c;
        std::cout << "Medie linia k: " << med << std::endl;
}

int main()
{
        int n = 0;
        float** matrice;
        alocMem(n, matrice);
        citireMatrice(n, matrice);
```

```
	if (piramida(n, matrice) == 1)
		std::cout << "Este constructie piramidala." << std::endl;
	else
		std::cout << "Nu este constructie piramidala." << std::endl;
	medie(n, matrice);

	return 0;
}
```

8. Se consideră o matrice $matr$ cu $nrows$ linii şi $ncols$ coloane, cu $ncols < 10$, ale cărei elemente sunt numere naturale formate dintr-o singură cifră. Se consideră că fiecare coloană col reprezintă un număr în baza $col + 2$. Să se scrie o funcţie care plasează numerele transformate în baza zece într-un vector $numbers$ şi returnează **true**, dacă matricea a fost validă şi **false** altfel. În cazul în care matricea a fost validă să se afişeze acest vector de numere. Matricea este validă dacă toate elementele de pe coloana col sunt numere naturale din intervalul $[0, col + 2)$. (2p)

```
#include <iostream>
#include <cmath>
#include<vector>
#include <fstream>
//O(nrows*ncols)
void alocareMemorieMatrice(int& nrows, int& ncols, int**& matrice) {
	std::cout << "Numar linii=";
	std::cin >> nrows;
	std::cout << "Numar coloane=";
	std::cin >> ncols;
	matrice = new int* [nrows];
	for (int i = 0; i < nrows; i++) {
		matrice[i] = new int[ncols];
	}
}

void citire(int nrows, int ncols, int**& matrice) {
	std::ifstream fin("date.in");
	for (int i = 0; i < nrows; i++) {
		for (int j = 0; j < ncols; j++) {
			fin >> matrice[i][j];
		}
	}
}

bool validare(int nrows, int ncols, int**& matrice, std::vector <int>& vector) {
	int col, nr;
	for (int j = 0; j < ncols; j++) {
		nr = 0;
		col = j;
		for (int i = 0; i < nrows; i++) {
			if (matrice[i][j] >= col + 2)
				return false;
			nr += matrice[i][j] * pow(col + 2, nrows - i - 1);
		}
		vector.push_back(nr);
```

```
		}
		return true;
}

void afisare(int nrows, int ncols, int**& matrice, std::vector<int>vector) {
		if (validare(nrows, ncols, matrice, vector) == true) {
				std::cout << "Vectorul este ";
				for (int i = 0; i < vector.size(); i++) {
						std::cout << vector[i] << " ";
				}
		}
		else
				std::cout << "Matricea nu este valida.";
}

void dealocare(int nrows, int**& matrice) {
		for (int i = 0; i < nrows; i++)
		{
				delete[] matrice[i];
		}
		delete[] matrice;
}
int main()
{
		int** matrice;
		std::vector<int>vector;
		int nrows = 0, ncols = 0;

		alocareMemorieMatrice(nrows, ncols, matrice);
		citire(nrows, ncols, matrice);
		afisare(nrows, ncols, matrice, vector);
		dealocare(nrows, matrice);
		return 0;
}
```

9. Fiind dată o matrice pătratică de dimensiuni $height \times width$, să se verifice pentru care dintre cele 4 zone determinate de diagonala principală şi de cea secundară suma elementelor este maximă.

```
#include <iostream>
#include<vector>
#include <fstream>
//O(height^2)

void alocareMemorieMatrice(int& height, int& width, int**& matrice) {
		std::cout << "Numarul de linii si coloane: ";
		std::cin >> height >> width;
		matrice = new int* [height];
		for (int i = 0; i < height; i++) {
				matrice[i] = new int[width];
		}
}

void citire(int height, int width, int**& matrice) {
		std::ifstream fin("date.in");
```

```
	for (int i = 0; i < height; i++) {
		for (int j = 0; j < width; j++) {
			fin >> matrice[i][j];
		}
	}
}
int sumaMaxima(int height, int width, int**& matrice) {
	int max = 0, zona = 0, suma1 = 0, suma2 = 0, suma3 = 0, suma4 = 0;
	for (int i = 0; i < height; i++)
		for (int j = 0; j < width; j++) {
			if (i < j && i + j < height - 1)
				suma1 += matrice[i][j];
			if (i > j && i + j > height - 1)
				suma2 += matrice[i][j];
			if (i > j && i + j < height - 1)
				suma3 += matrice[i][j];
			if (i<j && i + j>height - 1)
				suma4 += matrice[i][j];
		}
	max = suma2;
	zona = 2;
	if (suma1 > suma2) {
		max = suma1;
		zona = 1;
	}
	if (suma3 > max) {
		max = suma3;
		zona = 3;
	}
	if (suma4 > max) {
		max = suma4;
		zona = 4;
	}
	return zona;
}

void dealocare(int height, int**& matrice) {
	for (int i = 0; i < height; i++)
	{
		delete[] matrice[i];
	}
	delete[] matrice;
}

int main()
{
	int** matrice;
	int height = 0, width = 0;
	alocareMemorieMatrice(height, width, matrice);
	citire(height, width, matrice);
	std::cout << "Suma maxima se gaseste in zona: " << sumaMaxima(height, width, matrice);
	dealocare(height, matrice);
	return 0;
}
```

11. Să se definească structura *matrice*, care să reţină elementele unei matrice dreptunghiulare(alocate dinamic), precum şi numărul de linii şi de coloane ale acesteia. Utilizând o variabilă de tipul structurii create să se implementeze următoarele funcţii:

    - Alocare şi dealocare de memorie, citire şi afişare a matricei.(0.5p)
    - Dublarea liniilor ce conţin doar elemente pare(0.5p)

```
#include <iostream>
#include <fstream>
//O(n*m); O(n^2)
struct Matrice {
	int** matrice;
	int n, m;
};

void alocareMemorie(Matrice& matr) {
	std::cout << "Numarul de linii si coloane: ";
	std::cin >> matr.n >> matr.m;
	matr.matrice = new int* [matr.n * 2];
	for (int i = 0; i < matr.n * 2; i++) {
		matr.matrice[i] = new int[matr.m];
	}
}

void citire(Matrice& matr) {
	std::ifstream fin("date.in");
	for (int i = 0; i < matr.n; i++) {
		for (int j = 0; j < matr.m; j++) {
			fin >> matr.matrice[i][j];
		}
	}
}
int nrLiniiPare(Matrice matr)
{
	int nr = 0, ok;
	for (int i = 0; i < matr.n; i++) {
		ok = 1;
		for (int j = 0; j < matr.m; j++)
			if (matr.matrice[i][j] % 2 != 0)
			{
				ok = 0; break;
			}
		if (ok == 1)
			nr++;
	}
	return nr;
}


void dublareLinie(Matrice& matr, int linie, int& n) {
	for (int i = matr.n - 1; i > linie - 1; i--) {
		for (int j = 0; j < matr.m; j++)
			matr.matrice[i + 1][j] = matr.matrice[i][j];
	}
```

```
		n++;
}
void liniiPare(Matrice& matr, int& n) {
		int ok = 1;
		for (int i = 0; i < matr.n; i++) {
			ok = 1;
			for (int j = 0; j < matr.m; j++)
				if (matr.matrice[i][j] % 2 != 0) {
					ok = 0;
					break;
				}
			if (ok == 1) {
				dublareLinie(matr, i, matr.n); i++;
			}
		}
		n = matr.n;
}

void afisare(Matrice& matr) {
		std::cout << "Matricea modificata: " << std::endl;
		for (int i = 0; i < matr.n; i++) {
			for (int j = 0; j < matr.m; j++) {
				std::cout << matr.matrice[i][j] << " ";
			}
			std::cout << std::endl;
		}
}

void dealocare(Matrice& matr) {
		for (int i = 0; i < matr.n; i++)
		{
			delete[] matr.matrice[i];
		}
		delete[] matr.matrice;
}

int main()
{

		Matrice matr;
		matr.n = 0; matr.m = 0;
		alocareMemorie(matr);
		citire(matr);
		liniiPare(matr, matr.n);
		afisare(matr);
		dealocare(matr);
		return 0;
}
```

12. Se consideră un vector de intervale. Se va folosi tipul **pair** pentru a reţine cele 2 capete(numere întregi) ale unui interval. Să se definească o funcţie care să determine în cel mai eficient mod intervalul de lunigime minimă în care se află un element dat ca parametru. Dacă acesta nu se află în interiorul niciunui intervalul se va afişa un mesaj corespunzător.(2p)

```
#include <iostream>
#include<vector>
//O(n)

void citire(int& n, std::vector<std::pair<int, int> >& vector) {
	std::cout << "Numarul de perechi din vector: ";
	std::cin >> n;
	std::cout << "Aceste perechi sunt: " << std::endl;
	for (int i = 0; i < n; i++) {
		std::pair<int, int> element;
		std::cin >> element.first;
		std::cin >> element.second;
		vector.push_back(element);
	}
}

void interval(int n, std::vector<std::pair<int, int> > vector, int x) {
	int a, b, minim = 99999;
	std::cout << "Numarul cautat: ";
	std::cin >> x;
	for (int i = 0; i < n; i++) {
		if (x >= vector[i].first && x <= vector[i].second) {
			if (vector[i].second - vector[i].first < minim) {
				minim = vector[i].second - vector[i].first;
				a = vector[i].first;
				b = vector[i].second;
			}
		}

	}
	std::cout << "Intervalul de lungime minima [" << a << "," << b << "]" << std::endl;
}

int main()
{
	int n = 0, x = 0;
	std::vector<std::pair<int, int> > vector;
	citire(n, vector);
	interval(n, vector, x);
	return 0;
}
```