Matrici (tablouri bidimensionale)

```
/*
 * 1. Fiind date doua matrici F si A (dim(F)<dim(A)) si un intreg p, matricea
 caracteristica, C, cu pasul p se va construi astfel:
 - C[i,j] = sum(F * AklF),
 unde:
 - AklF - o matrice cu elemente din A, de dimensiune egala cu dimensiunea matricii F,
 elementul de pe pozitia (k,l) din A fiind elementul din stanga sus
 al matricii AklF (se merge in dreapta apoi in jos); v. exemplu
 - * - produsul Hadamard a doua matrici (Hij=Aij*Bij)
 - sum - suma elementelor unei matrici
 Pasul p=2 inseamna ca intai se iau in considerare elementele de pe pozitiile (1,1),
 apoi (1,3), ..., (3,1), ...
 Sa se scrie o functie care primeste ca parametri un vector de matrici de tip F,
 o matrice A, pasul p; se va returna un vector de matrici caracteristice.
----
Exemplu:
Fie matricea F={{1,0},{0,1}}, matricea A={{1,0,1,0},{0,1,0,0},{1,0,1,0},{0,0,0,0}}, pasul p=2.
Matricile AklF care se vor considera sunt:
{{1,0},{0,1}}; {{1,0},{0,0}} - se incepe cu elementul de pe pozitia [0,0] si se merge spre dreapta cu pasul 2
{{1,0},{0,0}}; {{1,0},{0,0}} - se continua cu elementul de pe pozitia [2,0] (pasul 2) si se merge spre dreapta cu
pasul 2
Matricea caracteristica, C, va fi: {{2,1},{1,1}}.
 */

#include <iostream>

#define DIM 10

struct Matrix {
   int n, m;
   int el[DIM][DIM];

   Matrix(int n = 0, int m = 0) {
      this->n = n;
      this->m = m;
      for (int i = 0; i < n; i++) {
         for (int j = 0; j < m; j++) {
            this->el[i][j] = 0;
         }
      }
   }

   int getLines() {
      return this->n;
   }
```

```
    int getColumns() {
        return this->m;
    }

    int getElementAt(int i, int j) {
        return this->el[i][j];
    }

    void setElementAt(int i, int j, int v) {
        this->el[i][j] = v;
    }
};

struct ArrayMatrix {
    int n;
    Matrix el[DIM];

    ArrayMatrix() {
        this->n = 0;
    }

    int getSize() {
        return this->n;
    }

    Matrix getElementAt(int i) {
        return this->el[i];
    }

    void setElementAt(int i, Matrix m) {
        this->el[i] = m;
    }

    ArrayMatrix add(Matrix m) {
        this->el[n++] = m;
        return *this;
    }
};

Matrix getMatrix(Matrix a, int k, int l, int lines, int columns) {
    Matrix result(lines, columns);
    for (int i = k; i < k + lines; i++) {
        for (int j = l; j < l + columns; j++) {
            result.setElementAt(i - k, j - l, a.getElementAt(i, j));
        }
    }
    return result;
```

```
}

/*
 * pre: dim(a)==dim(b)
 */
Matrix hadamard(Matrix a, Matrix b) {
   Matrix result(a.getLines(), a.getColumns());
   for (int i = 0; i < a.getLines(); i++) {
      for (int j = 0; j < b.getColumns(); j++) {
         result.setElementAt(i, j, a.getElementAt(i, j) * b.getElementAt(i, j));
      }
   }
   return result;
}

int sumMat(Matrix a) {
   int s = 0;
   for (int i = 0; i < a.getLines(); i++) {
      for (int j = 0; j < a.getColumns(); j++) {
         s += a.getElementAt(i, j);
      }
   }
   return s;
}

Matrix buildFeatureMatrix(Matrix f, Matrix a, int p) {
   Matrix result((a.getLines() - f.getLines()) / p + 1, (a.getColumns() - f.getColumns()) / p + 1);
   for (int i = 0; i < a.getLines() - f.getLines() + 1; i += p) {
      for (int j = 0; j < a.getColumns() - f.getColumns() + 1; j += p) {
         Matrix m = getMatrix(a, i, j, f.getLines(), f.getColumns());
         result.setElementAt(i / p, j / p, sumMat(hadamard(f, m)));
      }
   }
   return result;
}

ArrayMatrix buildArrayMatrix(ArrayMatrix arrayF, Matrix a, int p) {
   ArrayMatrix result;
   for (int i = 0; i < arrayF.getSize(); i++) {
      Matrix m = buildFeatureMatrix(arrayF.getElementAt(i), a, p);
      result.add(m);
   }
   return result;
}

//================================================initData=============================
===
```

```
Matrix buildCustomMatrix(int n, int m, int k, int el[][DIM]) {
   Matrix result(n, m);
   for (int i = 0; i < k; i++) {
      result.setElementAt(el[i][0], el[i][1], el[i][2]);
   }
   return result;
}

ArrayMatrix initArrF() {
   int el[][DIM] = {{0, 0, 1},
                    {1, 1, 1}};
   Matrix f1 = buildCustomMatrix(2, 2, 2, el);

   int el2[][DIM] = {{1, 0, 1},
                     {0, 1, 1}};
   Matrix f2 = buildCustomMatrix(2, 2, 2, el2);

   ArrayMatrix result;
   result = result.add(f1).add(f2);

   return result;
}

Matrix initA() {
   int el[][DIM] = {{0, 0, 1},
                    {0, 2, 1},
                    {1, 1, 1},
                    {2, 0, 1},
                    {2, 2, 1}};
   Matrix result = buildCustomMatrix(4, 4, 5, el);
   return result;
}

//============================================================================

void printMat(Matrix m) {
   for (int i = 0; i < m.getLines(); i++) {
      for (int j = 0; j < m.getColumns(); j++) {
         std::cout << m.getElementAt(i, j) << " ";
      }
      std::cout << std::endl;
   }
}

void printArrayMatrix(ArrayMatrix arrMat) {
   for (int i = 0; i < arrMat.getSize(); i++) {
      printMat(arrMat.getElementAt(i));
   }
```

```
}


int main() {
   ArrayMatrix arr = initArrF();
   Matrix a = initA();

   ArrayMatrix res = buildArrayMatrix(arr, a, 1);
   printArrayMatrix(res);

   std::cout << "Hello, World!" << std::endl;
   return 0;
}
/*
 *  Se da o matrice A si un intreg k. Sa se sorteze elementele de pe coloana k
  astfel incat matricea rezultata sa aiba aceleasi linii ca si matricea initiala
  (ordinea liniilor poate sa difere).
  S-ar dori o solutie cu complexitatea O(N^2) (N ~ nrLinii(A) ~ nrColoane(A)).

 */

#include <iostream>

#define DIM 10

struct Matrix {
   int n, m;
   int *el[DIM];

   Matrix(int n = 0, int m = 0) {
      this->n = n;
      this->m = m;
      for (int i = 0; i < n; i++) {
         this->el[i] = new int[m];
      }
   }

   int getElementAt(int i, int j) {
      return this->el[i][j];
   }

   int *getLineAt(int i) {
      return this->el[i];
   }

   void setLineAt(int i, int *l) {
      this->el[i] = l;
```

```
    }

    void setElementAt(int i, int j, int v) {
        this->el[i][j] = v;
    }

    int getLines() {
        return this->n;
    }

    int getColumns() {
        return this->m;
    }

    ~Matrix() {
        for (int i = 0; i < this->n; i++) {
            delete[] this->el[i];
        }
    }

};

void sortMatrix(Matrix *m, int k) {
    for (int i = 0; i < m->getLines() - 1; i++) {
        for (int j = i + 1; j < m->getColumns(); j++) {
            if (m->getElementAt(i, k) > m->getElementAt(j, k)) {
                int *aux = m->getLineAt(i);
                m->setLineAt(i, m->getLineAt(j));
                m->setLineAt(j, aux);
            }
        }
    }
}

void printMatrix(Matrix *m) {
    for (int i = 0; i < m->getLines(); i++) {
        for (int j = 0; j < m->getColumns(); j++) {
            std::cout << m->getElementAt(i, j) << " ";
        }
        std::cout << std::endl;
    }
}

void readMatrix(Matrix *a) {
    int v;
    for (int i = 0; i < a->getLines(); i++) {
        for (int j = 0; j < a->getColumns(); j++) {
            std::cin >> v;
```

```
            a->setElementAt(i, j, v);
        }
    }
}

int main() {
    Matrix *a = new Matrix(3, 3);
    std::cout << "read matrix elements (3x3): ";
    readMatrix(a);
    sortMatrix(a, 1);
    printMatrix(a);

    delete a;

    std::cout << "Hello, World!" << std::endl;
    return 0;
}

/*
 *  Iepurasul de Pasti doreste sa distribuie cadouri copiilor cuminti.
Intr-un dosar are fise pentru fiecare copil, fise in care e scris numele acestuia, adresa si diverse informatii
legate de activitatea sa din ultimul an.
Prima fisa din dosar este cea a primului copil pe care il va vizita in acest an.
In partea de jos a fiecarei fise e scris numele urmatorului copil pe care ar   trebui sa il viziteze, fisa acestuia
fiind si urmatoarea fisa din dosar.
Din pacate, dosarul iepurasului a cazut, iar fisele s-ar imprastiat si s-au amestecat intre ele. Ajutati-l pe
iepuras sa isi refaca dosarul.
  ----
  Sa se scrie o functie care primeste ca parametru o matrice, A, reprezentand
  fisele (amestecate) si va afisa dosarul refacut. Fiecare fisa este reprezentata
  pe o linie din A; pe prima coloana numele copilului la care se refera fisa, iar
  pe a doua coloana numele urmatorului copil (numele din partea de jos a fisei).
  ----
  Fiind vorba de foarte multe fise, s-ar dori o solutie in timp liniar.

 */

#include <iostream>
#include <map>

#define DIM 10

struct Matrix {
    int n = 5, m = 2;
    std::string el[DIM][DIM] = {{"hermione", "ginny"},
                                {"fred", "george"},
                                {"harry",   "ron"},
                                {"ginny", "fred"},
```

```
                        {"ron",  "hermione"}};

//    Matrix(int n = 0, int m = 0) {
//        this->n = n;
//        this->m = m;
//        for (int i = 0; i < n; i++) {
//            for (int j = 0; j < m; j++) {
//                this->el[i][j] = 0;
//            }
//        }
//    }

    int getLines() {
        return this->n;
    }

    int getColumns() {
        return this->m;
    }

    std::string getElementAt(int i, int j) {
        return this->el[i][j];
    }
};

std::map<std::string, std::string> createMapOfChildren(Matrix a, int k, int v) {
    std::map<std::string, std::string> m;
    for (int i = 0; i < a.getLines(); i++) {
        m[a.getElementAt(i, k)] = a.getElementAt(i, v);
    }
    return m;
}

std::string findFirst(std::map<std::string, std::string> sd, std::map<std::string, std::string> ds) {
    for (auto &it : sd) {
        if (ds.find(it.first) == ds.end()) {
            return it.first;
        }
    }
    return "";
}

void printFolder(const std::string &first, std::map<std::string, std::string> m) {
    using namespace std;
    auto it = m.find(first);
    while (it != m.end()) {
        cout << it->first << " " << it->second << endl;
        it = m.find(it->second);
```

```
    }
}

void rebuildFolder(const Matrix &m) {
    using namespace std;
    map<string, string> sd = createMapOfChildren(m, 0, 1);
    map<string, string> ds = createMapOfChildren(m, 1, 0);

    string first = findFirst(sd, ds);

    if (first.empty()) {
        cout << "first not found";
        return;
    }

    printFolder(first, sd);

}

int main() {
    Matrix m;
    rebuildFolder(m);

    std::cout << "Hello, Hogwarts!" << std::endl;
    return 0;
}
```