МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Вятский государственный университет»

Факультет автоматики и вычислительной техники

Кафедра электронных вычислительных машин

**«Создание оконных приложений»**

Отчет по лабораторной работе № 3 по дисциплине

«Технологии программирования»

Выполнил студент группы: ПИб-2301\_\_\_\_\_\_\_\_\_\_\_\_\_ /Морозов Д.А.

Проверил преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Долженкова М.Л.

**Киров 2023**

**Цели работы:**

* Согласовать задание с преподавателем  и разработать оконное приложение на основе функций WinAPI;
* Проанализировать особенности организации процесса выполнения задачи.

**Задание:**

Разработать оконное приложение, в котором на главной форме задается матрица смежности графа и две вершины, а на модульной дочерней форме выводится кратчайший путь от одной выбранной вершины до другой.

**Словесное описание алгоритма:**

При запуске приложения, пользователю доступна главная форма, на которой расположено поле с шаблоном пустой матрицы 6 на 6, поля ввода вершин, а так же статический текст, упрощающий работу с формой.

Пользователь вручную редактирует матрицу смежности, после чего вызывает дочернее модульное окно с ответом путем нажатия на кнопку.

Алгоритм раскраски графа, реализованный в данной программе, является жадным алгоритмом. Он основан на следующей идее:

1. Инициализация всех вершин графа без цвета.
2. Окрашивание первой вершины выбранным цветом.
3. Для каждой оставшейся вершины:
   * Находим доступные цвета, которые не используются среди соседних вершин.
   * Выбираем первый доступный цвет и окрашиваем вершину выбранным цветом.
4. Подсчитываем минимальное количество необходимых цветов для раскраски графа.

Программа считывает матрицу смежности, введенную пользователем, и проверяет ее корректность. Затем она применяет алгоритм раскраски графа и выводит минимальное количество необходимых цветов на дочернем окне.

Обработка графического интерфейса происходит в функциях SoftwareMainProcedure() и SoftwareChildProcedure(), которые являются обработчиками сообщений главного окна и дочернего окна соответственно.

**Листинг кода:**

main.cpp

#include "wndMain.h"

//в данном приложении матрица имеет строго фиксированный размер 6 на 6,

//а так же граф является невзвешенным

//точка входа для программы

int WINAPI WinMain(HINSTANCE hInst, HINSTANCE hPrevInst, LPSTR args, int ncmdshow) {

//инициализация класса главного окна

WNDCLASS SoftwareMainClass = NewWindowClass((HBRUSH)COLOR\_WINDOW, LoadCursor(NULL, IDC\_HAND), hInst, LoadIcon(NULL, IDI\_QUESTION),

L"MainWndClass", SoftwareMainProcedure);

//инициализация окна дочернего

WNDCLASS wc = NewWindowChildClass((HBRUSH)GetStockObject(WHITE\_BRUSH), NULL, NULL, NULL, L"ChildWndClass", SoftwareChildProcedure);

if (!RegisterClass(&wc)) {

MessageBox(hWnd, L"Не удалось зарегистрировать класс дочернего окна", L"Ошибка", MB\_OK | MB\_ICONERROR);

return -1;

}

if (!RegisterClassW(&SoftwareMainClass)) { return -1; }

MSG SoftwareMainMessage = { 0 };

//создание главного окна с заданным классом

hWnd = CreateWindowA("MainWndClass", "Ввод матрицы", WS\_OVERLAPPEDWINDOW | WS\_VISIBLE, 100, 100, 350, 220, NULL, NULL, NULL, NULL);

//цикл обработки сообщений

while (GetMessage(&SoftwareMainMessage, NULL, NULL, NULL)) {

TranslateMessage(&SoftwareMainMessage);

DispatchMessage(&SoftwareMainMessage);

}

return 0;

}

WNDmain.h

#pragma once

#include<Windows.h>

#include<string>

//макросы для облегчения работы

#define OnButtonClicked 1

#define TextBufferSize 50

#define DlgIndexNumberA 200

#define DlgIndexNumberB 210

char Buffer[TextBufferSize];

int readChar;

//беззнаковое целое число

unsigned numA;

unsigned numB;

//extern "C" int GraphColoring(int(\*)[6]);

extern "C" int GC(int(\*)[6], int numVertices, int\*);

//инициализация оконных переменных для передачи их значений между функциями

HWND childText;

HWND hEditControl;

HWND hStaticControl;

HWND hStaticControlNums;

HWND hNumberAControl;

HWND hNumberBControl;

HWND hWnd;

//инициализация необходимых глобальных переменных

int matrix[6][6];

int colors[6] = { 1, 1, 1, 1, 1, 1 };

std::string StrMat;

std::string path;

//прототипы функций

void MainWndAddWidgets(HWND hWnd);

LRESULT CALLBACK SoftwareChildProcedure(HWND childWindow, UINT msg, WPARAM wp, LPARAM lp);

//конструктор класса главного окна

WNDCLASS NewWindowClass(HBRUSH BGColor, HCURSOR Cursor, HINSTANCE hInst, HICON Icon, LPCWSTR Name, WNDPROC Procedure) {

WNDCLASS NWC = { 0 };

NWC.hCursor = Cursor;

NWC.hIcon = Icon;

NWC.hInstance = hInst;

NWC.lpszClassName = Name;

NWC.hbrBackground = BGColor;

NWC.lpfnWndProc = Procedure;

return NWC;

}

//конструктор класса дочернего окна

WNDCLASS NewWindowChildClass(HBRUSH BGColor, HCURSOR Cursor, HINSTANCE hInst, HICON Icon, LPCWSTR Name, WNDPROC ProcedureChild) {

WNDCLASS NWC = { 0 };

NWC.hCursor = Cursor;

NWC.hIcon = Icon;

NWC.hInstance = hInst;

NWC.lpszClassName = Name;

NWC.hbrBackground = BGColor;

NWC.lpfnWndProc = ProcedureChild;

return NWC;

}

//процедура преобразования текстовой матрицы в матрицу типа int

void ParseMatrix()

{

int k = 0;

for (int i = 0; i < 6; i++) {

for (int j = 0; j < 6; j++) {

if (StrMat[k] == '0') { matrix[i][j] = 0; };

if (StrMat[k] == '1') { matrix[i][j] = 1; };

k++;

}

}

}

//проверка корректности введенных значений

bool ValidateMatrix() {

if (StrMat.size() != 36) {

return false;

}

for (int i = 0; i < 36; i++) {

//если символ не 1 и не 0, то проверка не пройдена

if (StrMat[i] != 48 && StrMat[i] != 49) { return false; };

}

return true;

}

int solveGraphColoring(int matrix[6][6], int numVertices, int\* numColors)

{

// Инициализируем все вершины без цвета

int colors[6];

memset(colors, -1, sizeof(colors));

// Окрасим первую вершину первым цветом

colors[0] = 0;

\*numColors = 1;

// Окрашиваем оставшиеся вершины, используя жадный алгоритм

for (int v = 1; v < numVertices; v++) {

// Найдем доступные цвета для этой вершины

bool available[6];

memset(available, true, sizeof(available));

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

if (matrix[v][i] && colors[i] != -1) {

available[colors[i]] = false;

}

}

// Выбираем первый доступный цвет

int chosenColor;

for (chosenColor = 0; chosenColor < numVertices; chosenColor++) {

if (available[chosenColor]) {

break;

}

}

// Если доступного цвета не найдено, вернем -1 (неверный ввод)

if (chosenColor == numVertices) {

return -1;

}

// Окрашиваем вершину выбранным цветом

colors[v] = chosenColor;

// Обновляем минимальное количество необходимых цветов

if (chosenColor + 1 > \*numColors) {

\*numColors = chosenColor + 1;

}

}

// Возвращаем окрашенный граф

int result = 0;

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

result |= (colors[i] << (i \* 3));

}

return result;

}

int countVertices(const int matrix[6][6]) {

int numVertices = 0;

for (int i = 0; i < 6; i++) {

bool hasNonZeroElement = false;

for (int j = 0; j < 6; j++) {

if (matrix[i][j] != 0) {

hasNonZeroElement = true;

break;

}

}

if (hasNonZeroElement) {

numVertices++;

}

}

return numVertices;

}

//процедура обработки сообщений главного окна

LRESULT CALLBACK SoftwareMainProcedure(HWND hWnd, UINT msg, WPARAM wp, LPARAM lp) {

switch (msg) {

case WM\_COMMAND:

switch (wp)

{

//при нажатии кнопки

case OnButtonClicked:

//получение значений из полей выбора вершин

readChar = GetWindowTextA(hEditControl, Buffer, TextBufferSize);

//замена переносов строк в текстовой матрице

StrMat = Buffer;

StrMat.erase(std::remove(StrMat.begin(), StrMat.end(), '\r'), StrMat.end());

StrMat.erase(std::remove(StrMat.begin(), StrMat.end(), '\n'), StrMat.end());

//запуск проверок корректности ввода

if (ValidateMatrix()) {

SetWindowTextA(hStaticControl, "Матрица задана корректно");

ParseMatrix();

}

else {

SetWindowTextA(hStaticControl, "Матрица задана некорректно!");

};

if (ValidateMatrix()) {

//выключаем главное окно

//это необходимо, так как дочернее окно должно быть модальным

EnableWindow(hWnd, FALSE);

int numVertices = countVertices(matrix);

int result;

if (numVertices == 0) {

result = 0;

}

else {

int numColors = 0;

result = solveGraphColoring(matrix, 6, &numColors);

};

//int result = GraphColoring(matrix);

path = "\nThe minimum number of colors needed is " + std::to\_string(result) + "\n";

//создается дочернее окно

HWND childWindow = CreateWindowEx(

0, L"ChildWndClass", L"Дочернее окно", WS\_OVERLAPPEDWINDOW | WS\_VISIBLE,

300, 300, 300, 150, hWnd, NULL, NULL, NULL

);

//проверка на случай если есть проблемы с созданием дочернего окна

if (childWindow == NULL) {

MessageBox(childWindow, L"Не удалось создать дочернее окно", L"Ошибка", MB\_OK | MB\_ICONERROR);

return -1;

}

}

break;

default: break;

}

break;

//при создании

case WM\_CREATE:

//процедура с созданием необходимых компонентов главного окна

MainWndAddWidgets(hWnd);

break;

//при уничтожении

case WM\_DESTROY:

//этот код передастся в цикл, после чего приложение закроется

PostQuitMessage(0);

break;

default: return DefWindowProc(hWnd, msg, wp, lp);

}

}

//процедура обработки сообщений дочернего окна

LRESULT CALLBACK SoftwareChildProcedure(HWND childWindow, UINT msg, WPARAM wp, LPARAM lp) {

switch (msg) {

case WM\_CREATE: {

//создание статического текста, куда будет записан разультат Дийкстры

if (!IsWindow(childText)) {

childText = CreateWindowA("STATIC", "", WS\_CHILD | WS\_VISIBLE,

0, 0, 300, 300, childWindow, NULL, NULL, NULL);

}

//вывод на дочернее окно ответа

SetWindowTextA(childText, path.c\_str());

break; }

//при закрытии

case WM\_CLOSE:

//включаем главное окно и удаляем дочернее

EnableWindow(hWnd, TRUE);

DestroyWindow(childWindow);

break;

default: return DefWindowProc(childWindow, msg, wp, lp);

}

}

//процедура создания компонентов главного окна

//некоторые компоненты заносятся в переменные

//сделано это для того, чтобы в обработке сообщений главного окна можно было

//использовать функции, изменяющие окна, либо же функции,

//которые получают данные введенные в эти компоненты

void MainWndAddWidgets(HWND hWnd) {

hStaticControl = CreateWindowA("static", "Введите матрицу смежности(6 на 6)", WS\_VISIBLE | WS\_CHILD, 0, 0, 490, 20, hWnd, NULL, NULL, NULL, NULL);

hEditControl = CreateWindowA("edit", "000000\r\n000000\r\n000000\r\n000000\r\n000000\r\n000000", WS\_VISIBLE | WS\_CHILD | ES\_MULTILINE | ES\_NUMBER, 10, 40, 55, 100, hWnd, NULL, NULL, NULL, NULL);

//в таких компонентах как кнопка необходимо указывать идентификаторы для обработки нажатия

//в данном случае это макрос OnButtonClicked со значением 1

CreateWindowA("button", "Посчитать", WS\_VISIBLE | WS\_CHILD | ES\_CENTER, 200, 140, 100, 30, hWnd, (HMENU)OnButtonClicked, NULL, NULL, NULL);

}

**Экранные формы:**

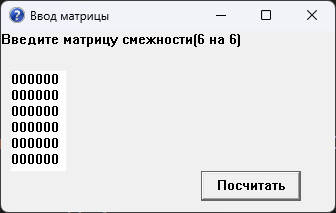


Рис.1 – главное окно при запуске

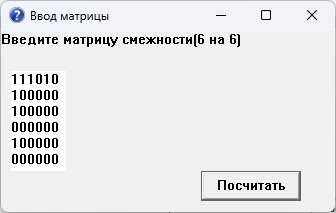


Рис. 2 – главное окно с заданными значениями

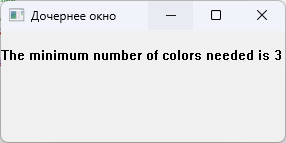


Рис.3 – дочернее окно с ответом

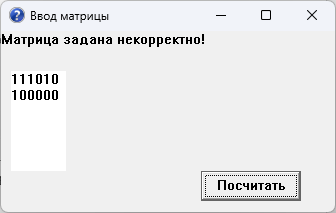


Рис. 4 – обработка некорректно ввода матрицы

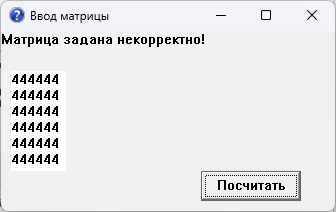


Рис. 5 – обработка некорректно ввода матрицы

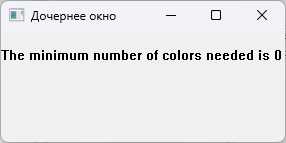


Рис. 6 – ответ в случае, когда матрица заполнена нулями (вершин нет)

**Вывод:**

В процессе выполнения данной лабораторной работы было разработано оконное приложение на основе библиотеки WinAPI для языка C++. Оно состоит из двух форм: главной и дочерней. Главная форма предназначена для ввода пользователем матрицы смежности графа, а дочерняя форма отображает минимальное количество цветов, необходимых для раскраски графа.

Были реализованы функции, которые выполняют следующие задачи:

1. Проверка корректности ввода матрицы смежности и вершин: В процессе ввода пользователем матрицы и вершин осуществляется проверка на корректность введенных значений. Это включает проверку наличия всех необходимых полей и правильность формата данных.

2. Жадный алгоритм нахождения минимального количества цветов: для решения задачи раскраски графа был реализован жадный алгоритм. Он просматривает каждую вершину и выбирает первый доступный цвет, который не используется среди ее соседей. Таким образом, определяется минимальное количество цветов, необходимых для правильной раскраски графа.

3. Модальная дочерняя форма: после ввода матрицы и вершин на главной форме, создается дочерняя форма, которая выводит минимальное количество цветов для раскраски графа. Дочерняя форма работает в модальном режиме, что означает, что пользователь не может взаимодействовать с главной формой, пока не закроет дочернюю.

Все эти функции и компоненты окна были реализованы в ходе лабораторной работы для создания оконного приложения, которое позволяет пользователю вводить матрицу смежности графа, а затем выводит минимальное количество цветов, необходимых для его раскраски.