|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **Министерство науки и высшего образования Российской Федерации**  Калужский филиал  федерального государственного бюджетного  образовательного учреждения высшего образования  ***«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»***  ***(КФ МГТУ им. Н.Э. Баумана)*** |

**ФАКУЛЬТЕТ** ***ИУК «Информатика и управление»***

**КАФЕДРА** \_\_***ИУК4 «Программное обеспечение ЭВМ, информационные технологии»***

**ЛАБОРАТОРНАЯ РАБОТА №4**

**«Обработка разреженных матриц»**

**ДИСЦИПЛИНА: «Типы и структуры данных»**

|  |  |  |
| --- | --- | --- |
| Выполнил: студент гр. ИУК4-42Б | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( Карельский М.К. )  (Подпись) |
| Проверил: | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( Пчелинцева Н.И. )  (Подпись) |
| Дата сдачи (защиты):  Результаты сдачи (защиты): | | |
|  | - Балльная оценка:  - Оценка: | |

Калуга, 2022

**Цель:** формирование практических навыков реализации хранения и обработки разреженных матриц

**Задание:**

1. Познакомиться с понятиями разреженная/плотная матрица.
2. Изучить алгоритмы представления матриц в компактной форме.
3. Научиться реализовывать изученные алгоритмы средствами ООП-технологии.
4. Реализовать операцию над разреженными матрицами согласно варианту.

**Вариант 7**

Разреженная матрица хранится в схеме Кнута. Смоделировать операцию перестановки двух столбцов матрицы с получением результата в том же формате.

**Листинг:**

***Knuth.h***

#pragma once

#include <vector>

#include <string>

namespace LW\_4

{

class Knuth

{

public:

Knuth();

Knuth(std::string inputFile, std::string saveFile = "Output.txt");

void Print();

void Restore();

void ShowNumberAtPosition(uint16\_t i, uint16\_t j);

void ReplaceColumns(uint16\_t firstColumn, uint16\_t secondColumn);

private:

void Save();

std::string \_saveFile{};

std::vector<int> \_an{};

std::vector<uint16\_t> \_i{};

std::vector<uint16\_t> \_j{};

std::vector<uint16\_t> \_nr{};

std::vector<uint16\_t> \_nc{};

std::vector<uint16\_t> \_jr{};

std::vector<uint16\_t> \_jc{};

uint16\_t \_numberOfRows{};

uint16\_t \_numberOfColumns{};

};

}

***Knuth.cpp***

#include "Knuth.h"

#include <fstream>

#include <sstream>

#include <iostream>

namespace LW\_4

{

Knuth::Knuth(): \_saveFile("Output.txt"),

\_an{}, \_i{}, \_j{}, \_nr{}, \_nc{}, \_jr{}, \_jc{},

\_numberOfRows{}, \_numberOfColumns{} {}

Knuth::Knuth(std::string inputFile, std::string saveFile): \_saveFile(saveFile)

{

std::ifstream file(inputFile);

std::string line{};

uint16\_t i = 1;

\_numberOfColumns = 0;

while (std::getline(file, line))

{

uint16\_t j = 1;

bool rowStartIsFounded = false;

std::string number{};

std::istringstream stream(line);

while (stream >> number)

{

if (number != "0")

{

\_an.push\_back(std::stoi(number));

\_i.push\_back(i);

\_j.push\_back(j);

if (j > \_numberOfColumns)

\_numberOfColumns = j;

if (!rowStartIsFounded)

{

\_jr.push\_back(\_i.size());

rowStartIsFounded = true;

}

}

++j;

}

++i;

if (!rowStartIsFounded)

\_jr.push\_back(0);

}

file.close();

\_numberOfRows = \_jr.size();

for (uint16\_t k{}; k < \_i.size() - 1; ++k)

{

if (\_i[k] == \_i[k + 1])

\_nr.push\_back(k + 2);

else

\_nr.push\_back(0);

}

\_nr.push\_back(0);

for (uint16\_t k{}; k < \_j.size() - 1; ++k)

{

uint16\_t l;

for (l = k + 1; l < \_j.size(); ++l)

if (\_j[k] == \_j[l])

{

\_nc.push\_back(l + 1);

break;

}

if (l == \_j.size())

\_nc.push\_back(0);

}

\_nc.push\_back(0);

for (uint16\_t k{}; k < \_numberOfColumns; ++k)

{

uint16\_t l;

for (l = 0; l < \_j.size(); ++l)

{

if (\_j[l] == k + 1)

{

\_jc.push\_back(l + 1);

break;

}

}

if (l == \_j.size())

\_jc.push\_back(0);

}

}

void Knuth::Restore()

{

int\*\* matrix = new int\* [\_numberOfRows] {};

for (uint16\_t i{}; i < \_numberOfRows; ++i)

matrix[i] = new int[\_numberOfColumns] {};

for (uint16\_t i{}; i < \_an.size(); ++i)

matrix[\_i[i] - 1][\_j[i] - 1] = \_an[i];

std::cout << " ";

for (uint16\_t j{}; j < \_numberOfColumns; ++j)

std::cout << j + 1 << " ";

std::cout << "\n\n";

for (uint16\_t i{}; i < \_numberOfRows; ++i)

{

std::cout << i + 1 << " ";

for (uint16\_t j{}; j < \_numberOfColumns; ++j)

std::cout << matrix[i][j] << " ";

std::cout << "\n";

}

for (uint16\_t i{}; i < \_numberOfRows; ++i)

delete[] matrix[i];

delete[] matrix;

}

void Knuth::ShowNumberAtPosition(uint16\_t i, uint16\_t j)

{

uint16\_t k;

for (k = 0; k < \_an.size(); ++k)

if (\_i[k] == i && \_j[k] == j)

{

std::cout << \_an[k];

return;

}

std::cout << 0;

}

void Knuth::Print()

{

std::cout << " ";

for (uint16\_t i{}; i < \_an.size(); ++i)

std::cout << i + 1 << " ";

std::cout << "\n";

std::cout << "AN = ";

for (uint16\_t i{}; i < \_an.size(); ++i)

std::cout << \_an[i] << " ";

std::cout << "\n";

std::cout << "I = ";

for (uint16\_t i{}; i < \_i.size(); ++i)

std::cout << \_i[i] << " ";

std::cout << "\n";

std::cout << "J = ";

for (uint16\_t i{}; i < \_j.size(); ++i)

std::cout << \_j[i] << " ";

std::cout << "\n";

std::cout << "NR = ";

for (uint16\_t i{}; i < \_nr.size(); ++i)

std::cout << \_nr[i] << " ";

std::cout << "\n";

std::cout << "NC = ";

for (uint16\_t i{}; i < \_nc.size(); ++i)

std::cout << \_nc[i] << " ";

std::cout << "\n";

std::cout << "JR = ";

for (uint16\_t i{}; i < \_jr.size(); ++i)

std::cout << \_jr[i] << " ";

std::cout << "\n";

std::cout << "JC = ";

for (uint16\_t i{}; i < \_jc.size(); ++i)

std::cout << \_jc[i] << " ";

std::cout << "\n";

}

void Knuth::ReplaceColumns(uint16\_t firstColumn, uint16\_t secondColumn)

{

if (firstColumn != secondColumn)

{

for (uint16\_t i{}; i < \_numberOfRows; ++i)

{

int firstIndex = 0;

int secondIndex = 0;

for (uint16\_t j{}; j < \_an.size(); ++j)

{

if (\_i[j] == i + 1 && \_j[j] == firstColumn)

firstIndex = j + 1;

if (\_i[j] == i + 1 && \_j[j] == secondColumn)

secondIndex = j + 1;

}

if (firstIndex != 0 && secondIndex != 0)

{

int temp = \_an[firstIndex - 1];

\_an[firstIndex - 1] = \_an[secondIndex - 1];

\_an[secondIndex - 1] = temp;

}

else if (firstIndex != 0 && firstIndex < \_an.size())

{

uint16\_t analysedIndex = firstIndex + 1;

while (\_i[firstIndex - 1] == \_i[analysedIndex - 1] &&

\_j[analysedIndex - 1] < secondColumn &&

analysedIndex - 1 < \_an.size())

{

int temp = \_an[firstIndex - 1];

\_an[firstIndex - 1] = \_an[analysedIndex - 1];

\_an[analysedIndex - 1] = temp;

temp = \_j[analysedIndex - 2];

\_j[analysedIndex - 2] = \_j[analysedIndex - 1];

\_j[analysedIndex - 1] = temp;

++analysedIndex;

}

\_j[analysedIndex - 2] = secondColumn;

}

else if (secondIndex != 0 && secondIndex > 1)

{

int16\_t analysedIndex = secondIndex - 1;

while (\_i[secondIndex - 1] == \_i[analysedIndex - 1] &&

\_j[analysedIndex - 1] > firstColumn &&

analysedIndex > 0)

{

int temp = \_an[secondIndex - 1];

\_an[secondIndex - 1] = \_an[analysedIndex - 1];

\_an[analysedIndex - 1] = temp;

temp = \_j[analysedIndex];

\_j[analysedIndex] = \_j[analysedIndex - 1];

\_j[analysedIndex - 1] = temp;

--analysedIndex;

}

\_j[analysedIndex] = firstColumn;

}

}

\_nc.clear();

for (uint16\_t i{}; i < \_j.size() - 1; ++i)

{

uint16\_t j;

for (j = i + 1; j < \_j.size(); ++j)

if (\_j[i] == \_j[j])

{

\_nc.push\_back(j + 1);

break;

}

if (j == \_j.size())

\_nc.push\_back(0);

}

\_nc.push\_back(0);

\_jc.clear();

for (uint16\_t i{}; i < \_numberOfColumns; ++i)

{

uint16\_t j;

for (j = 0; j < \_j.size(); ++j)

{

if (\_j[j] == i + 1)

{

\_jc.push\_back(j + 1);

break;

}

}

if (j == \_j.size())

\_jc.push\_back(0);

}

}

Save();

}

void Knuth::Save()

{

std::ofstream file(\_saveFile);

file << "AN = ";

for (uint16\_t i{}; i < \_an.size(); ++i)

file << \_an[i] << " ";

file << "\n";

file << "I = ";

for (uint16\_t i{}; i < \_i.size(); ++i)

file << \_i[i] << " ";

file << "\n";

file << "J = ";

for (uint16\_t i{}; i < \_j.size(); ++i)

file << \_j[i] << " ";

file << "\n";

file << "NR = ";

for (uint16\_t i{}; i < \_nr.size(); ++i)

file << \_nr[i] << " ";

file << "\n";

file << "NC = ";

for (uint16\_t i{}; i < \_nc.size(); ++i)

file << \_nc[i] << " ";

file << "\n";

file << "JR = ";

for (uint16\_t i{}; i < \_jr.size(); ++i)

file << \_jr[i] << " ";

file << "\n";

file << "JC = ";

for (uint16\_t i{}; i < \_jc.size(); ++i)

file << \_jc[i] << " ";

file << "\n";

file.close();

}

}

***Menu.h***

#pragma once

#include "Knuth.h"

namespace LW\_4

{

class Menu

{

public:

Menu();

Menu(std::string inputFile, std::string saveFile = "Output.txt");

~Menu();

void Run();

private:

void Pause();

Knuth \_knuth{};

};

}

***Menu.cpp***

#include "Menu.h"

#include <iostream>

namespace LW\_4

{

Menu::Menu() : \_knuth() {}

Menu::Menu(std::string inputFile, std::string saveFile) :

\_knuth(inputFile, saveFile) {}

Menu::~Menu() { remove("pauseTemp"); }

void Menu::Run()

{

uint16\_t command = 1;

while (command != 0)

{

\_knuth.Restore();

std::cout << "\n";

\_knuth.Print();

std::cout << "\n";

std::cout << "1. Replace columns\n";

std::cout << "2. Show number at position\n";

std::cout << "0. Exit\n";

std::cout << ">>> ";

std::cin >> command;

std::cout << "\n";

if (command == 1)

{

std::cout << "Input first column: ";

uint16\_t firstColumn;

std::cin >> firstColumn;

std::cout << "Input second column: ";

uint16\_t secondColumn;

std::cin >> secondColumn;

\_knuth.ReplaceColumns(firstColumn, secondColumn);

}

else if (command == 2)

{

std::cout << "i: ";

uint16\_t i;

std::cin >> i;

std::cout << "j: ";

uint16\_t j;

std::cin >> j;

std::cout << "Number: ";

\_knuth.ShowNumberAtPosition(i, j);

std::cout << "\n";

Pause();

}

system("cls");

}

}

void Menu::Pause()

{

system("pause>pauseTemp");

remove("pauseTemp");

}

}

***Main.cpp***

#include "Menu.h"

int main()

{

LW\_4::Menu menu("Input.txt");

menu.Run();

return 0;

}

**Результат:**

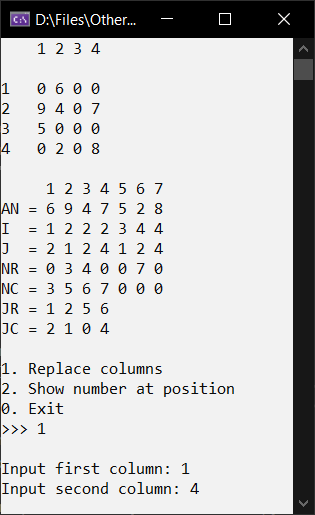
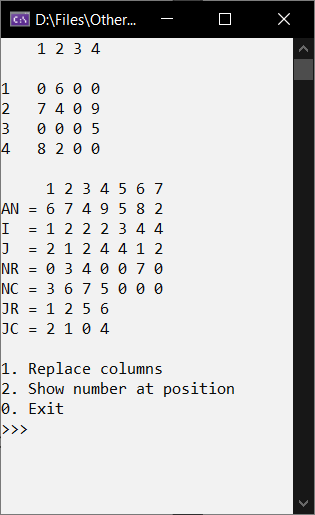
 

Рис. 1. Результат

**Вывод:** в ходе выполнения лабораторной работы были получены навыки реализации хранения разреженной матрицы в схеме Кнута, операции перестановки столбцов, восстановления нормального вида матрицы.