**Лабораторная робота №1**

Моделирование представления в памяти таблиц

**Цель работы:** приобретение и закрепление навыков размещения в памяти таблиц. Получение начальных представлений о модульности программы с точки зрения обрабатываемых данных.

*Задание*

Разработать способ экономного размещения в памяти заданной разреженной таблицы. Разработать процедуры/функции, обеспечивающие доступ к элементам таблицы по номерам строки и имени столбца. В контрольной программе обеспечить запись и чтение всех записей таблицы. Произвести хронометраж выполнения операций чтения и записи элементов в массивы.



**Решение**

1. **Текст программы**

***main.cpp:***

#include <iostream>

#include <fstream>

#include <vector>

#include <intrin.h>

#include "SparseMatrix.hpp"

typedef std::vector<std::vector<int>> VecMatrix;

template<typename F>

unsigned long long measure(F&& f, const int n = 100);

VecMatrix getVecMatrix();

SparseMatrix compress(VecMatrix& matrix);

VecMatrix uncompress(SparseMatrix& matrix);

void printMatrix(VecMatrix& matrix);

void printMatrix(SparseMatrix& matrix);

void printCompressedMatrix(SparseMatrix& matrix);

void measureAll(VecMatrix vecMatrix, SparseMatrix sparseMatrix);

int main() {

auto vecMatrix = getVecMatrix();

auto compressed = compress(vecMatrix);

auto uncompressed = uncompress(compressed);

std::cout << "= = = Matrix = = =" << std::endl;

printMatrix(compressed);

std::cout << "= = = Compressed Matrix = = =" << std::endl;

printCompressedMatrix(compressed);

measureAll(vecMatrix, compressed);

system("pause");

return 0;

}

void measureAll(VecMatrix vecMatrix, SparseMatrix sparseMatrix) {

int m = vecMatrix.size();

int n = vecMatrix[0].size();

int len = m \* n;

auto timeToReadVecMatrix = measure([&vecMatrix, m, n] {

int val;

for (int i = 0; i < m; i++) {

for (int j = 0; j < n; j++) val = vecMatrix[i][j];

}

}) / len;

auto timeToWriteVecMatrix = measure([&vecMatrix, m, n] {

for (int i = 0; i < m; i++) {

for (int j = 0; j < n; j++) vecMatrix[i][j] = 1;

}

}) / len;

auto timeToReadSparseMatrix = measure([&sparseMatrix, m, n] {

int val;

for (int i = 0; i < m; i++) {

for (int j = 0; j < n; j++) val = sparseMatrix.get(i, j);

}

}) / len;

auto timeToWriteSparseMatrix = measure([&sparseMatrix, m, n] {

for (int i = 0; i < m; i++) {

for (int j = 0; j < n; j++) sparseMatrix.set(i, j, 1);

}

}) / len;

std::cout << "timeToReadVecMatrix: " << timeToReadVecMatrix << std::endl;

std::cout << "timeToWriteVecMatrix: " << timeToWriteVecMatrix << std::endl;

std::cout << "timeToReadSparseMatrix: " << timeToReadSparseMatrix << std::endl;

std::cout << "timeToWriteSparseMatrix: " << timeToWriteSparseMatrix << std::endl;

}

SparseMatrix compress(VecMatrix& matrix) {

int m = matrix.size();

int n = matrix[0].size();

SparseMatrix compressed(m, n);

for (int i = 0; i < m; i++) {

for (int j = 0; j < n; j++) {

compressed.set(i, j, matrix[i][j]);

}

}

return compressed;

}

VecMatrix uncompress(SparseMatrix& matrix) {

int m = matrix.getM();

int n = matrix.getN();

VecMatrix uncompressed(m, std::vector<int>(n));

for (int i = 0; i < m; i++) {

for (int j = 0; j < n; j++) {

uncompressed[i][j] = matrix.get(i, j);

}

}

return uncompressed;

}

VecMatrix getVecMatrix() {

std::ifstream fin("input.txt");

int m, n;

fin >> m >> n;

VecMatrix matrix(m, std::vector<int>(n));

for (auto& row : matrix) {

for (auto& item : row) fin >> item;

}

fin.close();

return matrix;

}

void printMatrix(VecMatrix& matrix) {

for (auto row : matrix) {

for (auto item : row) {

std::cout << item << " ";

}

std::cout << std::endl;

}

std::cout << std::endl;

}

void printMatrix(SparseMatrix& matrix) {

for (int i = 0; i < matrix.getM(); i++) {

for (int j = 0; j < matrix.getN(); j++) {

std::cout << matrix.get(i, j) << " ";

}

std::cout << std::endl;

}

std::cout << std::endl;

}

void printCompressedMatrix(SparseMatrix& matrix) {

auto compressed = matrix.\_\_getCompressedMatrix();

for (auto item : compressed) {

std::cout << item << " ";

}

std::cout << std::endl << std::endl;

}

template<typename F>

unsigned long long measure(F&& f, const int n) {

unsigned long long res = UINT64\_MAX;

for (int i = 0; i < n; i++) {

auto start = \_\_rdtsc();

f();

auto time = \_\_rdtsc() - start;

if (time < res) res = time;

}

return res;

}

***SparseMatrix.hpp***

#pragma once

#include <vector>

class SparseMatrix {

public:

typedef unsigned int ui;

SparseMatrix(ui m, ui n);

int get(ui i, ui j);

void set(ui i, ui j, int val);

int getM();

int getN();

~SparseMatrix();

private:

int m\_m;

int m\_n;

std::vector<int> m\_compressedMatrix;

int \_getInternalIndex(ui i, ui j);

bool \_isZiroItem(ui i, ui j);

void \_checkIndexes(ui i, ui j);

};

***SparseMatrix.cpp***

#include "SparseMatrix.hpp"

SparseMatrix::SparseMatrix(ui m, ui n) :

m\_m(m), m\_n(n),

m\_compressedMatrix((m / 2)\* n)

{

}

int SparseMatrix::get(ui i, ui j) {

\_checkIndexes(i, j);

if (\_isZiroItem(i, j)) return 0;

return m\_compressedMatrix[\_getInternalIndex(i, j)];

}

void SparseMatrix::set(ui i, ui j, int val) {

\_checkIndexes(i, j);

if (\_isZiroItem(i, j)) return;

m\_compressedMatrix[\_getInternalIndex(i, j)] = val;

}

inline int SparseMatrix::\_getInternalIndex(ui i, ui j) {

return i \* m\_n + j;

}

inline bool SparseMatrix::\_isZiroItem(ui i, ui j) {

return i >= m\_m / 2;

}

inline void SparseMatrix::\_checkIndexes(ui i, ui j) {

if (i >= 0 && j >= 0 && i < m\_m && j < m\_n) {

return;

}

throw std::runtime\_error("SparseMatrix subscript out of range");

}

***input.txt***

6 9

1 2 3 5 6 4 5 5 8

4 5 7 8 2 0 5 4 7

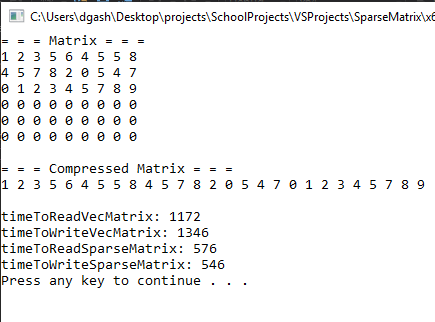
0 1 2 3 4 5 7 8 9

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

1. **Результат выполнения программы**



**Итоги:** на это лабораторной работе я приобрел и закрепил навыки размещения в памяти таблиц. Получил начальные представления о модульности программы с точки зрения обрабатываемых данных.