

**Содержание**

Реферат**5**

Введение**6**

1. Постановка задачи**7**
2. Теоретическая часть задания**8**
3. Описание алгоритма программы**10**
4. Описание программы**13**
5. Тестирование**24**
6. Ручной расчёт задачи**28**

Заключение**29**

Список литературы**30**

Приложение A. Листинг программы**31**

Приложение B. Результат работы программы **39**

**Реферат**

Отчет 39 стр, 8 рисунков.

ГРАФ, ТЕОРИЯ ГРАФОВ,

Цель исследования – разработка программы, выполняющей поиск максимальных паросочетаний в графе

В курсовой работе представлена программа способная выполнять поиск максимальных паросочетаний в графе, использующая для этого алгоритм Куна.

**Введение**

Для реализации алгоритма поиска максимальных паросочетаний можно использовать различные виды обхода графа (в глубину или в ширину).

Обход в графа в глубину заключается в обходе графа по некоторому пути в ходе, которого посещаются все вершины графа. Реализация может быть как рекурсивная, так и не рекурсивная. Во втором случае используется стек.

Обход графа в ширину заключается в проходе по всем возможным смежностям в графе, при этом в каждый момент исполнения алгоритма, рассматривается одна вершина.

Для поиска максимальных паросочетаний можно использовать алгоритм Куна, построенный на основе поиска в глубину.

Разработка программы будет вестись в среде Microsoft VisualStudio2019, на языке программирования - С++.

1. **Постановка задачи**

Требуется разработать программу выполняющею поиск максимальных паросочетаний в графе использую алгоритм Куна.

Исходный граф будет задаваться списком смежности. Пользователь будет выбирать, загрузить существующий граф из файла или сгенерировать случайный. После выполнения алгоритма будет выведен на экран граф и найденные паросочетания. Все результаты будут сохранены в файл.

В случае, какой либо ошибки будет выводиться соответствующее сообщение на экран

1. **Теоретическая часть задания**

Пусть дан неориентированный граф G = (V, Е). Паросочетанием (matching) называется подмножество ребер М є Е, такое что для всех вершин v е V в М содержится не более одного ребра, инцидентного v.

Максимальное паросочетание — это такое паросочетание M в графе G, которое не содержится ни в каком другом паросочетании этого графа, то есть к нему невозможно добавить ни одно ребро, которое бы являлось несмежным ко всем рёбрам паросочетания.

Задан граф G⟨V,E⟩, про который известно, что он двудольный, но разбиение не задано явно. Требуется найти наибольшее паросочетание в нём

Алгоритм можно описать так: сначала возьмём пустое паросочетание, а потом — пока в графе удаётся найти увеличивающую цепь, — будем выполнять чередование паросочетания вдоль этой цепи, и повторять процесс поиска увеличивающей цепи. Как только такую цепь найти не удалось — процесс останавливаем, — текущее паросочетание и есть максимальное.

В массиве matching хранятся паросочетания (v,matching[v]) (Если паросочетания с вершиной v не существует, то matching[v]=−1). А used — обычный массив "посещённостей" вершин в обходе в глубину (он нужен, чтобы обход в глубину не заходил в одну вершину дважды). Функция dfs возвращает true, если ей удалось найти увеличивающую цепь из вершины v, при этом считается, что эта функция уже произвела чередование паросочетания вдоль найденной цепи.

Внутри функции просматриваются все рёбра, исходящие из вершины v, и затем проверяется: если это ребро ведёт в ненасыщенную вершину to, либо если эта вершина to насыщена, но удаётся найти увеличивающую цепь рекурсивным запуском из matching[to], то мы говорим, что мы нашли увеличивающую цепь, и перед возвратом из функции с результатом true производим чередование в текущем ребре: перенаправляем ребро, смежное с to, в вершину v.

В основной программе сначала указывается, что текущее паросочетание — пустое (массив matching заполняется числами −1). Затем перебирается вершина v, и из неё запускается обход в глубину dfs, предварительно обнулив массив used.

Стоит заметить, что размер паросочетания легко получить как число вызовов dfs в основной программе, вернувших результат true. Само искомое максимальное паросочетание содержится в массиве matching. После того, как все вершины v∈V будут просмотрены, текущее паросочетание будет максимальным

1. **Описание алгоритма программы**

В программе граф записан в виде списка смежности, его можно прочитать из файла или сгенерировать случайный.

После помещения графа в список смежности представленный вектором, проверяем его на двудольность, если двудольный, то можем выполнять функцию поиска максимальных паросочетаний.

для начало массив паросочетаний обнуляем записывая в него “-1“.

затем запускаем цикл FOR от i=0 до размера графа и в нём в массив used записываем “-1” и вызываем функцию алгоритма Куна. В алгоритме куна запускаем цикл FOR от i=0 до размера графа и находим ненасыщенную вершину, помещаем её в переменную to или если вершина насыщена, но найдена увеличивающаяся цепь, то занчит, что паросочетание найдено и записываем его в pairs[to]. Когда найдены все паросочетания, выводим их на экран и сохраняем результаты в файл.

Псевдокод программы:

main(){

выбрать способ заполнения графа;

если выбран случайный граф;

{

создать граф();

}

иначе{

прочитать граф из файла

}

Если граф задан

{

Вывести граф на экран

Если (Граф двудольный))

Поиск максимальных паросочетаний();

Иначе

Вывести сообщение об ошибке

}

Иначе

Вывести сообщение об ошибке

}

создать граф(){

ввести размер 1 и 2 доли графа

для ( i = 0; до size)

{

для( j = 0; до size; )

{

graph[i][j] = 1 или 0;

graph[j][i] = graph[i][j];

}

}

size3= размер 1 доли + 2 доли

для(i = 0; до size3; )

{

для(j = 0; до size3; )

{

если (i < size1 и j < size1) {

graph[i][j] = 0;

}

}

}

для(int i =0; до size3; )

{

для( j = 0; до size3;)

{

Если (i >= size1 и j>= size1) {

graph[i][j] = 0;

}

}

}

пока (!граф не двудольный) {

создать граф();

}

для( i = 0; до size3;)

{

для( j = 0; до size3;)

{

Если (i == j) {

graph[i][j] = 0;

}

вывести на экран graph[i][j]

}

перейти на следующею строку

}

Алгоритм куна(){

Если (NUM[v] ==0)

Если(matching[i] == -1 или dfs(matching[i]==0))

возвратить *false*

used[v] = *true*

ДЛЯ всех i ИЗ g[i][j].

matching[i] = v

возвратить *true*

возвратить *false*

}

Поиск максимальных паросочетаний(){

ДЛЯ всех v matching[v] = -1;

NOMER = NOMER + 1;

ДЛЯ всех i ИЗ g[i][j]

used[i] = *false*

Алгоритм куна();

}

1. **Описание программы**

Приведённые ниже две функции выполняют генерацию случайного двудольного графа.

1) Генерируется рандомный граф размером (размер 1 доли + размер 2 доли).

2) убегаем связи между вершинами первой доли и повторяем для второй доли.

3) проверяем граф на двудольность, если не двудольный, то переходим к 1 пункту (граф может получиться не двудольным, так как значения задаются случайно).

int rand\_graph(int\*\* graph, int size)

{

for (int i = 0; i < size; i++)

{

for (int j = 0; j < size; j++)

{

graph[i][j] = rand()%2;

graph[j][i] = graph[i][j];

}

}

return 0;

}

int rand\_2dol(vector<size\_t> first, vector<size\_t> second, vector<vector<size\_t>> graph\_s)

{

int size1, size2, size3, \*\* graph;

printf("Ведите размер доли");

scanf\_s("%d", &size1);

printf("Ведите размер доли");

scanf\_s("%d", &size2);

size3 = size1 + size2;

graph = (int\*\*)malloc(size3 \* sizeof(int));

for (int i = 0; i < size3; i++)

{

graph[i] = (int\*)malloc(size3 \* sizeof(int));

}

rand\_graph(graph, size3);

for (int i = 0; i < size3; i++)

{

for (int j = 0; j < size3; j++)

{

if (i < size1 && j < size1) {

graph[i][j] = 0;

}

}

}

for (int i = 0; i < size3; i++)

{

for (int j = 0; j < size3; j++)

{

if (i >= size1 && j>= size1) {

graph[i][j] = 0;

}

}

}

while (!is\_bipartite(graph\_s, first, second)) {

rand\_2dol(first, second, graph\_s);

}

for (int i = 0; i < size3; i++)

{

for (int j = 0; j < size3; j++)

{

if (i == j) {

graph[i][j] = 0;

}

printf("%2d", graph[i][j]);

}

printf("\n");

}

FILE\* file;

file = fopen("SomeText.txt", "w");

for (int i = 0; i < size3; i++)

{

if (i != 0) {

fprintf(file, "\n");

}

fprintf(file, "%d ", i + 1);

for (int j = 0; j < size3; j++)

{

if (graph[i][j] == 1)

{

fprintf(file, "%d ", j+1);

}

}

}

fclose(file);

return 0;

}

В функции main реализовано меню, чтение из файла графа, и проверка на двудольность.

int main()

{

srand(time(NULL));

setlocale(LC\_CTYPE, "Russian");

int k;

std::cout << "Выбирите способ задачи графа (2 - сущ. из файла | 1 - сгенерировать случайный) ";

std::cin >> k;

std::string filename;

vector<size\_t> first, second;

vector<vector<size\_t>> graph;

if (k == 1)

{

rand\_2dol(first, second, graph);

filename = "SomeText.txt";

}

else {

std::cout << "Введите имя файла: ";

std::cin >> filename;

}

if (read\_graph(filename, graph))

{

std::cout << "Граф: " << std::endl;

write\_graph(graph);

//vector<size\_t> first, second;

if (is\_bipartite(graph, first, second))

algorithm\_kuhn(graph, first, second);

else

std::cout << "Граф не двудольный" << std::endl;

}

else

std::cout << "Файл не найден или возникла ошибка при чтении графа" << std::endl;

}

int main()

{

srand(time(NULL));

setlocale(LC\_CTYPE, "Russian");

int k;

std::cout << "Выбирите способ задачи графа (2 - сущ. из файла | 1 - сгенерировать случайный) ";

std::cin >> k;

std::string filename;

vector<size\_t> first, second;

vector<vector<size\_t>> graph;

if (k == 1)

{

rand\_2dol(first, second, graph);

filename = "SomeText.txt";

}

else {

std::cout << "Введите имя файла: ";

std::cin >> filename;

}

if (read\_graph(filename, graph))

{

std::cout << "Граф: " << std::endl;

write\_graph(graph);

//vector<size\_t> first, second;

if (is\_bipartite(graph, first, second))

algorithm\_kuhn(graph, first, second);

else

std::cout << "Граф не двудольный" << std::endl;

}

else

std::cout << "Файл не найден или возникла ошибка при чтении графа" << std::endl;

}

Функция read\_graph реалезует чтение графа из файла в список смежности.

bool read\_graph(std::string filename, std::vector<std::vector<size\_t>>& graph)

{

try

{

std::ifstream input(filename);

if (!input.good())

return 0;

while (!input.eof())

{

graph.push\_back(std::vector<size\_t>());

std::string row;

std::getline(input, row);

std::stringstream ss(row);

size\_t a;

ss >> a;

while (ss >> a)

{

graph[graph.size() - 1].push\_back(a - 1);

}

}

return 1;

}

catch (std::exception&)

{

return 0;

}

}

Функция write\_graph реалезует вывод графа на экран.

void write\_graph(const std::vector<std::vector<size\_t>>& graph)

{

for (size\_t i = 0; i < graph.size(); i++)

{

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

for (size\_t j = 0; j < graph[i].size(); j++)

std::cout << graph[i][j] + 1 << " ";

std::cout << std::endl;

}

}

// Вспомогательная функция для проверки графа на двудольность

bool is\_bipartite\_util(const std::vector<std::vector<size\_t>>& graph,

int src, std::vector<int>& param)

{

param[src] = 1;

std::queue<int> q;

q.push(src);

while (!q.empty())

{

int u = q.front();

q.pop();

if (std::find(graph[u].begin(), graph[u].end(), u) != graph[u].end())

return false;

for (size\_t v = 0; v < graph[u].size(); v++)

{

if (param[graph[u][v]] == -1)

{

param[graph[u][v]] = 1 - param[u];

q.push(graph[u][v]);

}

else if (param[graph[u][v]] == param[u])

return false;

}

}

return true;

}

Функция is\_bipartite реалезует проверку графа на двудольность

bool is\_bipartite(const std::vector<std::vector<size\_t>>& graph,

std::vector<size\_t>& first, std::vector<size\_t>& second)

{

std::vector<int> param(graph.size(), -1);

for (size\_t i = 0; i < graph.size(); i++)

if (param[i] == -1)

if (!is\_bipartite\_util(graph, i, param))

return 0;

for (size\_t i = 0; i < param.size(); i++)

if (param[i])

first.push\_back(i);

else

second.push\_back(i);

return 1;

}

Функция try\_algorithm\_kuhn представляет алгоритм куна

bool try\_algorithm\_kuhn(size\_t from, const vector<vector<size\_t>>& graph, vector<int>& pairs,

vector<bool>& visited, const vector<size\_t>& first, const vector<size\_t>& second)

{

int vfrom = std::distance(first.begin(), std::find(first.begin(), first.end(), from));

if (visited[vfrom])

return false;

visited[vfrom] = true;

for (size\_t i = 0; i < graph[from].size(); i++)

{

int to = std::distance(second.begin(), std::find(second.begin(), second.end(), graph[from][i]));

if (pairs[to] == -1 || try\_algorithm\_kuhn(pairs[to], graph, pairs, visited, first, second))

{

pairs[to] = from;

return true;

}

}

return false;

}

Функция algorithm\_kuhn выполняет запуска алгоритма Куна для каждой вершины, вывод и сохранение в файл максимальных паросочетаний.

void algorithm\_kuhn(const vector<vector<size\_t>>& graph,

const vector<size\_t>& first, const vector<size\_t>& second)

{

size\_t n = first.size();

size\_t k = second.size();

vector<int> pairs;

pairs.assign(k, -1);

for (size\_t v = 0; v < n; v++)

{

vector<bool> visited;

visited.assign(n, false);

try\_algorithm\_kuhn(first[v], graph, pairs, visited, first, second);

}

ofstream fout("rez.txt");

std::cout << "Максимальное паросочетание:" << std::endl;

fout << "Максимальное паросочетание:" << std::endl;

for (size\_t i = 0; i < k; ++i) {

if (pairs[i] != -1) {

std::cout << pairs[i] + 1 << " - " << second[i] + 1 << std::endl;

fout << pairs[i] + 1 << "-" <<second[i] + 1 << std::endl;

}

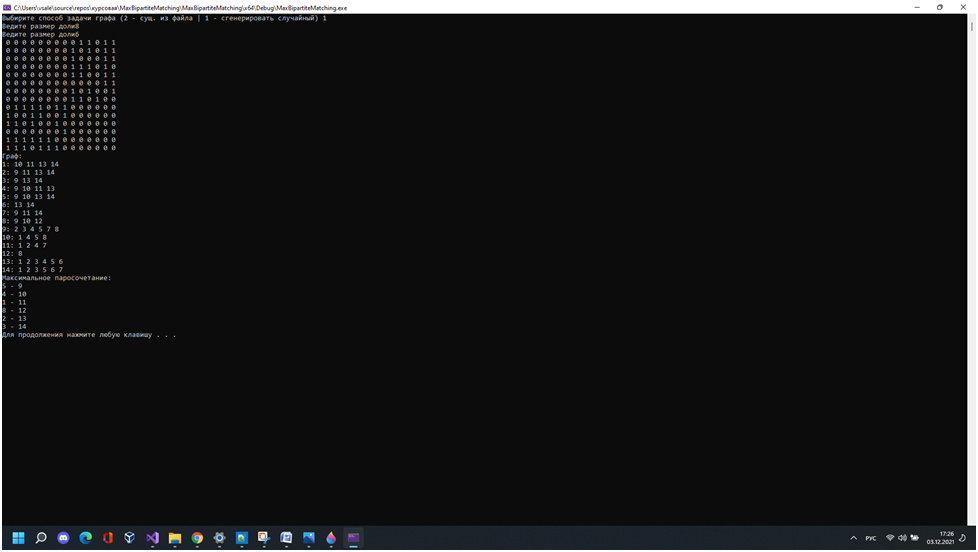
}

fout.close();

system("pause");

}

Ниже приведен результат работы программы:



Рисунок

1. **Тестирование**

Среда разработки MicrosoftVisualStudio 2019 предоставляет все

средства, необходимые при разработке и отладке многомодульной программы.

Тестирование проводилось в рабочем порядке, в процессе разработки,

после завершения написания программы. В ходе тестирования было выявлено и

исправлено множество проблем, связанных с вводом данных, алгоритмом программы.

функций.

Ниже продемонстрирован результат тестирования программы при тестировании на разных наборах данных.

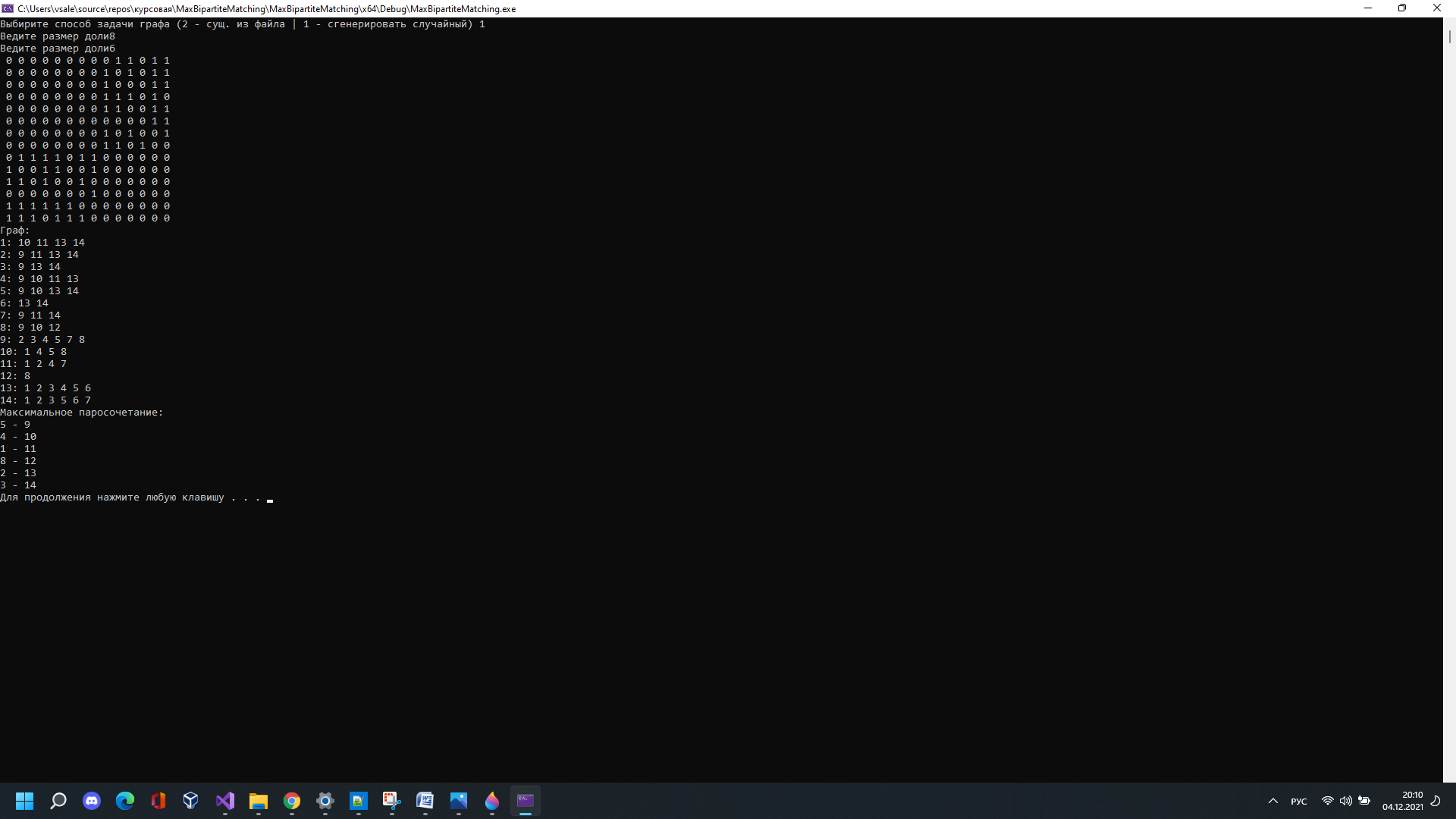


Рисунок 1 - Результат работы на случайном графе из 14 вершин.

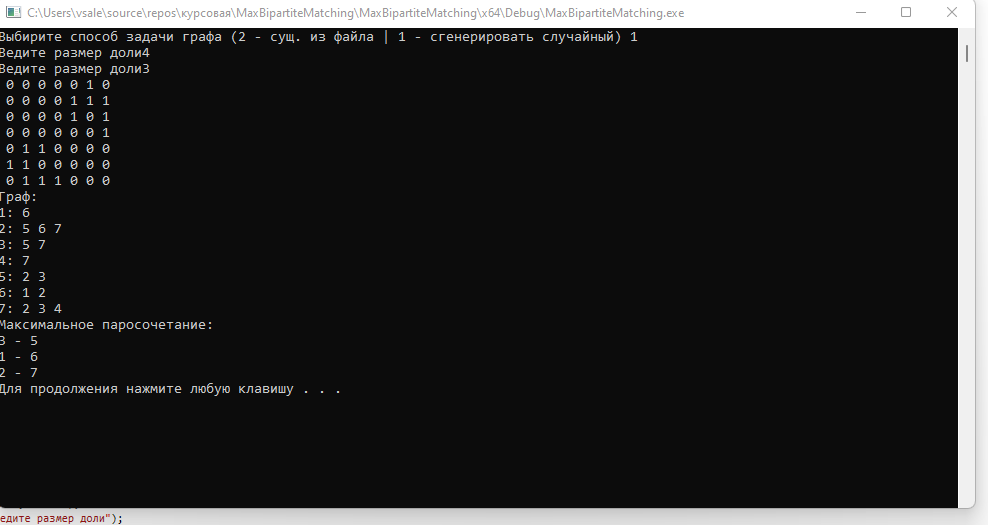


Рисунок 2 - Результат работы на случайном графе из 7 вершин.

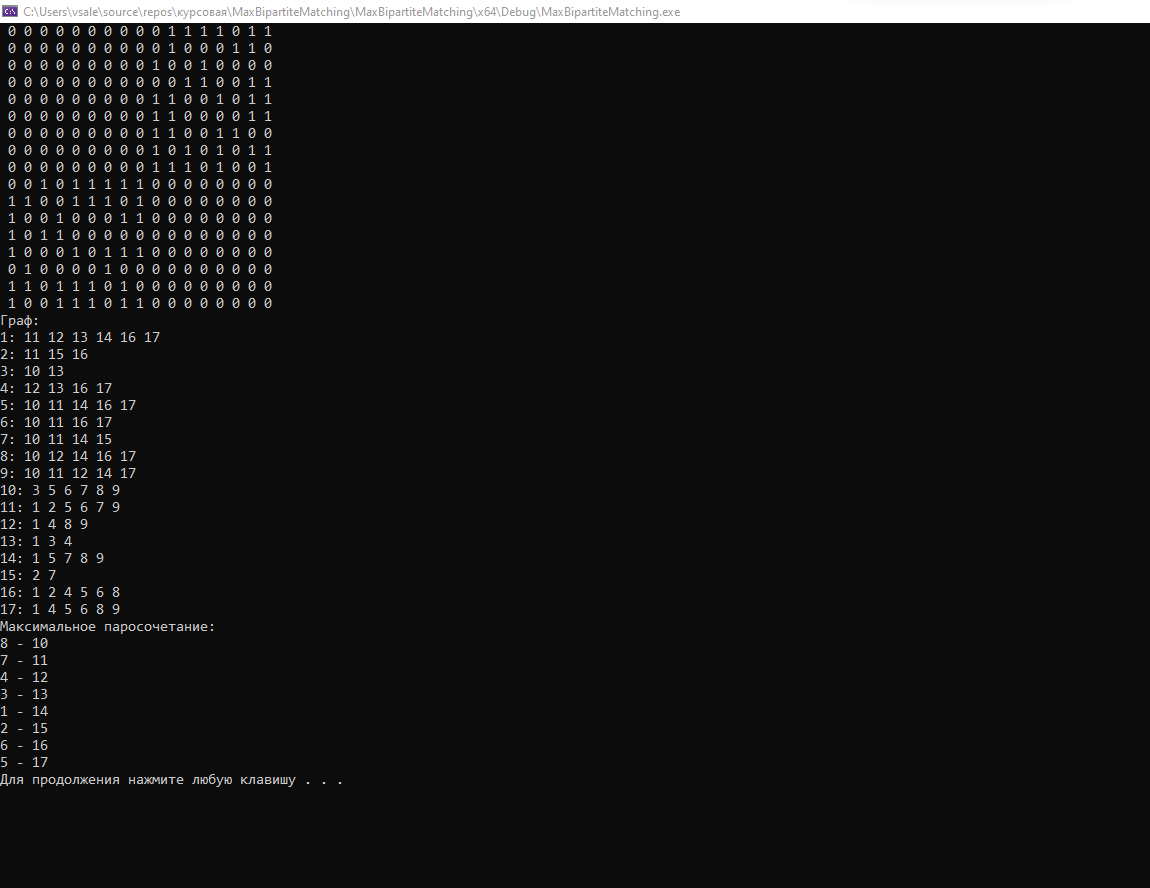


Рисунок 3 - Результат работы на случайном графе из 17 вершин.

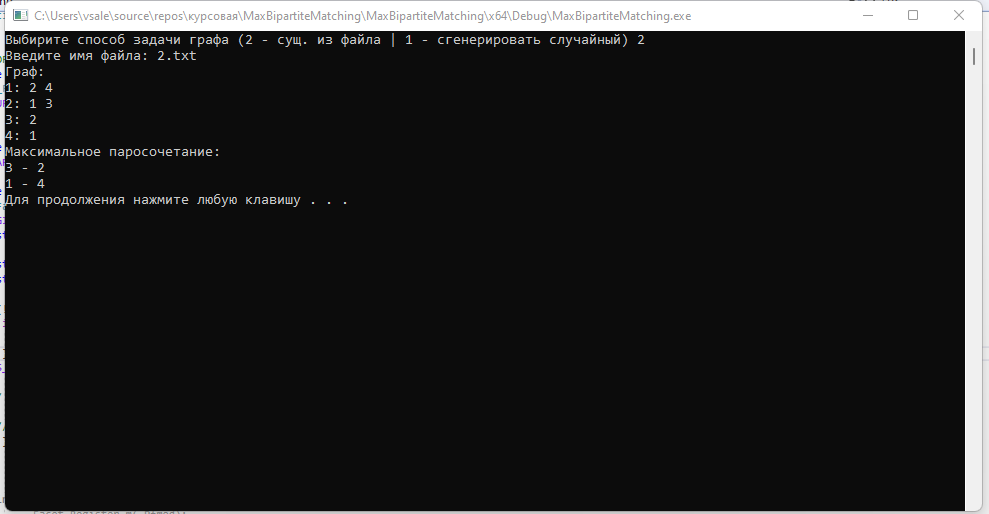


Рисунок 4 - Результат работы на существующем графе из 4 вершин загруженном из файла.

Таблица 1 Описание поведения программы при тестировании.

|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Запуск программы | Программа запустилась | Верно |
| Генерация графа | Сгенерировался граф размером “N” | Верно |
| Загрузка графа | Прочитался и загрузился в программу граф. | Верно |
| Выполнение алгоритма Куна | Алгоритм Куна выполнился без ошибок | Верно |
| Вывод и сохранение результатов | Результаты вывелись на экран и сохранились в файл | Верно |

1. **Ручной расчѐт задачи**

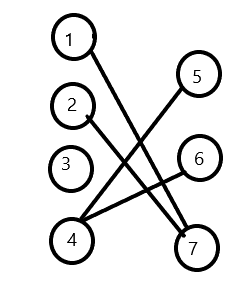
Создали двудольный граф на 7 вершин,

Рисунок 6

1:

2: 7

3: 6

4: 6 7

5:

6: 3 4

7: 2 4

Граф двудольный, поэтому начинаем поиск максимальных паросочетаний, берём вершину 1 и из неё исходит увеличивающаяся цепь, значит эта вершина является вершиной паросочетания. Те же действие проводим со 2 вершиной, но она смежная с уже найденным паросочетанием, поэтому пропускаем, следующая вершина из которой исходит увеличивающаяся цепь, это вершина 4, значит эта вершина является вершиной паросочетания.

Получились паросочетания 1 - 7 и 4 – 5

**Заключение**

В результате выполнения данной работы, была написана программа выполняющая поиск максимальных паросочетаний через алгоритм Куна.

Были получены навыки в работе с графами, изучены различные способы обхода графа, генерации, представления в памяти и операции с ними. Также были улучшены знания языка С++, такие как, работа с классами, потоками и изучены различные стандартные библиотеки данного языка программирования.

Полученные знания безусловно пригодятся в будущем для написания более сложных программ.

**Список литературы**

1. [**https://ravesli.com/urok-95-vvedenie-v-std-vector-vektory/**](https://ravesli.com/urok-95-vvedenie-v-std-vector-vektory/)
2. **Теория графов Д.В.Карпов.**
3. **https://prog-cpp.ru/data-graph/**
4. **https://neerc.ifmo.ru/wiki/index.php?title=%D0%A2%D0%B5%D0%BE%D1%80%D0%B8%D1%8F\_%D0%B3%D1%80%D0%B0%D1%84%D0%BE%D0%B2**
5. **https://e-maxx.ru/algo/kuhn\_matching**
6. **https://users.math-cs.spbu.ru/~okhotin/teaching/tcs1\_2016/okhotin\_tcs1\_2016\_l6.pdf**

**Приложение А.**

Листинг программы.

#pragma warning(disable : 4996)

#include "kuhn.h"

#include "utils.h"

#include <iostream>

#include <string>

#include <vector>

#include <ctime>"

#include <fstream>

#include <iostream>

#include <cstdlib>

#include <queue>

#include <sstream>

using namespace std;

int rand\_graph(int\*\* graph, int size)

{

for (int i = 0; i < size; i++)

{

for (int j = 0; j < size; j++)

{

graph[i][j] = rand()%2;

graph[j][i] = graph[i][j];

}

}

return 0;

}

int rand\_2dol(vector<size\_t> first, vector<size\_t> second, vector<vector<size\_t>> graph\_s)

{

int size1, size2, size3, \*\* graph;

printf("Ведите размер доли");

scanf\_s("%d", &size1);

printf("Ведите размер доли");

scanf\_s("%d", &size2);

size3 = size1 + size2;

graph = (int\*\*)malloc(size3 \* sizeof(int));

for (int i = 0; i < size3; i++)

{

graph[i] = (int\*)malloc(size3 \* sizeof(int));

}

rand\_graph(graph, size3);

for (int i = 0; i < size3; i++)

{

for (int j = 0; j < size3; j++)

{

if (i < size1 && j < size1) {

graph[i][j] = 0;

}

}

}

for (int i = 0; i < size3; i++)

{

for (int j = 0; j < size3; j++)

{

if (i >= size1 && j>= size1) {

graph[i][j] = 0;

}

}

}

while (!is\_bipartite(graph\_s, first, second)) {

rand\_2dol(first, second, graph\_s);

}

for (int i = 0; i < size3; i++)

{

for (int j = 0; j < size3; j++)

{

if (i == j) {

graph[i][j] = 0;

}

printf("%2d", graph[i][j]);

}

printf("\n");

}

FILE\* file;

file = fopen("SomeText.txt", "w");

for (int i = 0; i < size3; i++)

{

if (i != 0) {

fprintf(file, "\n");

}

fprintf(file, "%d ", i + 1);

for (int j = 0; j < size3; j++)

{

if (graph[i][j] == 1)

{

fprintf(file, "%d ", j+1);

}

}

}

fclose(file);

return 0;

}

int main()

{

srand(time(NULL));

setlocale(LC\_CTYPE, "Russian");

int k;

std::cout << "Выбирите способ задачи графа (2 - сущ. из файла | 1 - сгенерировать случайный) ";

std::cin >> k;

std::string filename;

vector<size\_t> first, second;

vector<vector<size\_t>> graph;

if (k == 1)

{

rand\_2dol(first, second, graph);

filename = "SomeText.txt";

}

else {

std::cout << "Введите имя файла: ";

std::cin >> filename;

}

if (read\_graph(filename, graph))

{

std::cout << "Граф: " << std::endl;

write\_graph(graph);

//vector<size\_t> first, second;

if (is\_bipartite(graph, first, second))

algorithm\_kuhn(graph, first, second);

else

std::cout << "Граф не двудольный" << std::endl;

}

else

std::cout << "Файл не найден или возникла ошибка при чтении графа" << std::endl;

}

#include "utils.h"

//#include <algorithm>

#include <iostream>

#include <queue>

#include <sstream>

// Чтение графа из файла в список смежности

bool read\_graph(std::string filename, std::vector<std::vector<size\_t>>& graph)

{

try

{

std::ifstream input(filename);

if (!input.good())

return 0;

while (!input.eof())

{

graph.push\_back(std::vector<size\_t>());

std::string row;

std::getline(input, row);

std::stringstream ss(row);

size\_t a;

ss >> a;

while (ss >> a)

{

graph[graph.size() - 1].push\_back(a - 1);

}

}

return 1;

}

catch (std::exception&)

{

return 0;

}

}

// Вывод графа на экран

void write\_graph(const std::vector<std::vector<size\_t>>& graph)

{

for (size\_t i = 0; i < graph.size(); i++)

{

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

for (size\_t j = 0; j < graph[i].size(); j++)

std::cout << graph[i][j] + 1 << " ";

std::cout << std::endl;

}

}

// Вспомогательная функция для проверки графа на двудольность

bool is\_bipartite\_util(const std::vector<std::vector<size\_t>>& graph,

int src, std::vector<int>& param)

{

param[src] = 1;

std::queue<int> q;

q.push(src);

while (!q.empty())

{

int u = q.front();

q.pop();

if (std::find(graph[u].begin(), graph[u].end(), u) != graph[u].end())

return false;

for (size\_t v = 0; v < graph[u].size(); v++)

{

if (param[graph[u][v]] == -1)

{

param[graph[u][v]] = 1 - param[u];

q.push(graph[u][v]);

}

else if (param[graph[u][v]] == param[u])

return false;

}

}

return true;

}

// Проверка графа на двудольность

bool is\_bipartite(const std::vector<std::vector<size\_t>>& graph,

std::vector<size\_t>& first, std::vector<size\_t>& second)

{

std::vector<int> param(graph.size(), -1);

for (size\_t i = 0; i < graph.size(); i++)

if (param[i] == -1)

if (!is\_bipartite\_util(graph, i, param))

return 0;

for (size\_t i = 0; i < param.size(); i++)

if (param[i])

first.push\_back(i);

else

second.push\_back(i);

return 1;

}

#include "kuhn.h"

#include <iostream>

#include <string>

#include <vector>

#include <ctime>"

#include <fstream>

#include <iostream>

#include <cstdlib>

#include <queue>

#include <sstream>

using namespace std;

// Попытка выполнить алгоритм Куна

bool try\_algorithm\_kuhn(size\_t from, const vector<vector<size\_t>>& graph, vector<int>& pairs,

vector<bool>& visited, const vector<size\_t>& first, const vector<size\_t>& second)

{

int vfrom = std::distance(first.begin(), std::find(first.begin(), first.end(), from));

if (visited[vfrom])

return false;

visited[vfrom] = true;

for (size\_t i = 0; i < graph[from].size(); i++)

{

int to = std::distance(second.begin(), std::find(second.begin(), second.end(), graph[from][i]));

if (pairs[to] == -1 || try\_algorithm\_kuhn(pairs[to], graph, pairs, visited, first, second))

{

pairs[to] = from;

return true;

}

}

return false;

}

// Запуск алгоритма Куна

void algorithm\_kuhn(const vector<vector<size\_t>>& graph,

const vector<size\_t>& first, const vector<size\_t>& second)

{

size\_t n = first.size();

size\_t k = second.size();

vector<int> pairs;

pairs.assign(k, -1);

for (size\_t v = 0; v < n; v++)

{

vector<bool> visited;

visited.assign(n, false);

try\_algorithm\_kuhn(first[v], graph, pairs, visited, first, second);

}

ofstream fout("rez.txt");

std::cout << "Максимальное паросочетание:" << std::endl;

fout << "Максимальное паросочетание:" << std::endl;

for (size\_t i = 0; i < k; ++i) {

if (pairs[i] != -1) {

std::cout << pairs[i] + 1 << " - " << second[i] + 1 << std::endl;

fout << pairs[i] + 1 << "-" <<second[i] + 1 << std::endl;

}

}

fout.close();

system("pause");

}

**Приложение В.**

Результат работы программы.

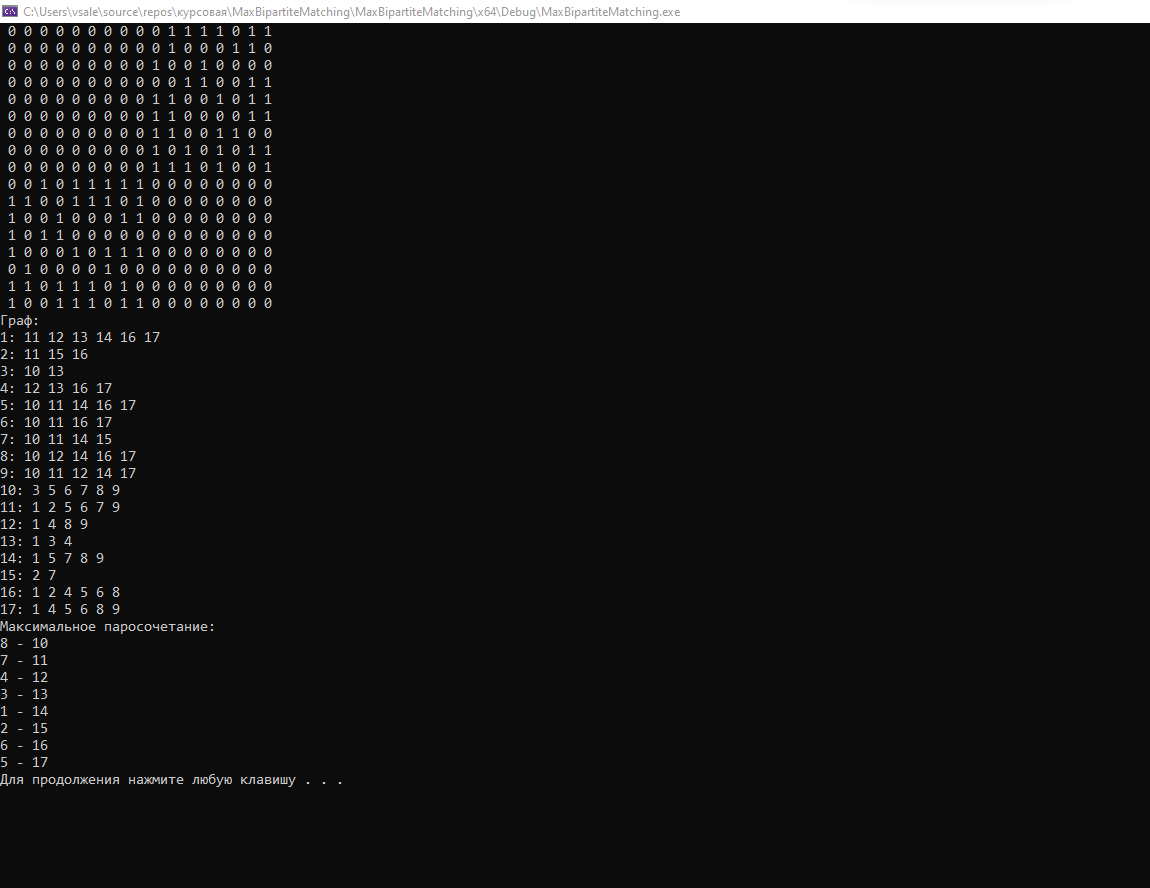


Рисунок 8