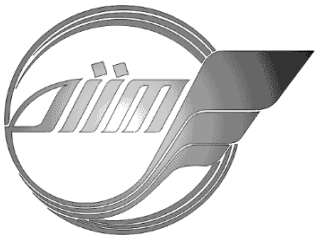
МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ



Дніпровський національний університет
залізничного транспорту імені академіка В. Лазаряна

Кафедра «Комп'ютерні інформаційні технології»

Звіт
з навчальної практики

Виконав:
студент гр.ПЗ1911
Сафонов Д.Є.
Прийняла:
Шаповал І.В.

Дніпро, 2020

1. Постановка завдання

Розробити та реалізувати структуру даних, що відображає роботу з розрідженою матрицею (не менше 70% елементів дорівнюють нулю). Необхідно реалізувати наступні операції:

- додавання елемента матриці (первинне розміщення елемента при створенні матриці, додавання елемента у вже створену матрицю – заміна нуля на інше);
- заміна значення $a_{i,j}$ на інше;
- видалення елемента матриці за заданими індексами та по значенню;
- пошук елемента матриці за заданими індексами;
- друк матриці у звичному форматі та форматі розрідженої матриці – індекси та значення. Наприклад, форма для одного елементу (i,j) $a_{i,j}$;
- індивідуальне завдання (може бути оформлено у вигляді декількох функцій, для поліпшення структурованості програми).

Всі операції повинні бути оформлені у вигляді окремої бібліотеки, що підключаються до основної програми.

Тип оформлення інтерфейсу користувача текстовий (консольна програма) або графічний за бажанням.

В якості підвищення складності роботи та оцінки виконання рекомендується використовувати лівосторонній виклик функції, наприклад, для функції додавання елементу матриці
add(i,j) = A, де A – значення елемента.

В звіті в розділі «Вибір методу» необхідно описати розроблену структуру даних, надати її графічне представлення та на основі чого будуть реалізовуватись базові алгоритми. У висновках необхідно оцінити швидкість доступу до елементів структури, гнучкість (збільшення/зменшення розмірів у відповідності до вимог користувача під час виконання програми), види та обсяги використаної пам'яті (обсяги вказати для кожного виду окремо).

ІНДИВІДУАЛЬНІ ЗАВДАННЯ (рівень А)

1. Сформувати одновимірний масив з мінімальних елементів рядків матриці.
2. Сформувати одновимірний масив з сум елементів стовбців матриці.
3. Сформувати одновимірний масив з елементів, що розташовані нижче та ліворуч заданої позиції (номер рядка, номер стовбця).
4. Сформувати одновимірний масив з максимальних елементів стовбців матриці.
5. Сформувати одновимірний масив з елементів розташованих на другорядній діагоналі матриці.
6. Сформувати одновимірний масив з максимальних елементів рядків матриці.
7. Сформувати одновимірний масив з сум елементів рядків матриці.
8. Сформувати одновимірний масив з елементів, що розташовані вище та ліворуч заданої позиції (номер рядка, номер стовбця).
9. Сформувати одновимірний масив з елементів розташованих нижче основної діагоналі матриці.
10. Сформувати одновимірний масив з кількості від'ємних елементів стовбців матриці.
11. Сформувати одновимірний масив з елементів, що розташовані нижче та праворуч заданої позиції (номер рядка, номер стовбця).
12. Сформувати одновимірний масив з максимальних від'ємних елементів стовбців матриці.
13. Сформувати одновимірний масив з елементів розташованих вище другорядній діагоналі матриці.
14. Сформувати одновимірний масив з сум додатних елементів рядків матриці.

15. Сформувати одновимірний масив з елементів, що розташовані вище та праворуч заданої позиції (номер рядка, номер стовбця).
16. Сформувати одновимірний масив з кількості парних елементів рядків матриці.
17. Сформувати одновимірний масив з від'ємних елементів матриці.
18. Сформувати одновимірний масив з середньо арифметичних елементів рядків матриці.
19. Сформувати одновимірний масив з середньо геометричного ненульових елементів стовбців матриці.
20. Сформувати одновимірний масив з сум максимальних елементів рядків матриці та їх позицій.
21. Сформувати одновимірний масив з елементів розташованих на головній діагоналі матриці.
22. Сформувати одновимірний масив з мінімальних елементів рядків матриці та їх позицій.
23. Сформувати одновимірний масив з сум індексів не нульових елементів рядків матриці.
24. Сформувати одновимірний масив з суми індексів мінімальних елементів рядків матриці.

ІНДИВІДУАЛЬНІ ЗАВДАННЯ (рівень В)

1. Додавання матриць.
2. Транспонування матриць.
3. Множення матриці на вектор.
4. Множення двох матриць.
5. Сортування елементів рядків матриці.
6. Сортування елементів стовбців матриці.
7. Сортування рядків матриці за заданим стовбцем.
8. Сортування стовбців матриці за заданим рядком.

СТРУКТУРИ ДЛЯ ЗБЕРІГАННЯ МАТРИЦЬ

1. Одновимірний масив
2. Список
3. Схема Кнута

ІНДИВІДУАЛЬНІ ЗАВДАННЯ

рівень	завдання	Структура
A	Рівень В	3
B	Рівень А	3
C	Рівень А	2
D	Рівень А	1
E	Рівень А	1

2. Зовнішні специфікації

Конструктор об’єкту матриці:

№	Назва	Умовне позначення	Вимоги	Приклад
1	Кількість рядків матриці	N	Ціле число більше нуля	7
2	Кількість стовпців матриці	M	Ціле число більше нуля	1

Пошук заміна значення елементу:

№	Назва	Умовне позначення	Вимоги	Приклад
1	Нове значення	value	Дійсне число	64.67584
2	Рядок	i	Ціле число від нуля до кількості рядків матриці не включно	4
3	Стовпець	j	Ціле число від нуля до кількості стовпців матриці не включно	0

Значення елементу за індексами:

№	Назва	Умовне позначення	Вимоги	Приклад
1	Рядок	i	Ціле число від нуля до кількості рядків матриці не включно	4
2	Стовпець	j	Ціле число від нуля до кількості стовпців матриці не включно	0

Заміна на нуль за значенням(тільки першого елемента):

№	Назва	Умовне позначення	Вимоги	Приклад
1	Значення	value	Дійсне число	64.67584

3. Вибір методу рішення задачі

Розроблений клас розрідженої матриці складається з чотирьох полів:

- Кількість рядків
- Кількість стовпців
- Вказівник на масив вказівників на перші елементи стовпців
- Вказівник на масив вказівників на перші елементи рядків

Елемент у свою чергу складається з семи полів:

- Значення
- Рядок
- Стовпець
- Вказівник на лівого ненульового сусіда
- Вказівник на правого ненульового сусіда
- Вказівник на верхнього ненульового сусіда
- Вказівник на нижнього ненульового сусіда

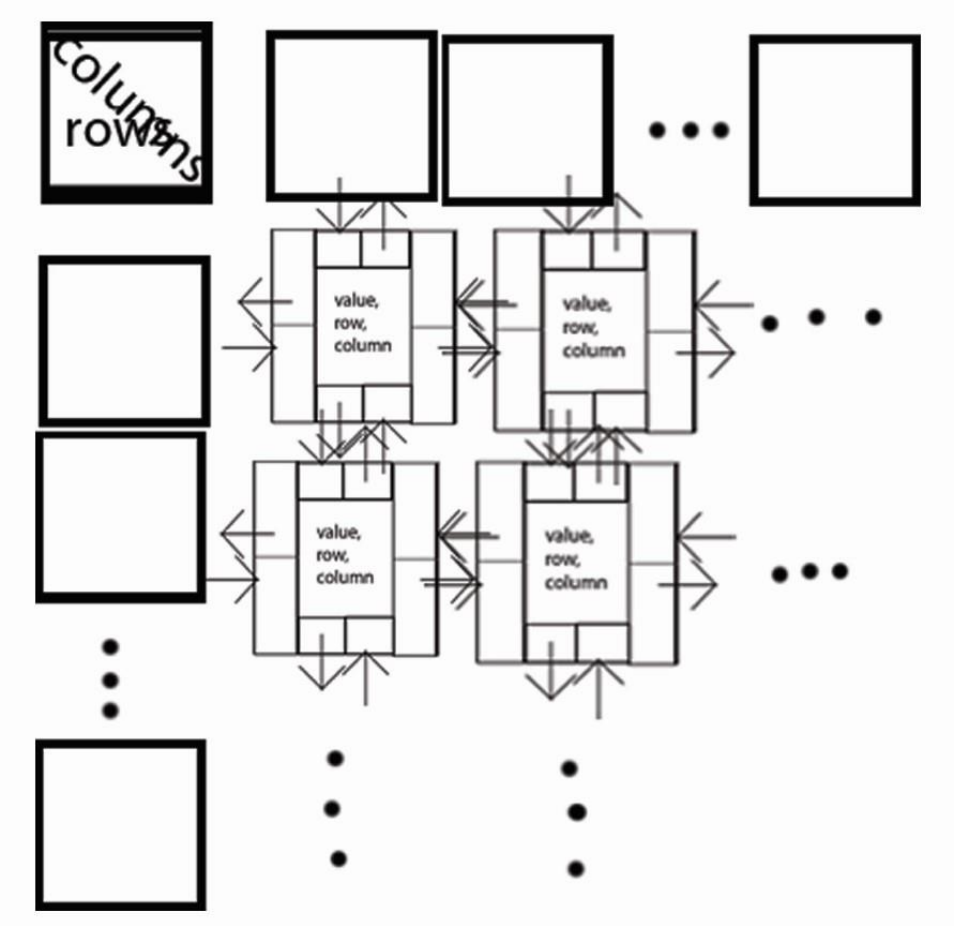


Рисунок 1

Були реалізовані:

- Генерація матриці(заповнення матриці ≤ 30% ненульовими елементами)
  Розраховується максимально можливе число N ненульових елементів.
  В циклі, який повторюється N разів, генерується випадкове число(0 або 1), в залежності від його значення ітерація пропускається , або генерується випадкове значення лементу та рядку та стовпця, такі що заміняти тільки нульовий елемент, якщо цей елемент ненульовий то генерація повторюється доки індекси не будуть вказувати на нульовий елемент.
- Виведення матриці у звичайному вигляді.
  Два вкладених цикла для пересування по матриці, в тілі внутрішнього перевіряється значення елементу та виводиться.
- Виведення матриці у вигляді розрідженої(виводяться тільки значення ненульових елементів та їх індекси).
  Для кожного рядка виводиться перший ненульовий елемент та усі його послідовники в циклі, якщо такого елемента не існує рядок пропускається.
- Зміна елементу:
  - елемент нуль, нове значення нуль - нічого не робити
  - елемент не нуль, нове значення нуль – видалити елемент
  - елемент нуль, нове значення не нуль – створити новий елемент із новим значенням, вказаними індексами. Зв’язати цей елемент із усіма старими.
  - елемент не нуль, нове значення не нуль – замінити значення на нове.

  Лівосторонній виклик функції не був реалізований в угоду функціональності.
- Функція, яка повертає значення елемента за вказаними індексами.

Виклик функції пошуку повертає значення знайденого елемента, або нуль якщо вказівник на елемент – нульовий.

- Видалення за значенням(тільки першого елемента).
  Пошук за значенням та видалення елемента, якщо він існує.
- Оператор додавання(+)
  Два вкладених цикла для пересування по матриці, в тілі внутрішнього додається сума значень відповідних елементів першої та другої матриці до матриці-результату.

- Оператор присвоєння(=)
  Копіювання усіх елементів у матрицю.

Та додаткові службові функції:

- Пошук за індексами
  Перевіряється чи вказівники на рядок та стовпець існують, якщо хоча б один з них не існує – усі елементи рядка/стовпця дорівнюють нулю, потрібний елемент відповідно теж нуль. Далі перевіряється чи більше індекс рядків чи стовпців, в залежності від цього шукають по рядкам чи стовпцям відповідно.
- Пошук за значенням
  Перевіряється чи значення нуль(якщо нуль, то елемент вже видалений).
  Далі в циклі перевіряються значення усіх ненульових елементів, повертається вказівник на перший елемент із потрібним значенням. Якщо жоден з елементів співпадає за значенням з потрібним, повертається нуль.
- Видалення елемента
  Видаляються зв’язки між елементом та його сусідами.
  Створюються нові між сусідами(ігноруючи елемент).

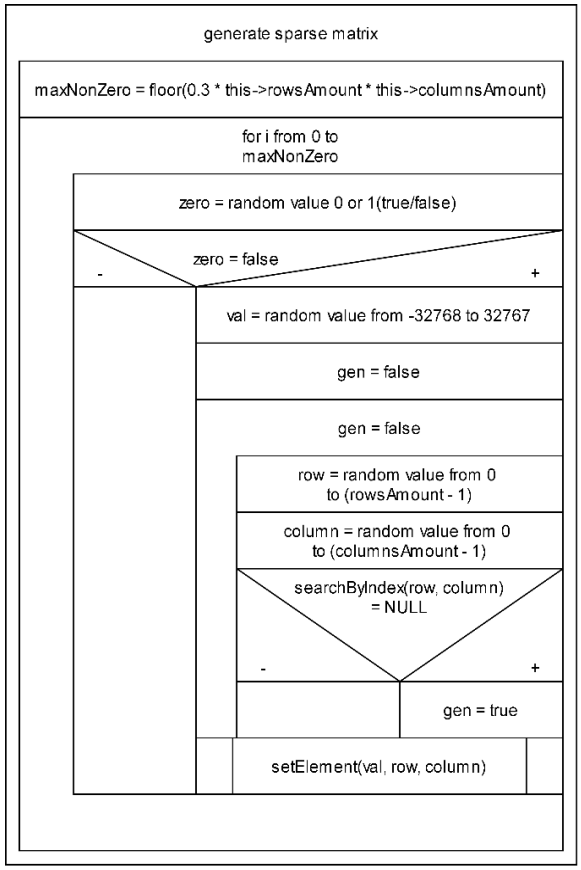
- Вставка перед горизонтальним послідовником
- Вставка перед вертикальним послідовником
- Вставка після горизонтального попередника
- Вставка після вертикального попередника

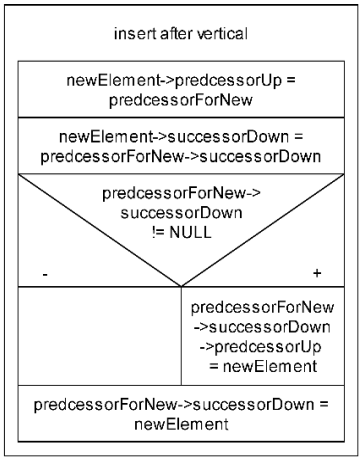
Для усіх вставок алгоритм ідентичний:

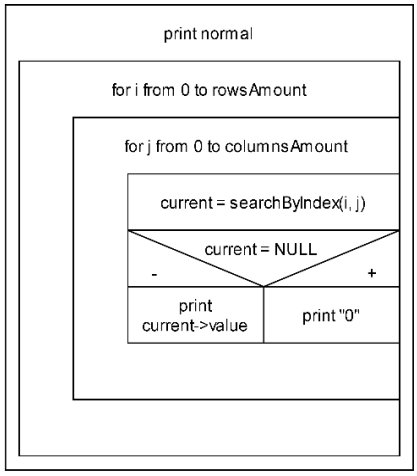
Видалення зв’язків між елементами, між якими вставляється новий.

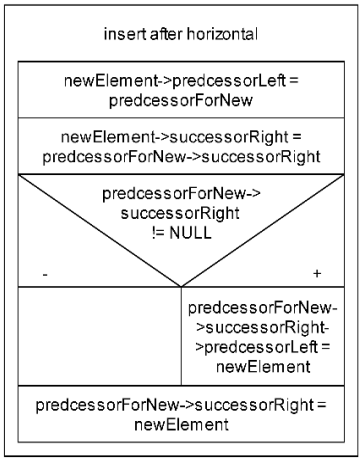
Створення нових зв’язків між старими елементами та новим.

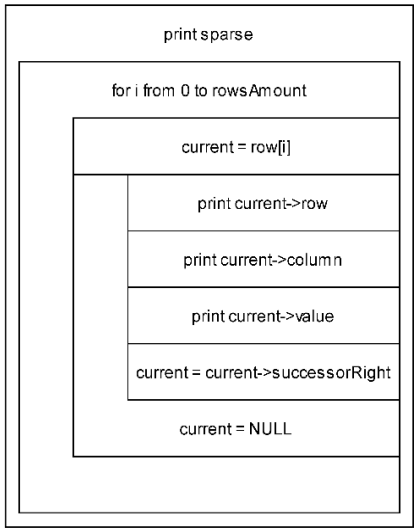
4. Розробка алгоритмів рішення програми

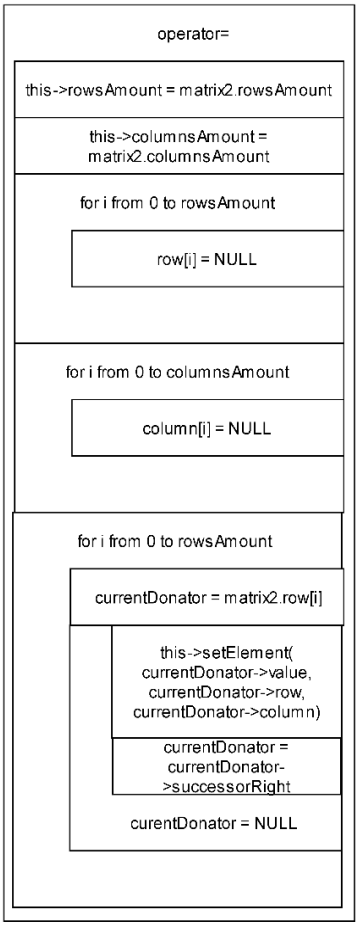

generate sparse matrix
maxNonZero = floor(0.3 * this->rowsAmount * this->columnsAmount)
for i from 0 to maxNonZero
zero = random value 0 or 1(true/false)
zero = false
-
+
val = random value from -32768 to 32767
gen = false
gen = false
row = random value from 0 to (rowsAmount - 1)
column = random value from 0 to (columnsAmount - 1)
searchByIndex(row, column) = NULL
-
+
gen = true
setElement(val, row, column)

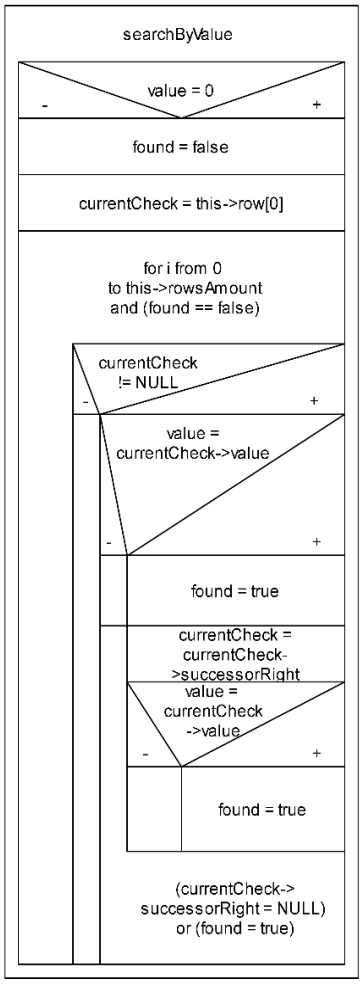

insert after vertical
newElement->predcessorUp = predcessorForNew
newElement->successorDown = predcessorForNew->successorDown
predcessorForNew->successorDown != NULL
-
+
predcessorForNew->successorDown->predcessorUp = newElement
predcessorForNew->successorDown = newElement

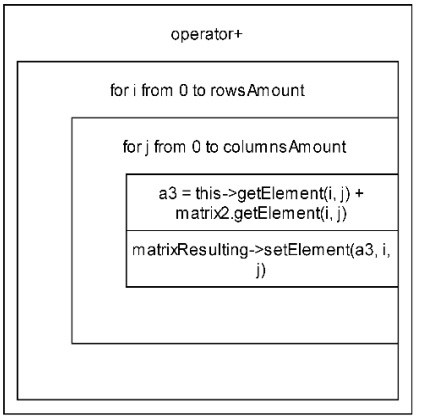

print normal
for i from 0 to rowsAmount
for j from 0 to columnsAmount
current = searchByIndex(i, j)
current = NULL
-
+
print current->value
print "0"

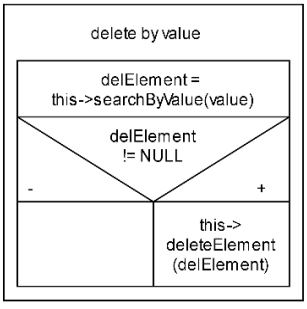

insert after horizontal
newElement->predcessorLeft = predcessorForNew
newElement->successorRight = predcessorForNew->successorRight
predcessorForNew->successorRight != NULL
-
+
predcessorForNew->successorRight->predcessorLeft = newElement
predcessorForNew->successorRight = newElement


print sparse
for i from 0 to rowsAmount
current = row[i]
print current->row
print current->column
print current->value
current = current->successorRight
current = NULL


operator=
this->rowsAmount = matrix2.rowsAmount
this->columnsAmount = matrix2.columnsAmount
for i from 0 to rowsAmount
row[i] = NULL
for i from 0 to columnsAmount
column[i] = NULL
for i from 0 to rowsAmount
currentDonator = matrix2.row[i]
this->setElement(currentDonator->value, currentDonator->row, currentDonator->column)
currentDonator = currentDonator->successorRight
curentDonator = NULL


searchByValue
value = 0
-
+
found = false
currentCheck = this->row[0]
for i from 0 to this->rowsAmount and (found == false)
currentCheck != NULL
-
+
value = currentCheck->value
found = true
currentCheck = currentCheck->successorRight
value = currentCheck ->value
found = true
(currentCheck-> successorRight = NULL) or (found = true)


operator+
for i from 0 to rowsAmount
for j from 0 to columnsAmount
a3 = this->getElement(i, j) + matrix2.getElement(i, j)
matrixResulting->setElement(a3, i, j)


delete by value
delElement = this->searchByValue(value)
delElement != NULL
-
+
this-> deleteElement (delElement)

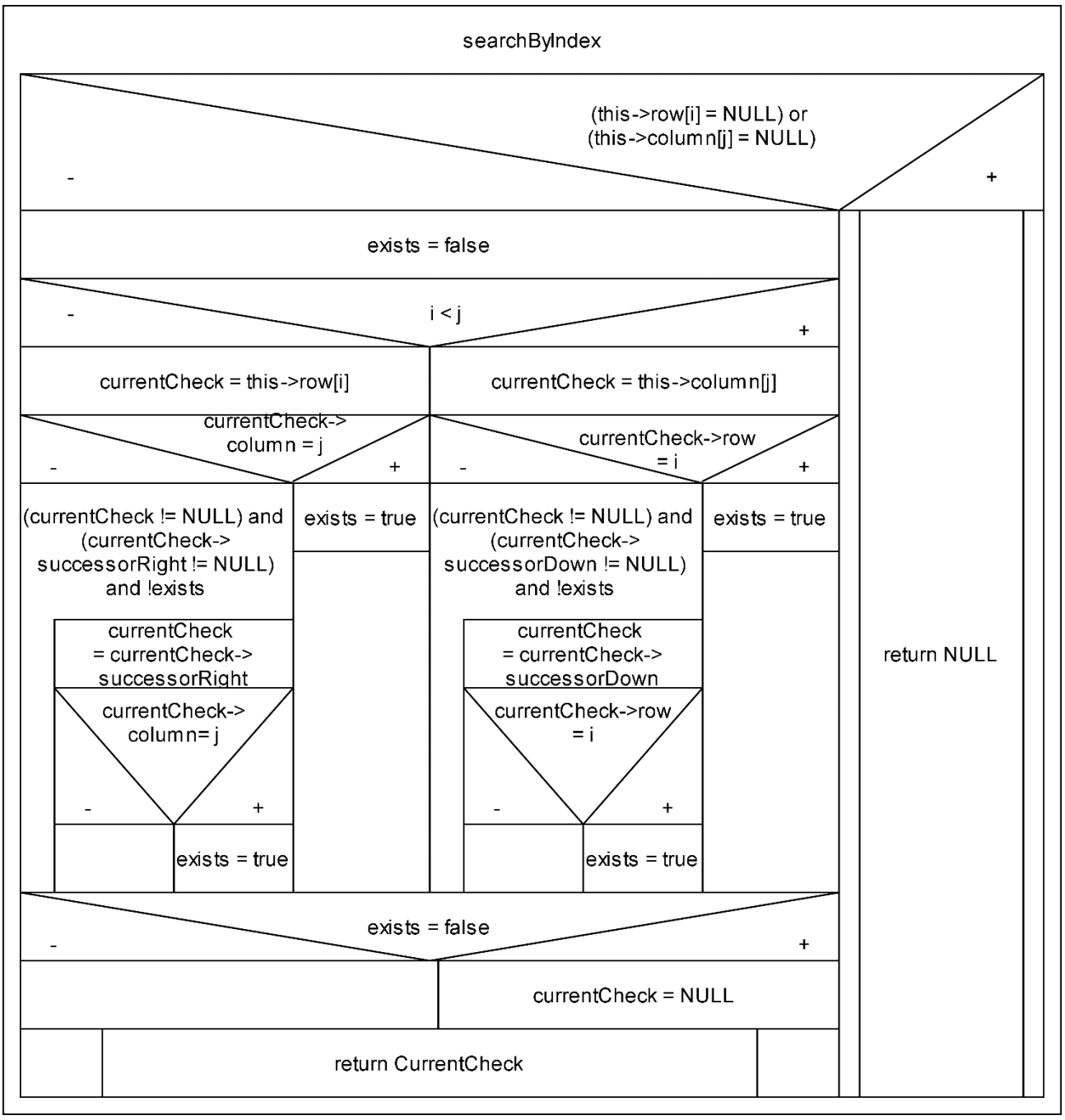

searchByIndex
(this->row[i] = NULL) or
(this->column[j] = NULL)
-
+
exists = false
-
i < j
+
currentCheck = this->row[i]
currentCheck = this->column[j]
currentCheck->
column = j
-
+
currentCheck->row
= i
-
+
(currentCheck != NULL) and
(currentCheck->
successorRight != NULL)
and !exists
exists = true
(currentCheck != NULL) and
(currentCheck->
successorDown != NULL)
and !exists
exists = true
currentCheck
= currentCheck->
successorRight
currentCheck->
column= j
-
+
exists = true
currentCheck
= currentCheck->
successorDown
currentCheck->row
= i
-
+
exists = true
return NULL
exists = false
-
+
currentCheck = NULL
return CurrentCheck

Рисунок 2

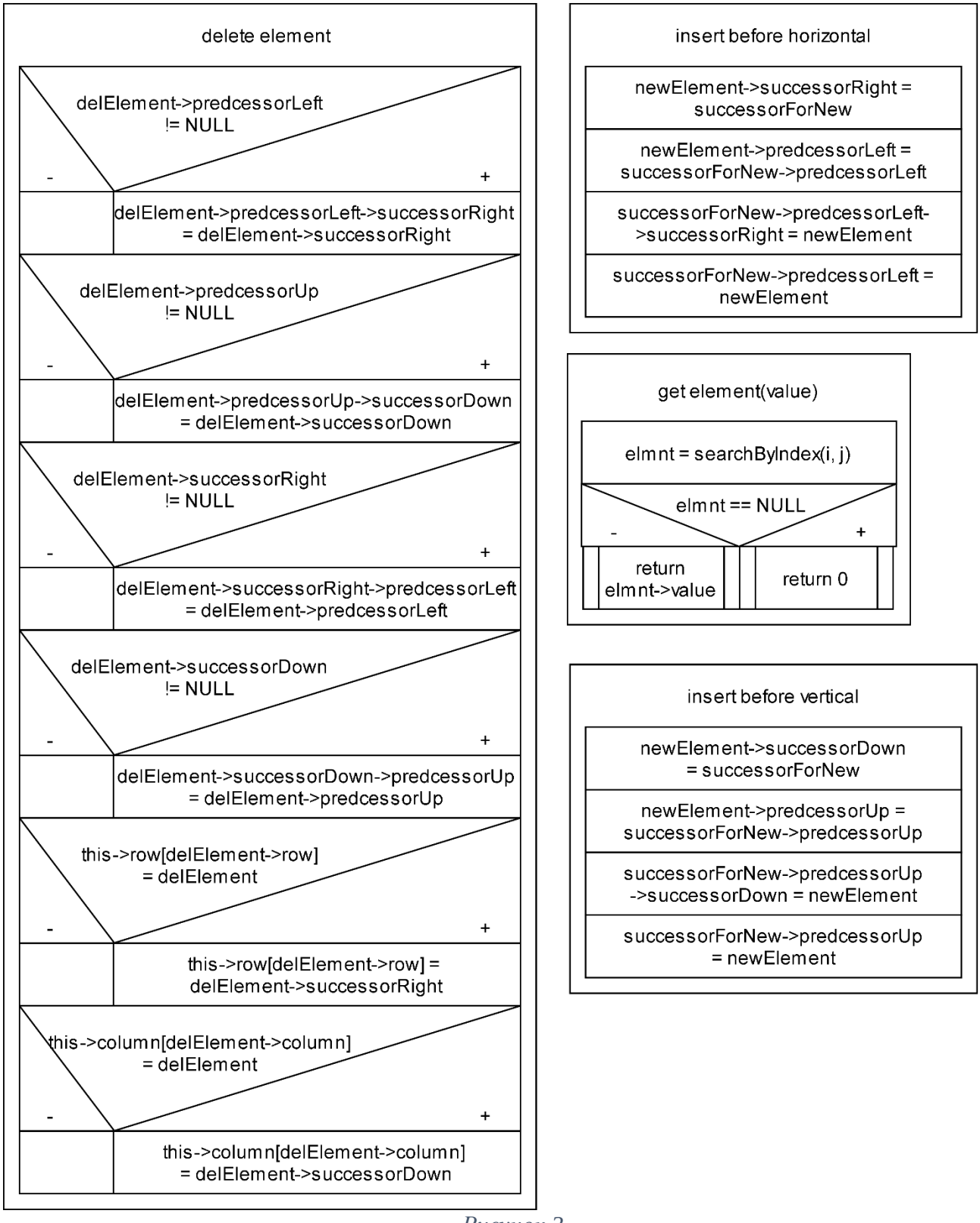
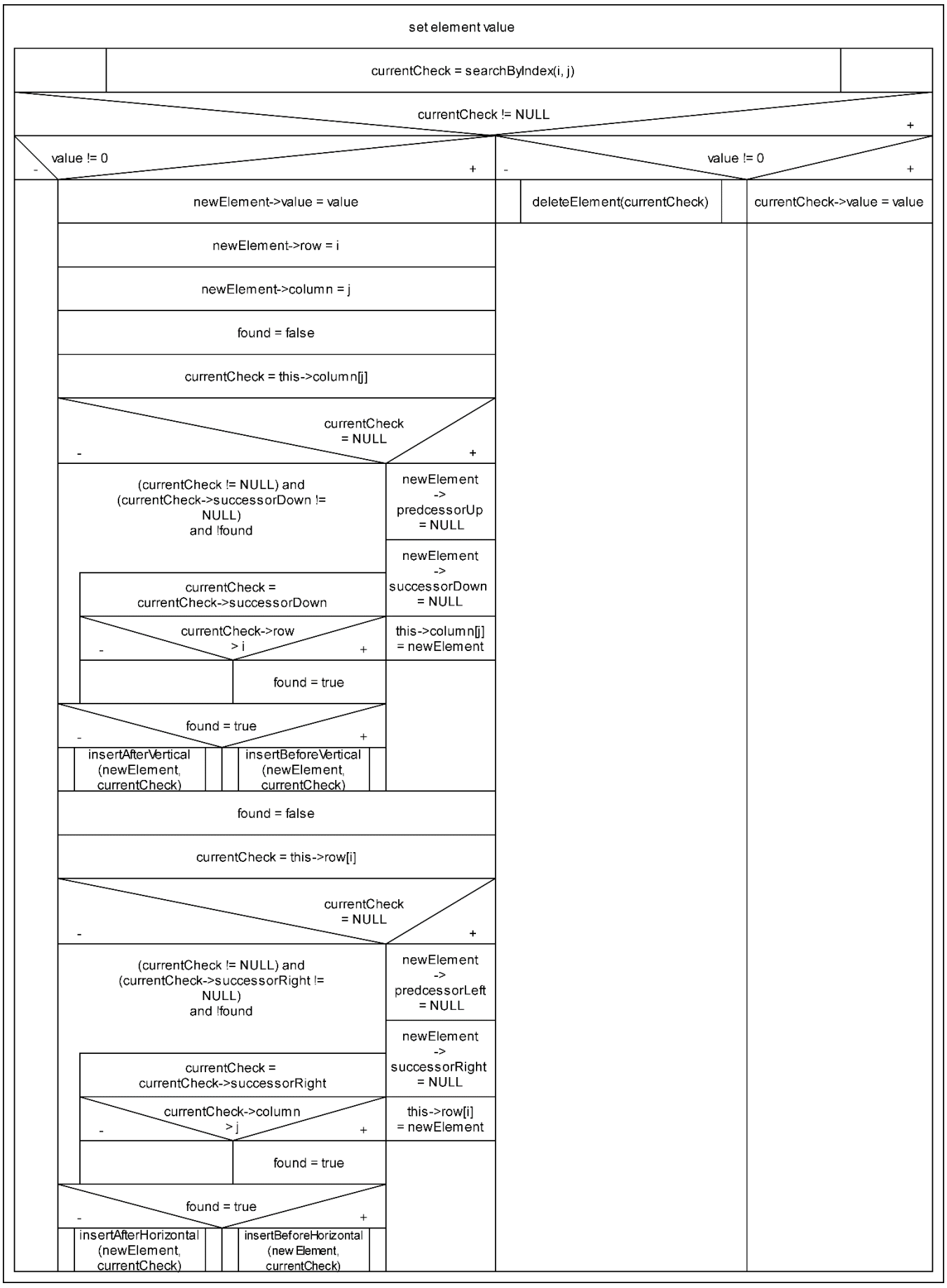

delete element
delElement->predcessorLeft != NULL
-
+
delElement->predcessorLeft->successorRight = delElement->successorRight
delElement->predcessorUp != NULL
-
+
delElement->predcessorUp->successorDown = delElement->successorDown
delElement->successorRight != NULL
-
+
delElement->successorRight->predcessorLeft = delElement->predcessorLeft
delElement->successorDown != NULL
-
+
delElement->successorDown->predcessorUp = delElement->predcessorUp
this->row[delElement->row] = delElement
-
+
this->row[delElement->row] = delElement->successorRight
this->column[delElement->column] = delElement
-
+
this->column[delElement->column] = delElement->successorDown
insert before horizontal
newElement->successorRight = successorForNew
newElement->predcessorLeft = successorForNew->predcessorLeft
successorForNew->predcessorLeft->successorRight = newElement
successorForNew->predcessorLeft = newElement
get element(value)
elmnt = searchByIndex(i, j)
elmnt == NULL
-
+
return elmnt->value
return 0
insert before vertical
newElement->successorDown = successorForNew
newElement->predcessorUp = successorForNew->predcessorUp
successorForNew->predcessorUp ->successorDown = newElement
successorForNew->predcessorUp = newElement

Рисунок 3


set element value
currentCheck = searchByIndex(i, j)
currentCheck != NULL
+
value != 0
-
+
value != 0
-
+
newElement->value = value
deleteElement(currentCheck)
currentCheck->value = value
newElement->row = i
newElement->column = j
found = false
currentCheck = this->column[j]
currentCheck = NULL
-
+
(currentCheck != NULL) and (currentCheck->successorDown != NULL) and !found
newElement -> predcessorUp = NULL
newElement -> successorDown = NULL
currentCheck = currentCheck->successorDown
currentCheck->row > i
-
+
this->column[j] = newElement
found = true
found = true
-
+
insertAfterVertical (newElement, currentCheck)
insertBeforeVertical (newElement, currentCheck)
found = false
currentCheck = this->row[i]
currentCheck = NULL
-
+
(currentCheck != NULL) and (currentCheck->successorRight != NULL) and !found
newElement -> predcessorLeft = NULL
newElement -> successorRight = NULL
currentCheck = currentCheck->successorRight
currentCheck->column > j
-
+
this->row[i] = newElement
found = true
found = true
-
+
insertAfterHorizontal (newElement, currentCheck)
insertBeforeHorizontal (newElement, currentCheck)

5. Текст програми

“sparseMatrix.h”

```
#include <cstdlib>

struct element {//non zero
    double value;
    element* successorRight, *successorDown, *predcessorLeft, *predcessorUp;
    int row, column;
};

class sparseMatrix {
public:
    sparseMatrix(int N, int M) {
        this->rowsAmount = N;
        this->columnsAmount = M;
        row = new element*[N];
        for (int i = 0; i < N; i++)
            row[i] = NULL;
        column = new element*[M];
        for (int i = 0; i < M; i++)
            column[i] = NULL;
    }
    /*~sparseMatrix() {
        for (int i = 0; i < this->rowsAmount; i++) {
            element* currentCheck = row[i];
            if (currentCheck != NULL) {
                while (currentCheck->successorRight != NULL) {
                    currentCheck = currentCheck->successorRight;
                        deleteElement(currentCheck->predcessorLeft);//Exception thrown: read access
violation. currentCheck was 0xDDDDDDDD.
                }
                delete currentCheck;
            }
        }
    }*/

    void generate();
    void printNormal();
    void printSparse();
    void setElement(double value, int i, int j);
    double getElement(int i, int j);
    void deleteByValue(double value);
    sparseMatrix operator+ (sparseMatrix& matrix2);
    sparseMatrix& operator= (sparseMatrix matrix2);

private:
    int rowsAmount, columnsAmount;
```

```
    element** row, **column;

    element* searchByValue(double value);//returns pointer to first element with this value(top to down, left to right search), NULL if none exist or value = 0
    element* searchByIndex(int row, int column);//works as get
    void deleteElement(element* delElement);
    void insertBeforeVertical(element* newElement, element* successorForNew);
    void insertBeforeHorizontal(element* newElement, element* successorForNew);
    void insertAfterVertical(element* newElement, element* predcessorForNew);
    void insertAfterHorizontal(element* newElement, element* predcessorForNew);
};
```

“sparseMatrix.cpp”

```
#include "sparseMatrix.h"
#include <cstdlib>
#include <iostream>
#include <iomanip>
#include <random>
#include <chrono>

void sparseMatrix::generate() {
    typedef std::chrono::high_resolution_clock myclock;
    myclock::time_point beginning = myclock::now();
    int maxNonZero = static_cast<int>(floor(0.3 * this->rowsAmount * this->columnsAmount));
    std::uniform_int_distribution<int> coinflip(0, 1);
    std::uniform_int_distribution<int> rDistribution(0, this->rowsAmount - 1);
    std::uniform_int_distribution<int> cDistribution(0, this->columnsAmount - 1);
    std::uniform_real_distribution<double> vDistribution(-32768.0, 32767.0);
    myclock::duration d = myclock::now() - beginning;
    unsigned seed = d.count();
    std::default_random_engine generator(seed);
    for (int i = 0; i < maxNonZero; i++) {
        bool zero = coinflip(generator);//0 or 1
        if (!zero) {
            double val = vDistribution(generator);
            bool gen = false;
            int row, column;
            do {
                row = rDistribution(generator);
                column = cDistribution(generator);
                if (this->searchByIndex(row, column) == NULL)
                    gen = true;
            } while (!gen);
            this->setElement(val, row, column);
        }
    }
    /*for (int i = 0; i < maxNonZero; i++) {
        bool zero = rand() / INT_MAX;//0 or 1
        if (!zero) {
            double val = rand();
            bool gen = false;
            int row, column;
            do {
                row = rand() % this->rowsAmount;
                column = rand() % this->columnsAmount;
                if (this->searchByIndex(row, column) != NULL)
                    gen = true;
            } while (!gen);
            this->setElement(val, row, column);
```

```
        }
    }*/
}

void sparseMatrix::setElement(double value, int i, int j) {//????????

    element* currentCheck = this->searchByIndex(i, j);
    if (currentCheck != NULL) {
        if(value != 0)
            currentCheck->value = value;
        else
            this->deleteElement(currentCheck);
    }
    else {//connect with *successorRight, *successorDown, *predcessorLeft, *predcessorUp
        if (value != 0) {
            element* newElement = new element;
            newElement->value = value;
            newElement->row = i;
            newElement->column = j;//
            bool found = false;
            //find first with bigger i
            currentCheck = this->column[j];
            if (currentCheck == NULL) {
                newElement->predcessorUp = NULL;
                newElement->successorDown = NULL;
                this->column[j] = newElement;
            }
            else {
                while ((currentCheck != NULL) && (currentCheck->successorDown != NULL) &&
!found) {
                    currentCheck = currentCheck->successorDown;//next row
                    if (currentCheck->row > i)
                        found = true;
                }
                if (found)
                    insertBeforeVertical(newElement, currentCheck);
                else
                    insertAfterVertical(newElement, currentCheck);
            }
            //find first with bigger j
            found = false;
            currentCheck = this->row[i];
            if (currentCheck == NULL) {
                newElement->predcessorLeft = NULL;
                newElement->successorRight = NULL;
                this->row[i] = newElement;
            }
```

```
        else {
            while ((currentCheck != NULL) && (currentCheck->successorRight != NULL) &&
!found) {
                currentCheck = currentCheck->successorRight;//next row
                if (currentCheck->column > j)
                    found = true;
            }
            if (found)
                insertBeforeHorizontal(newElement, currentCheck);
            else
                insertAfterHorizontal(newElement, currentCheck);
        }
    }
  }
}

element* sparseMatrix::searchByIndex(int i, int j) {
  if ((this->row[i] == NULL) || (this->column[j] == NULL))
    return NULL;
  bool exists = false;
  element* currentCheck;
  if (i < j) {
    currentCheck = this->column[j];
    if (currentCheck->row == i)
      exists = true;
    while ((currentCheck != NULL) && (currentCheck->successorDown != NULL) && !exists) {
      currentCheck = currentCheck->successorDown;//next row
      if (currentCheck->row == i)
        exists = true;
    }
  }
  else {
    currentCheck = this->row[i];
    if (currentCheck->column == j)
      exists = true;
    while ((currentCheck != NULL) && (currentCheck->successorRight != NULL) && !exists) {
      currentCheck = currentCheck->successorRight;//next row
      if (currentCheck->column == j)
        exists = true;
    }
  }

  if (!exists)
    currentCheck = NULL;
  return currentCheck;
}
```

```
void sparseMatrix::printNormal() {
    element* current;
    std::cout << std::left;
    for (int i = 0; i < this->rowsAmount; i++) {
        for (int j = 0; j < this->columnsAmount; j++) {
            current = searchByIndex(i, j);
            if (current == NULL)
                std::cout << std::setw(14) << "0";
            else
                std::cout << std::setw(14) << std::setprecision(6) << std::fixed << current->value;
            std::cout << " ";
        }
        std::cout << std::endl;
    }
    std::cout << std::right;
}

void sparseMatrix::printSparse() {
    element* current;
    for (int i = 0; i < rowsAmount; i++) {
        current = row[i];
        while (current != NULL) {
            std::cout << "r = " << current->row << ";";
            std::cout << "c = " << current->column << ";";
            std::cout << "v = " << current->value << ";";
            std::cout << std::endl;
            current = current->successorRight;
        }
    }
}

void sparseMatrix::insertBeforeVertical(element* newElement, element* successorForNew) {
    newElement->successorDown = successorForNew;
    newElement->predcessorUp = successorForNew->predcessorUp;
    successorForNew->predcessorUp->successorDown = newElement;
    successorForNew->predcessorUp = newElement;
}

void sparseMatrix::insertBeforeHorizontal(element* newElement, element* successorForNew) {
    newElement->successorRight = successorForNew;
    newElement->predcessorLeft = successorForNew->predcessorLeft;
    successorForNew->predcessorLeft->successorRight = newElement;
    successorForNew->predcessorLeft = newElement;
}

void sparseMatrix::insertAfterVertical(element* newElement, element* predcessorForNew) {
    newElement->predcessorUp = predcessorForNew;
```

```
    newElement->successorDown = predcessorForNew->successorDown;
    if(predcessorForNew->successorDown != NULL)
        predcessorForNew->successorDown->predcessorUp = newElement;
    predcessorForNew->successorDown = newElement;
}

void sparseMatrix::insertAfterHorizontal(element* newElement, element* predcessorForNew) {
    newElement->predcessorLeft = predcessorForNew;
    newElement->successorRight = predcessorForNew->successorRight;
    if (predcessorForNew->successorRight != NULL)
        predcessorForNew->successorRight->predcessorLeft = newElement;
    predcessorForNew->successorRight = newElement;
}

void sparseMatrix::deleteElement(element* delElement) {
    if(delElement->predcessorLeft != NULL)
        delElement->predcessorLeft->successorRight = delElement->successorRight;
    if(delElement->predcessorUp != NULL)
        delElement->predcessorUp->successorDown = delElement->successorDown;
    if(delElement->successorRight != NULL)
        delElement->successorRight->predcessorLeft = delElement->predcessorLeft;
    if(delElement->successorDown != NULL)
        delElement->successorDown->predcessorUp = delElement->predcessorUp;
    if (this->row[delElement->row] == delElement)
        this->row[delElement->row] = delElement->successorRight;
    if (this->column[delElement->column] == delElement)
        this->column[delElement->column] = delElement->successorDown;
    delete delElement;
}

double sparseMatrix::getElement(int i, int j) {
    element* elmnt = searchByIndex(i, j);
    if (elmnt == NULL)
        return 0;
    return elmnt->value;
}

sparseMatrix sparseMatrix::operator+ (sparseMatrix& matrix2) {
    sparseMatrix* matrixResulting = new sparseMatrix(this->rowsAmount, this->columnsAmount);
    double a3;
    for (int i = 0; i < matrixResulting->rowsAmount; i++) {
        for (int j = 0; j < matrixResulting->columnsAmount; j++) {
            a3 = this->getElement(i, j) + matrix2.getElement(i, j);
            matrixResulting->setElement(a3, i, j);
        }
    }
    return *matrixResulting;
```

```
}

sparseMatrix& sparseMatrix::operator= (sparseMatrix matrix2) {
    this->rowsAmount = matrix2.rowsAmount;
    this->columnsAmount = matrix2.columnsAmount;
    for (int i = 0; i < this->rowsAmount; i++)
        row[i] = NULL;
    for (int i = 0; i < this->columnsAmount; i++)
        column[i] = NULL;
    element* currentDonator;
    for (int i = 0; i < rowsAmount; i++) {
        currentDonator = matrix2.row[i];
        while (currentDonator != NULL) {
            this->setElement(currentDonator->value, currentDonator->row, currentDonator->column);
            currentDonator = currentDonator->successorRight;
        }
    }
    return *this;
}

element* sparseMatrix::searchByValue(double value) {
    if (value == 0)
        return NULL;
    bool found = false;
    element* currentCheck = this->row[0];//
    for (int i = 0; (i < this->rowsAmount) && (found == false); i++) {
        currentCheck = this->row[i];
        if (currentCheck != NULL) {
            if (fabs(currentCheck->value - value) <= pow(10,-6))
                found = true;
            while ((currentCheck->successorRight != NULL) && (found == false)) {
                currentCheck = currentCheck->successorRight;
                if (fabs(currentCheck->value - value) <= pow(10, -6))
                    found = true;
            }
        }
    }
    if (!found)
        currentCheck = NULL;
    return currentCheck;
}

void sparseMatrix::deleteByValue(double value) {
    element* delElement = this->searchByValue(value);
    if (delElement != NULL)
        this->deleteElement(delElement);
}
```

“Приклад програми яка демонструє роботу зі структурою”

```
#include <iostream>
#include "sparseMatrix.h"

int main()
{
    std::cout << "Matrix1:" << std::endl;
    sparseMatrix sp(5, 5);
    sp.generate();
    std::cout << std::endl;
    sp.printSparse();
    std::cout << std::endl;
    sp.printNormal();

    std::cout << "Matrix2:" << std::endl;
    sparseMatrix sp1(5, 5);
    sp1.generate();
    std::cout << std::endl;
    sp1.printSparse();
    std::cout << std::endl;
    sp1.printNormal();

    std::cout << "Matrix1 + Matrix2 =" << std::endl;
    sparseMatrix sp2(5, 5);
    sp2 = sp1 + sp;
    std::cout << std::endl;
    sp2.printSparse();
    std::cout << std::endl;
    sp2.printNormal();

    double value;
    std::cout << "input value to delete: ";
    std::cin >> value;
    sp2.deleteByValue(value);

    std::cout << std::endl;
    sp2.printSparse();
    std::cout << std::endl;
    sp2.printNormal();

    /*do {
        int i, j, val;
        std::cout << "i = ";
        std::cin >> i;
        std::cout << "j = ";
        std::cin >> j;
        std::cout << "val = ";
```

```
        std::cin >> val;
        sp.setElement(val, i, j);
        system("cls");
        sp.printSparse();
        std::cout << std::endl;
        sp.printNormal();
    } while (true);*/

    system("pause");
    return 0;
}
```

6. Контрольний приклад

Початкова матриця:

0; -7; 0; 0;
0; 0; 11; 0;
0; 0; 0; 0

Додавання елементу(заміна/видалення):

$$i = 0; j = 1; value = 0$$

0; 0; 0; 0;
0; 0; 11; 0;
0; 0; 0; 0

$$i = 0; j = 0; value = 3$$

3; 0; 0; 0;
0; 0; 11; 0;
0; 0; 0; 0

$$i = 0; j = 0; value = 7$$

7; 0; 0; 0;
0; 0; 11; 0;
0; 0; 0; 0

Пошук елементу за індексами:

$$i = 1; j = 2$$

7; 0; 0; 0;
0; 0; 11; 0;
0; 0; 0; 0

$$value = 11$$

Видалення елементу за індексами:

$$i = 1; j = 2$$

```
7; 0; 0; 0;
0; 0; 0; 0;
0; 0; 0; 0
```

Додавання матриць:

```
7; 0; 0; 0;
0; 0; 0; 0;
0; 0; 0; 0
```

+

```
2; 0; 0; -8;
0; 13; 0; 0;
0; 0; 0; 0
```

=

```
9; 0; 0; -8;
0; 13; 0; 0;
0; 0; 0; 0
```

Виведення у вигляді розрідженої матриці:

```
9; 0; 0; -8;
0; 13; 0; 0;
0; 0; 0; 0
```

=

Рядок	Стовпець	Значення
0	0	9
0	3	-8
1	1	13

7. Тестування програми

Очікуваний результат програми – прикладу:

- Матриця 1 в обох виглядах, згенерована випадково
- Матриця 2 в обох виглядах, згенерована випадково
- Матриця 3 в обох виглядах, сумма першої та другої матриць
- Матриця 3 із виділеним елементом(значення вводиться користувачем)

Тести розробити неможливо через випадковість генерації, а самостійна ініціалізація матриці заважає тестуванню у неконтрольованих умовах.

```
Matrix1:

r = 1;c = 2;v = 15328.1;
r = 3;c = 2;v = 14402.2;
r = 4;c = 2;v = 17914.7;
r = 4;c = 3;v = -7707.09;
r = 4;c = 4;v = -31356.3;

0               0               0               0               0
0               0               15328.141845    0               0
0               0               0               0               0
0               0               14402.215586    0               0
0               0               17914.670412    -7707.086542    -31356.302097
Matrix2:

r = 0;c = 1;v = -3149.837156;
r = 2;c = 1;v = -11157.178274;
r = 4;c = 1;v = -32244.297708;

0               -3149.837156    0               0               0
0               0               0               0               0
0               -11157.178274   0               0               0
0               0               0               0               0
0               -32244.297708   0               0               0
Matrix1 + Matrix2 =

r = 0;c = 1;v = -3149.837156;
r = 1;c = 2;v = 15328.141845;
r = 2;c = 1;v = -11157.178274;
r = 3;c = 2;v = 14402.215586;
r = 4;c = 1;v = -32244.297708;
r = 4;c = 2;v = 17914.670412;
r = 4;c = 3;v = -7707.086542;
r = 4;c = 4;v = -31356.302097;

0               -3149.837156    0               0               0
0               0               15328.141845    0               0
0               -11157.178274   0               0               0
0               0               14402.215586    0               0
0               -32244.297708   17914.670412    -7707.086542    -31356.302097
input value to delete: -32244.297708

r = 0;c = 1;v = -3149.837156;
r = 1;c = 2;v = 15328.141845;
r = 2;c = 1;v = -11157.178274;
r = 3;c = 2;v = 14402.215586;
r = 4;c = 2;v = 17914.670412;
r = 4;c = 3;v = -7707.086542;
r = 4;c = 4;v = -31356.302097;

0               -3149.837156    0               0               0
0               0               15328.141845    0               0
0               -11157.178274   0               0               0
0               0               14402.215586    0               0
0               0               17914.670412    -7707.086542    -31356.302097
Press any key to continue . . .
```

Рисунок 4

8. Аналіз результатів тестування і роботи програми

Розроблений клас розрідженої матриці складається з чотирьох полів:

- Кількість рядків
- Кількість стовпців
- Вказівник на масив вказівників на перші елементи стовпців
- Вказівник на масив вказівників на перші елементи рядків

  Можна було зекономити пам'ять ще більше і замість двох масивів виділяти пам'ять тільки на верхній лівий елемент(i = 0, j = 0), але тоді був би дуже сильно ускладнений алгоритм пошуку елементів.

  Також можна було зберігати тільки один масив(або рядків, або стовпців). Але тоді гнучкість пошуку була б трохи меншою, бо в цій реалізації завжди пошук починався б з тільки з однієї з сторін матриці. Це погано коли треба знайти елемент, у якого один індекс сильно більше другого. Наприклад у третьому рядку, шістдесятому стовпці. Так можуть бути ситуації, коли усі елементи з трьох – ненульові, а для стовпця цей елемент – перший ненульовий. Але в середньому пошук з вибором початкової сторони має бути швидший ніж пошук з тільки однієї.

Елемент у свою чергу складається з семи полів:

- Значення
- Рядок
- Стовпець
- Вказівник на лівого ненульового сусіда
- Вказівник на правого ненульового сусіда
- Вказівник на верхнього ненульового сусіда
- Вказівник на нижнього ненульового сусіда

Пошук елемента за індексами виконується наступним чином:

Перевіряється чи вказівники на рядок та стовпець існують, якщо хоча б один з них не існує – усі елементи рядка/стовпця дорівнюють нулю, потрібний елемент відповідно теж нуль. Далі перевіряється чи більше індекс рядків чи стовпців, в залежності від цього шукають по рядкам чи стовпцям відповідно.

Порівняно зі звичайним зберіганням матриці у двовимірному масиві, ця реалізація має ряд переваг:

- По перше це економія пам'яті(зберігаються тільки вказівники на зв'язані списки рядків/стовпців(ТІЛЬКИ НЕНУЛЬОВИХ ЕЛЕМЕНТІВ)), звичайний двовимірний масив виграє тільки в дуже маленьких масивах(наприклад 2*2, 3*3)

  Порівняємо на прикладі цієї матриці:

  ```
  01; 00; 00;
  -3; 00; 00;
  00; 00; 07
  ```

  Для зберігання у звичайному двовимірному масиві потрібно три вказівника на рядки(8 байтів для 64розрядної системи), вказівник на масив вказівників(8 байтів для 64розрядної системи), та дев'ять елементів(8 байтів(double)). В сумі це 104 байти і

цей розмір не буде змінюватись при змінені кількості ненульових елементів.
Для розробленої реалізації схеми Кнута потрібні 6 вказівників на рядки/стовпці(8 байтів для 64розрядної системи) та два поля цілих чисел(4 байти) і три елементи(48 байтів), в сумі це 200 байтів. І це число буде збільшуватись при додаванні ненульових елементів.

Представимо шо ця матриця має розміри 200*200 і такі самі елементи(усі нові - нулі).
Для двовимірного масиву знадобиться 321608 байтів.
Для реалізованої схеми Кнута знадобиться лише 3256 байтів, зі збільшенням розмірів матриці ця різниця буде тільки зростати.

- По друге це швидкість доступу. Так порівняно з двовимірним масивом швидкість доступу до елементу за індексом може бути швидшою у відносно малих матрицях, але із зростанням розмірів схема Кнута виграє. Пошук за значенням швидший у схемі Кнута тому, що не треба ітерувати через нульові значення.

Для покращення можна було також додати пошук знизу та справа(структура елемента це дозволяє), для цього лише потрібно трохи переробити вже існуючу функцію пошуку, та додати в клас матриці ще два масиви з вказівниками на перші елементи з іншої сторони. Але прискорення буде помітне тільки в матрицях великих розмірів, які мають відносно велику кількість ненульових елементів. Але за прискорення потрібно буде витратити дуже багато пам’яті.

Можливо також зробити це не додаючи нові поля – потрібно зробити списки циклічними, таким чином можна буде лівим сусідом самого першого елемента буде останній елемент. Цей варіант набагато краще, бо не потребує додаткової пам’яті(саме такім чином була реалізована перша версія програми).

Загалом можна знайти ще багато варіантів покращення даної реалізації, але це вже буде занадто далеко від схеми Кнута.

Література

//