Московский Авиационный Институт (национальный исследовательский университет)  
Факультет прикладной математики

и физики.

**Курсовая работа по дискретной математике.**

Выполнила:

Студентка группы 8О-104Б

Черыгова Е.Е.

Проверила:

Смерчинская С.О.

Москва 2015

Оглавление

[Задание 3](#_Toc422091646)

[Теория 3](#_Toc422091647)

[Алгоритм решения задачи 4](#_Toc422091648)

[Вычислительная сложность 4](#_Toc422091649)

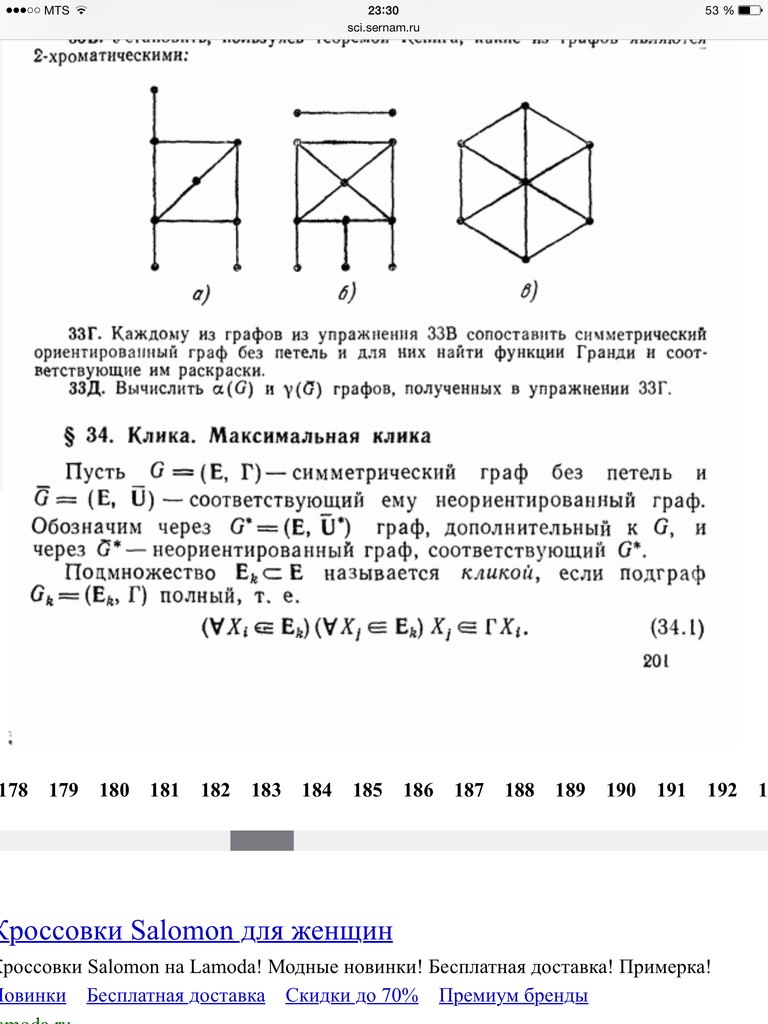
[Описание программы 5](#_Toc422091650)

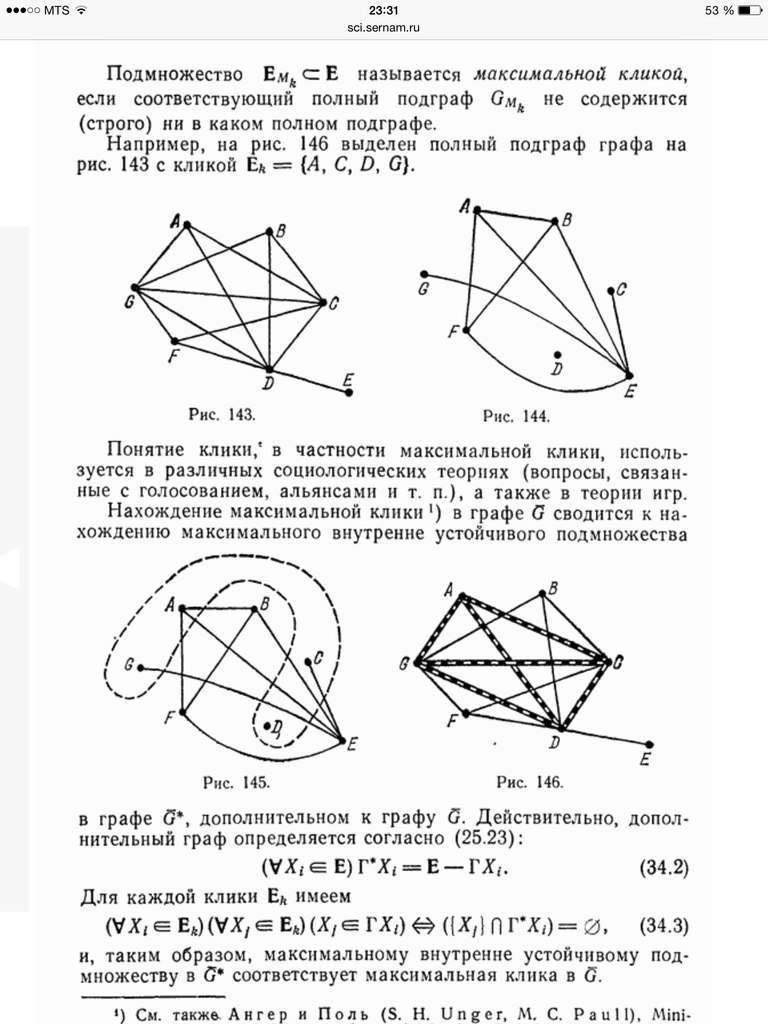
[Код программы 10](#_Toc422091651)

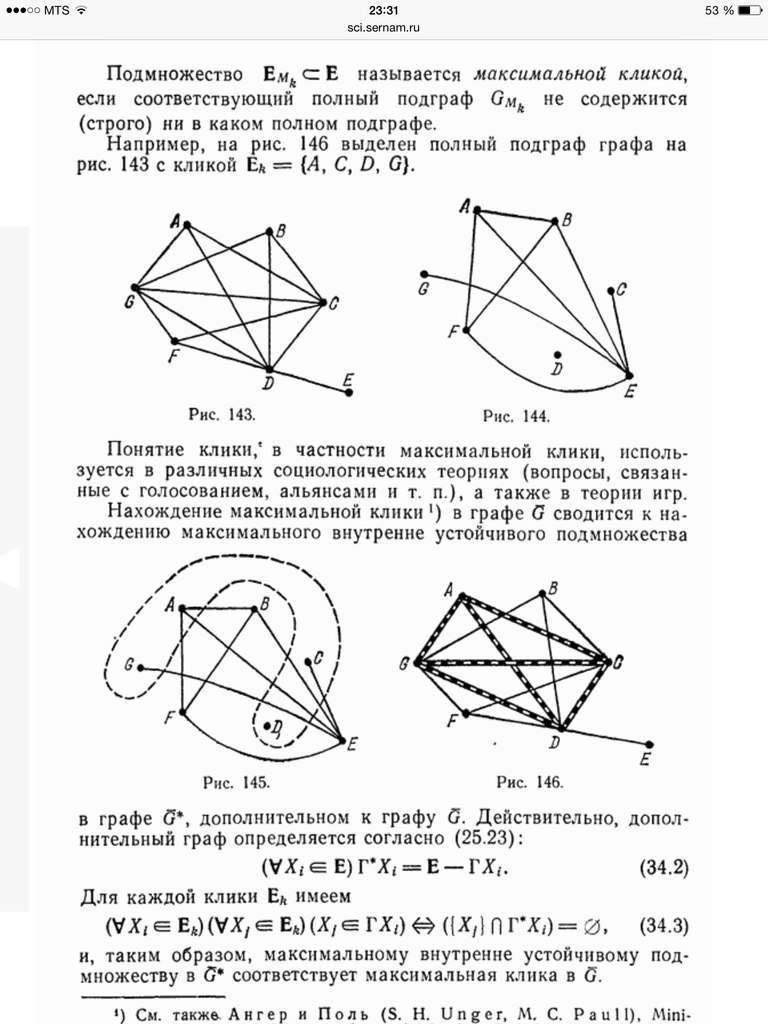
### Задание

Составить программу для поиска максимальной клики в графе

### Теория







### Алгоритм решения задачи

За основу взят алгоритм Брона-Кербоша.

Алгоритм использует тот факт, что всякая клика в графе является его максимальным по включению полным подграфом. Начиная с одиночной вершины (образующей полный подграф), алгоритм на каждом шаге пытается увеличить уже построенный полный подграф, добавляя в него вершины из множества кандидатов. Высокая скорость обеспечивается отсечением при переборе вариантов, которые заведомо не приведут к построению клики, для чего используется дополнительное множество, в которое помещаются вершины, которые уже были использованы для увеличения полного подграфа.

Алгоритм оперирует тремя множествами вершин графа:

1. Множество **compsub** — множество, содержащее на каждом шаге рекурсии полный подграф для данного шага. Строится рекурсивно.
2. Множество **candidates** — множество вершин, которые могут увеличить compsub
3. Множество **used** — множество вершин, которые уже использовались для расширения **compsub** на предыдущих шагах алгоритма.

Алгоритм является рекурсивной процедурой, применяемой к этим трем множествам.

**ПРОЦЕДУРА** *extend* (*candidates*, *not*):

**ПОКА** *candidates* НЕ пусто **И** *used* НЕ содержит вершины, СОЕДИНЕННОЙ СО ВСЕМИ вершинами из *candidates*,

**ВЫПОЛНЯТЬ**:

1 Выбираем вершину *v* из *candidates* и добавляем ее в *compsub*

2 Формируем *new\_candidates* и *new\_not*, удаляя из *candidates* и *not* вершины, не СОЕДИНЕННЫЕ с *v*

3 **ЕСЛИ** *new\_candidates* и *new\_not* пусты

4 **ТО** *compsub* – клика

5 **ИНАЧЕ** рекурсивно вызываем *extend* (*new\_candidates*, *new\_not*)

6 Удаляем v из *compsub* и *candidates*, и помещаем в *used*

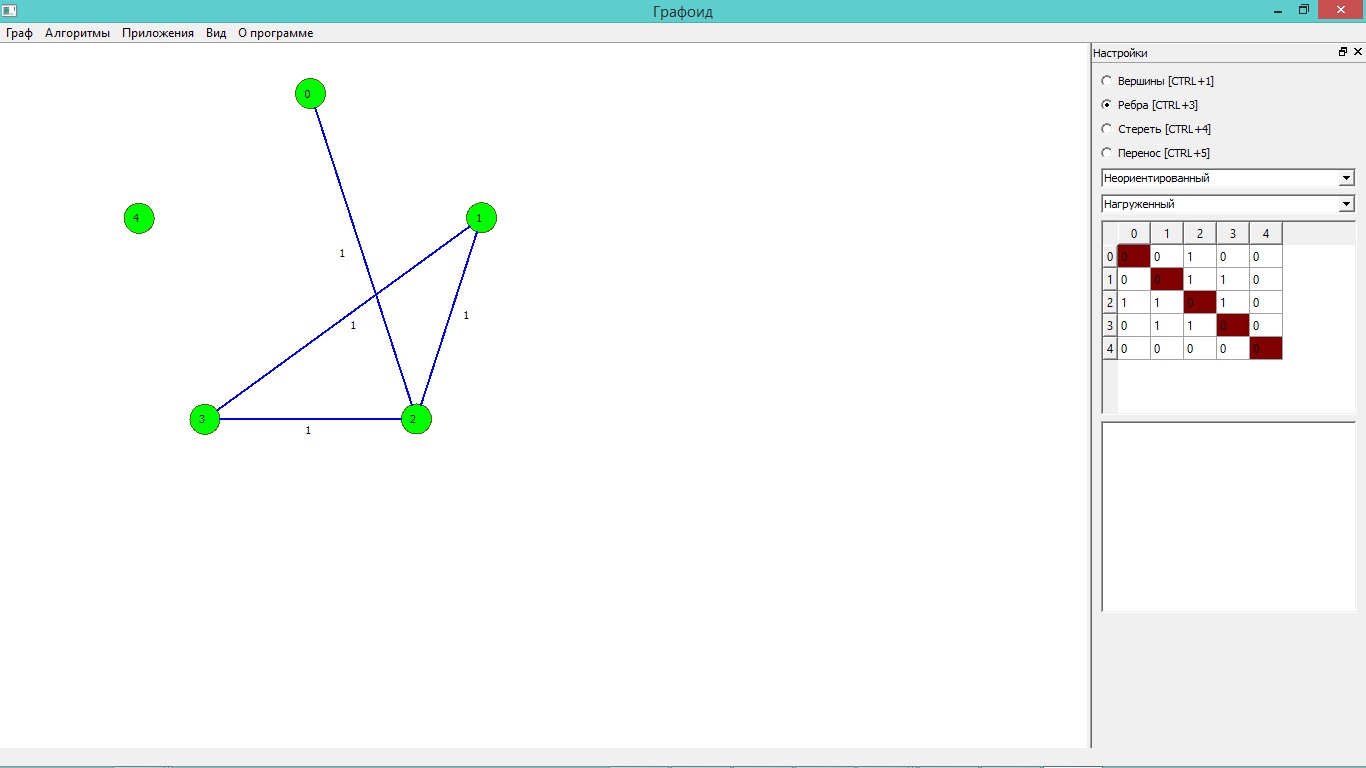
### Вычислительная сложность

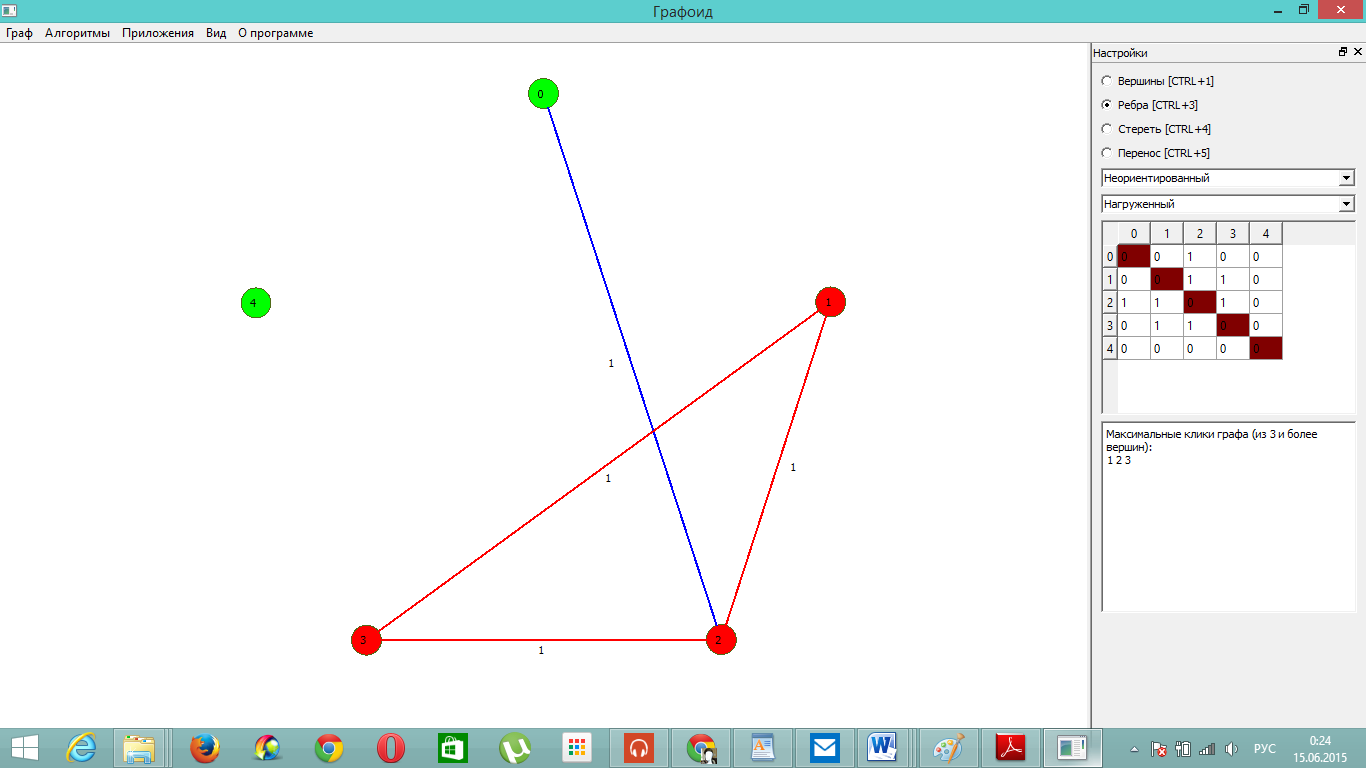
Линейна относительно количества клик в графе. В худшем случае алгоритм работает за O(3n/3), где n — количество вершин в графе.

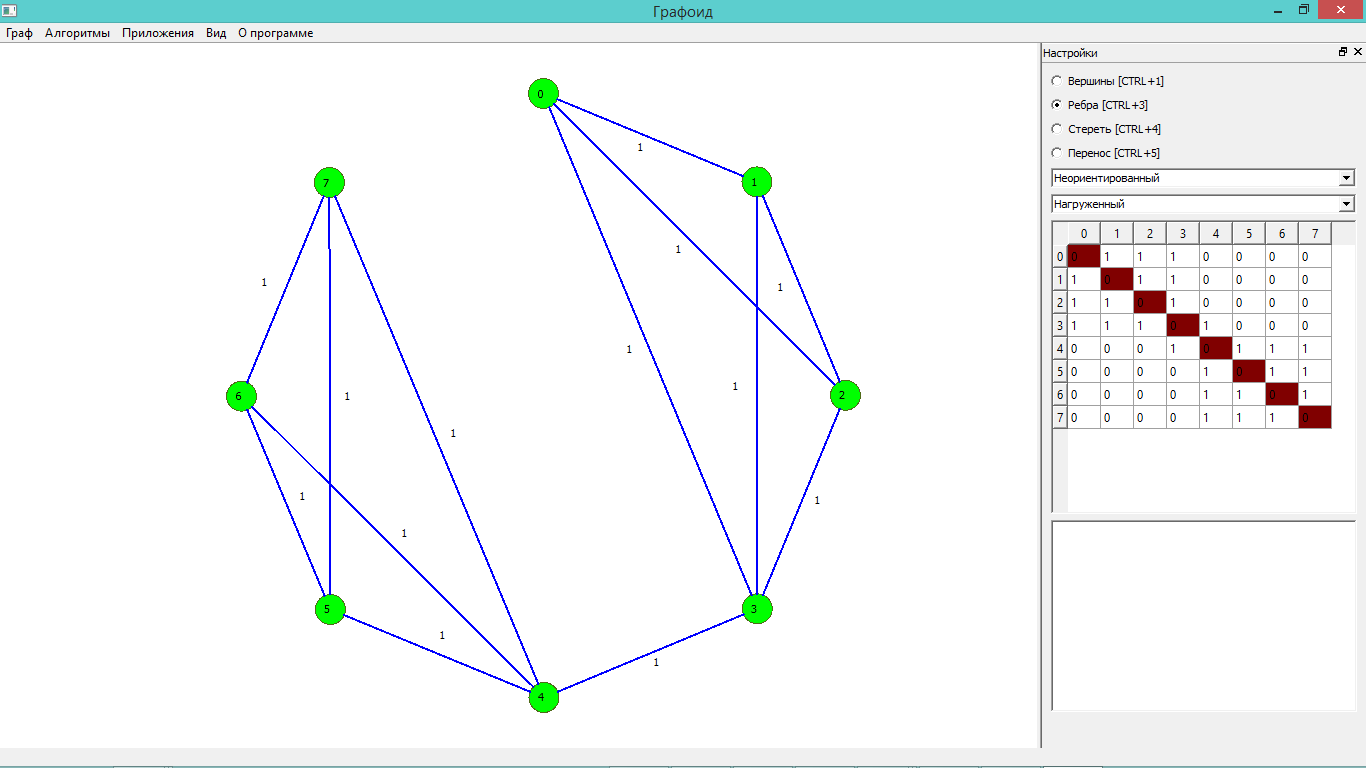
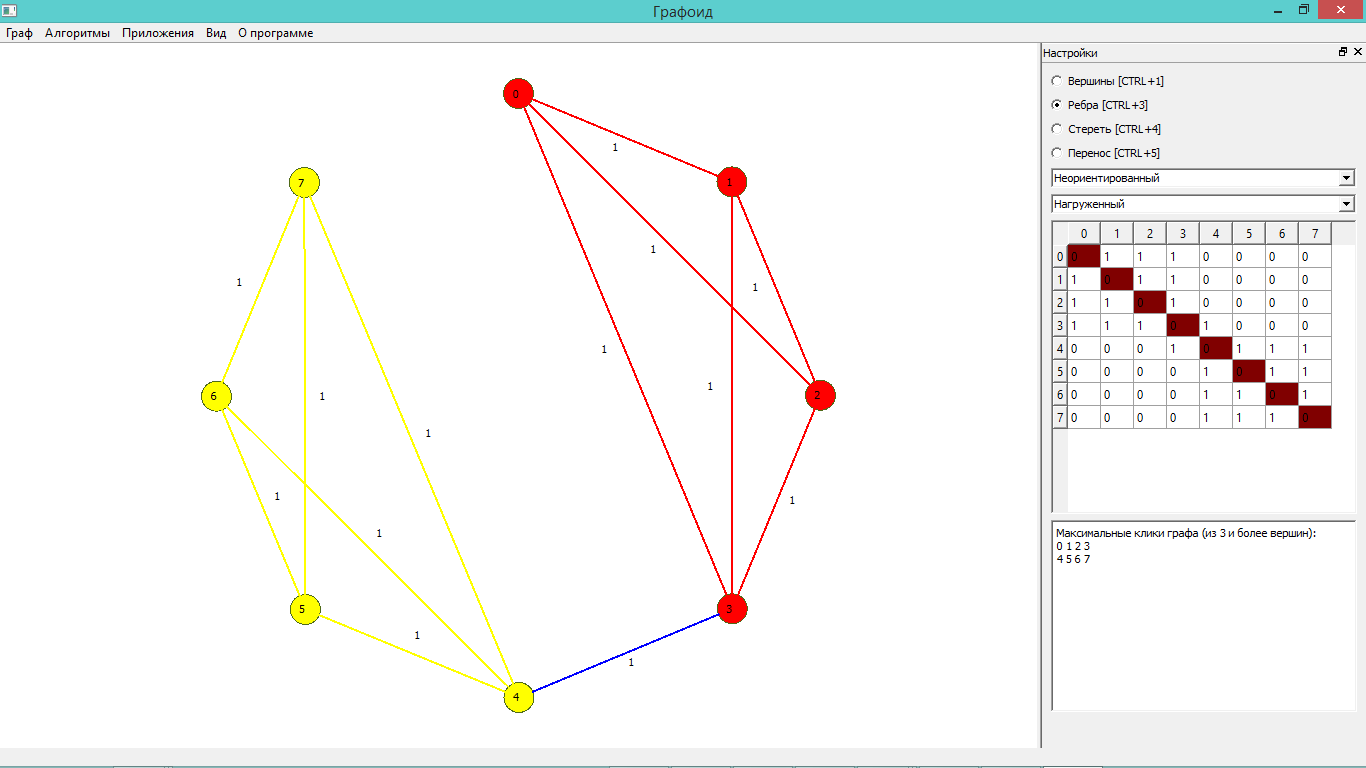
### Описание программы

Разработанная программа находит все максимальные клики графа используя алгоритм Брона-Кербоша. Изначально в программе рисуется граф и автоматически строится его матрица смежности. Программа работает непосредственно с этой матрицей. В конечном счете, на экран выводится список всех максимальных клик графа с номерами вершин и каждая максимальная клика исходного графа выделяется с помощью определённого цвета. Если у нескольких максимальных клик есть общие вершины и ребра, то приоритет в раскраске отдаётся наибольшей из найденных.

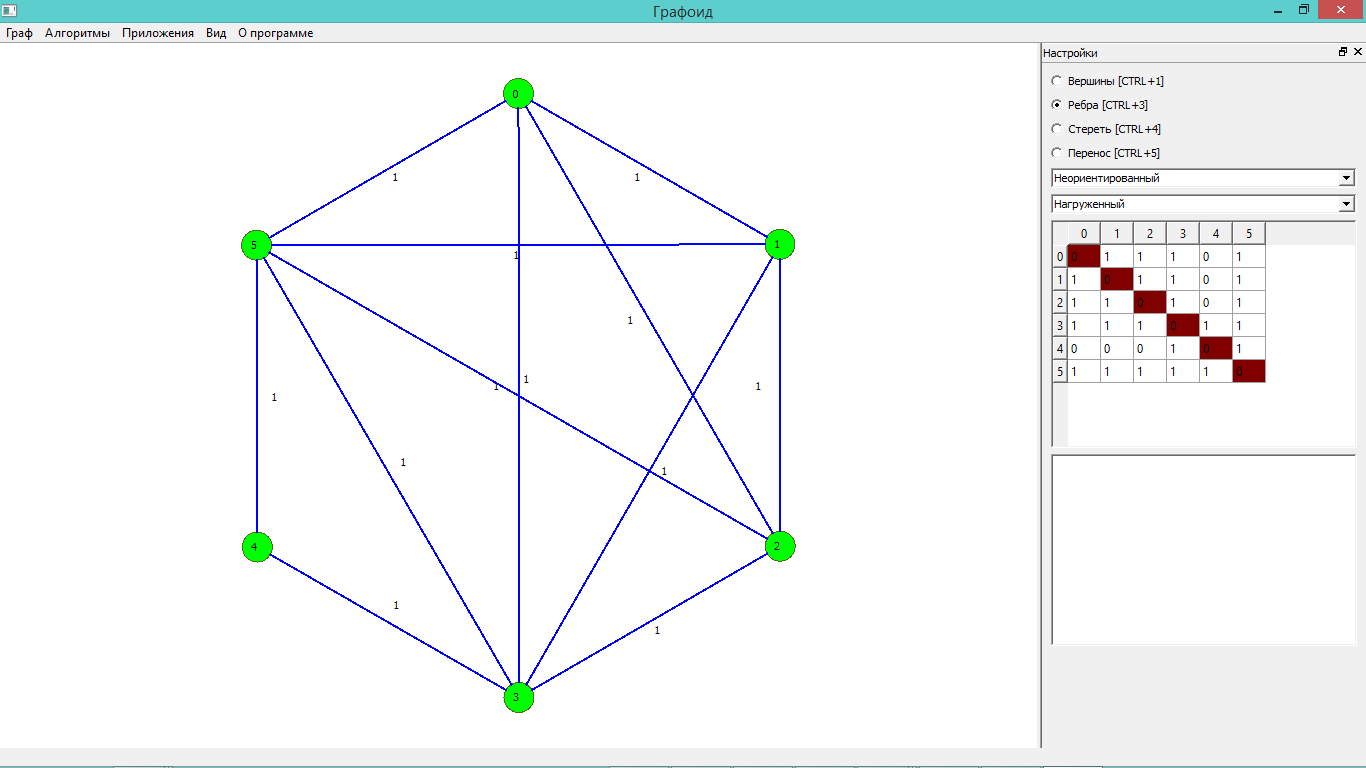
Пример 1.

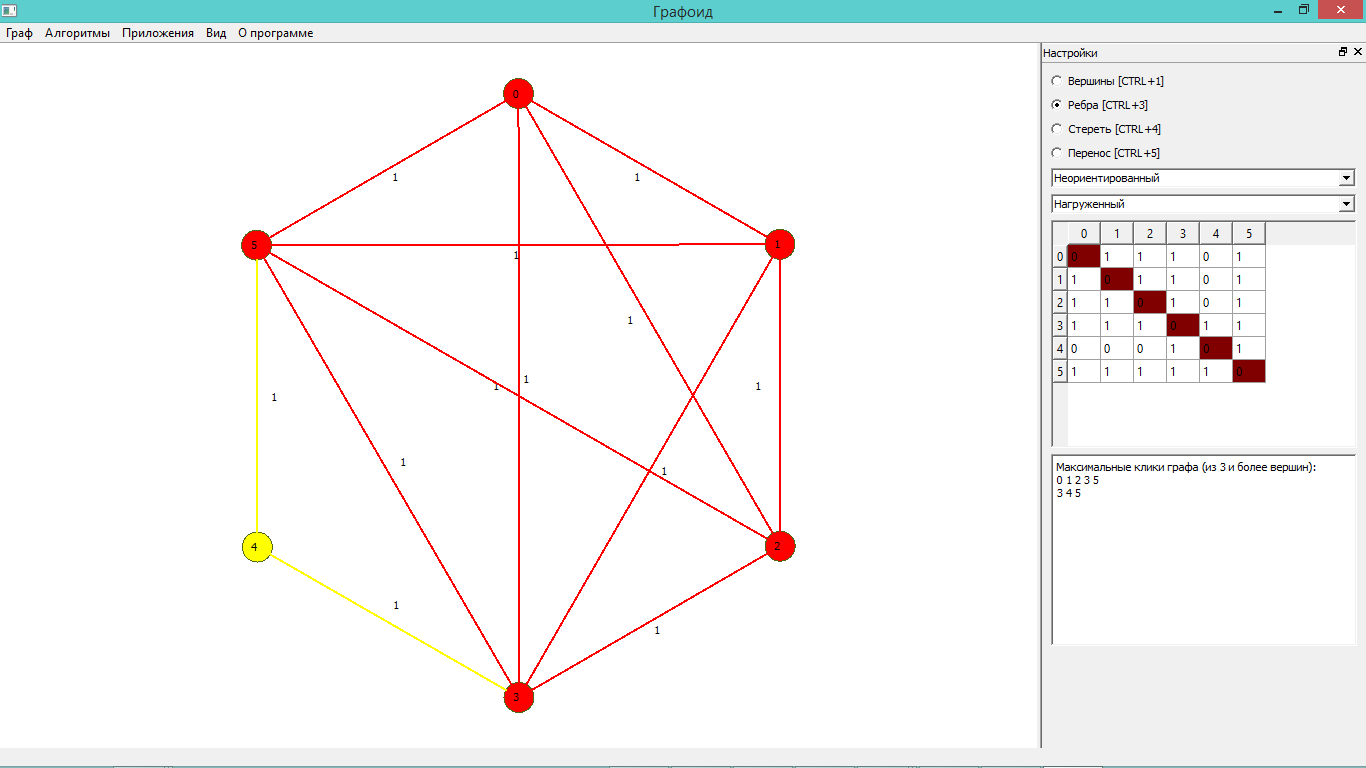




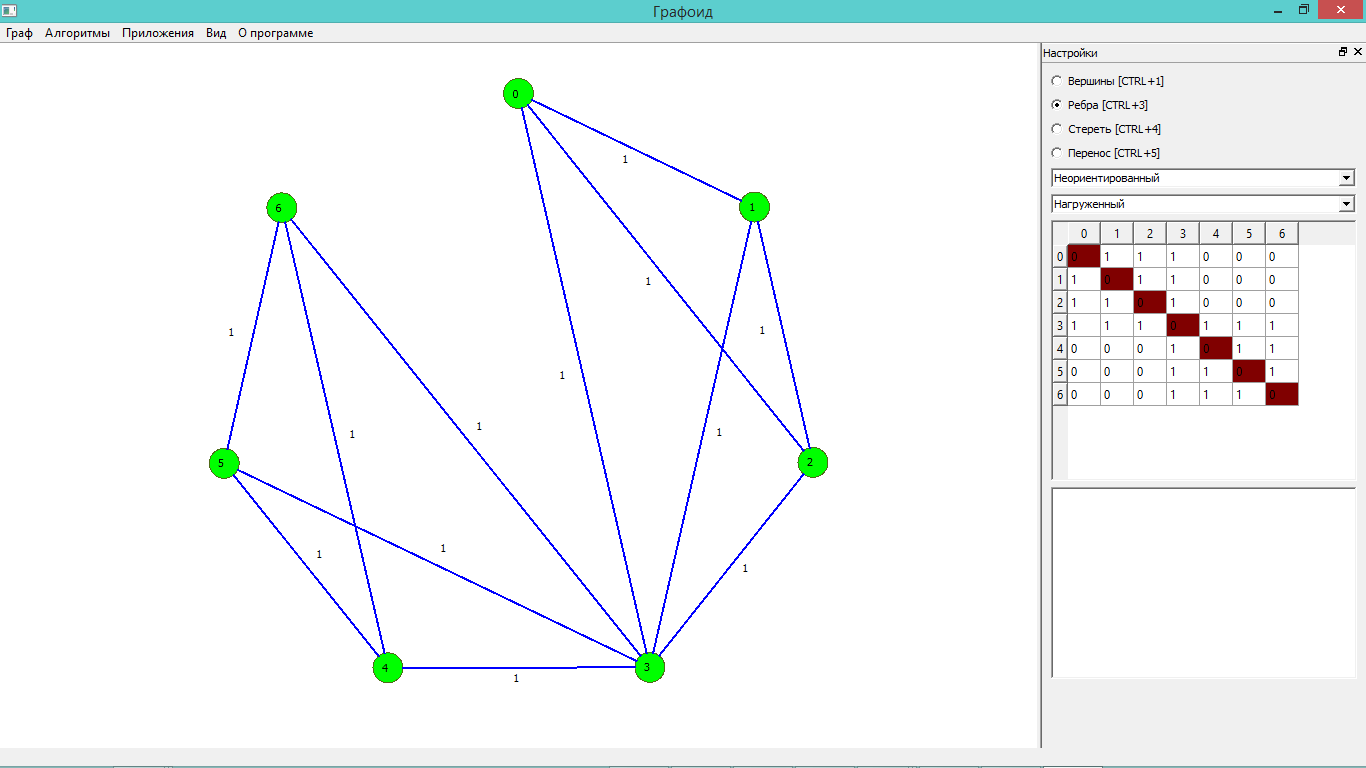
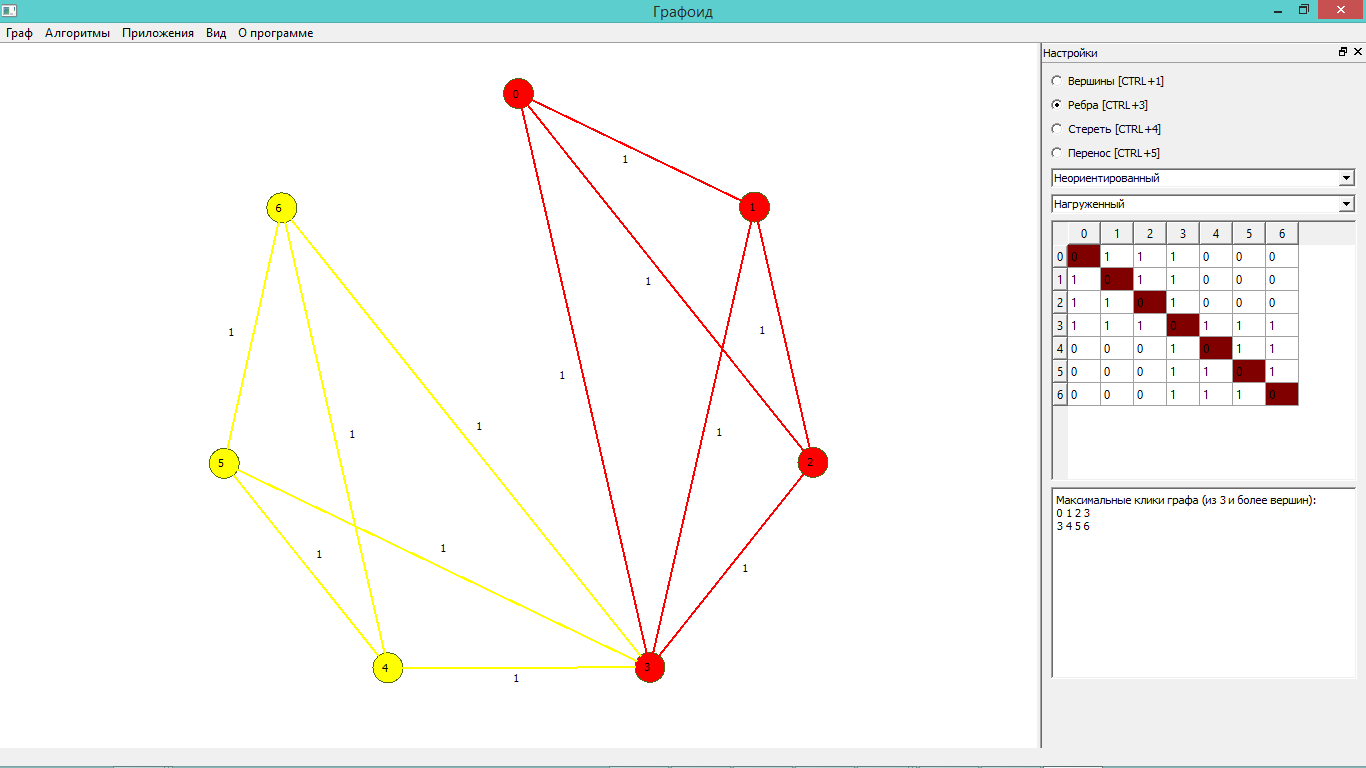
Пример 2. 

Пример 3.

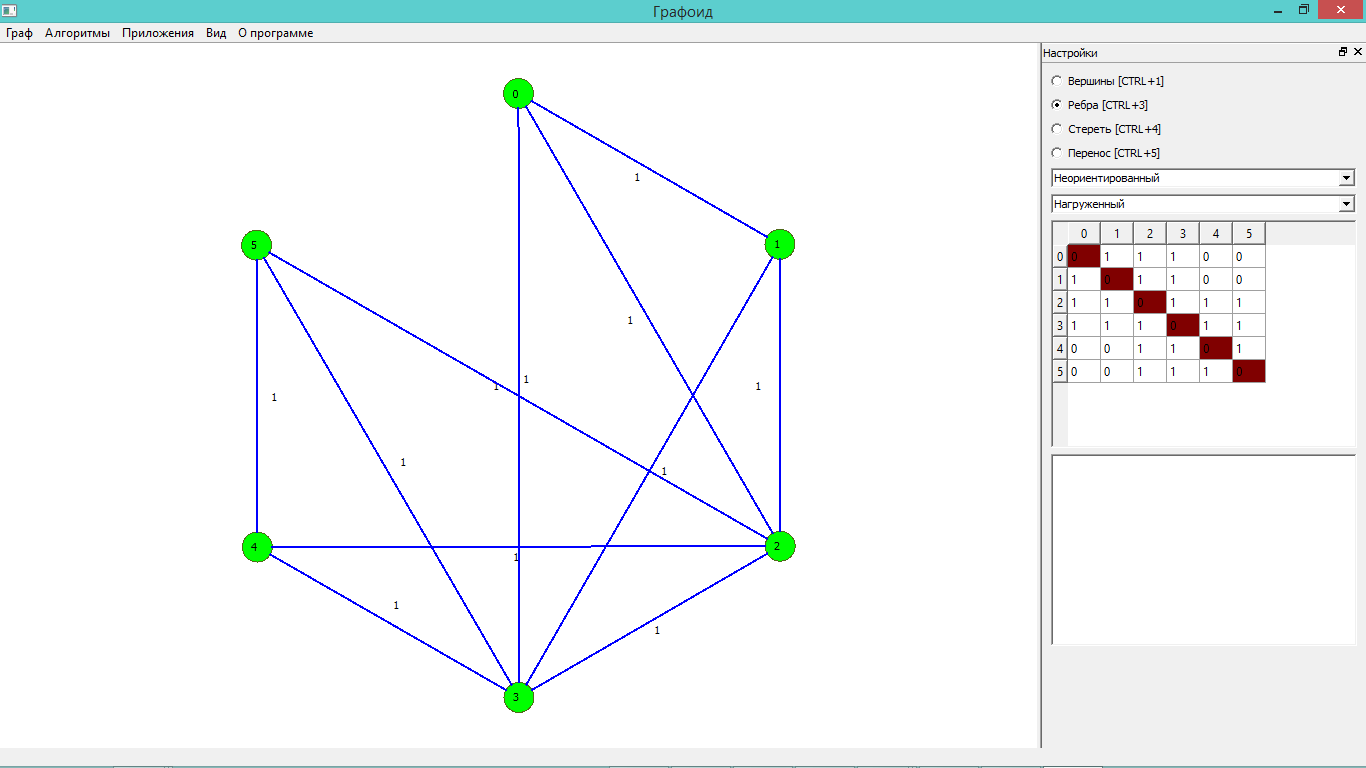
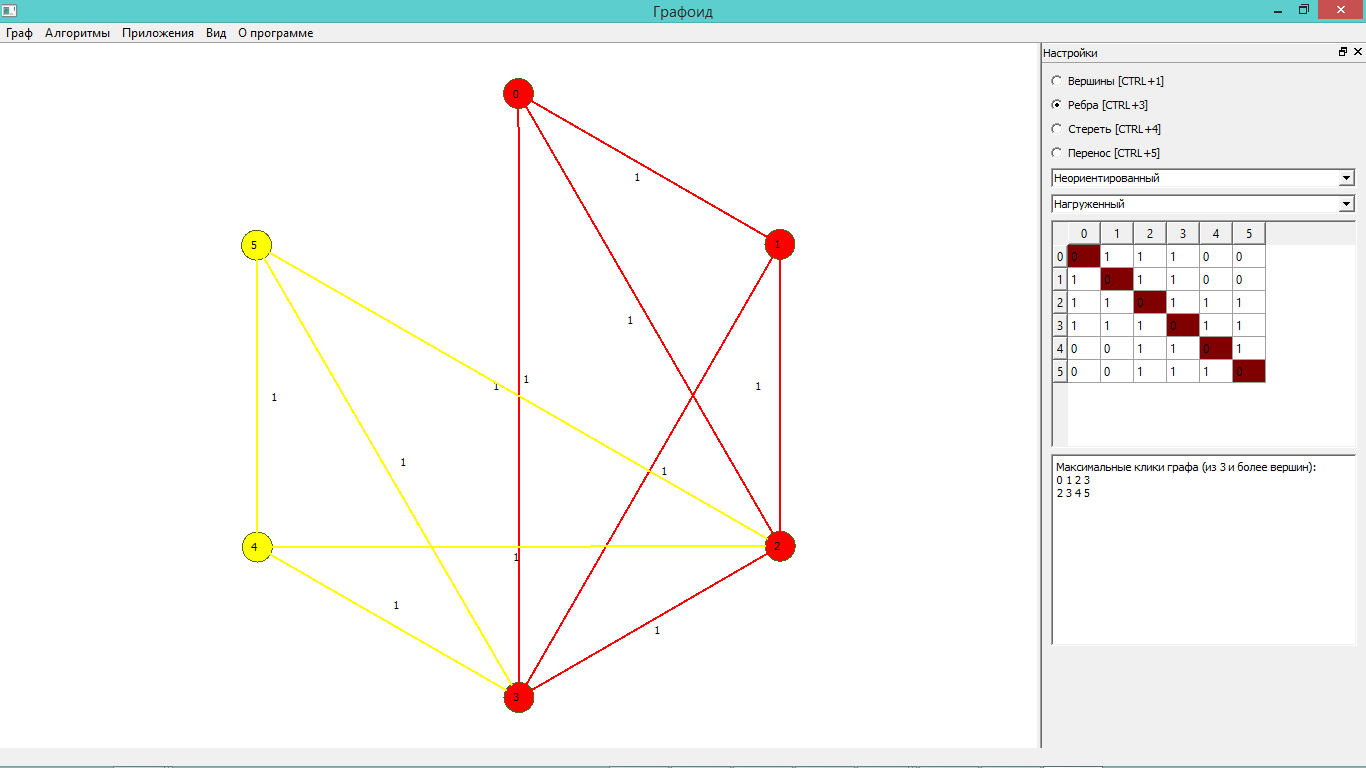




Пример 4.

Пример 5.

### Код программы

#include <fstream>

#include <vector>

#include <set>

using namespace std;

int \*\*matrix; // матрица смежности графа

int dim; // размерность матрицы смежности

vector < set <int> > result; // вектор множеств, содержащих вершины макс. клик

bool no\_node\_connected\_to\_all(set <int> candidates, set <int> used); // функция проверки наличия в used вершины, соединённой со всеми вершинами в candidates

set <int> maxclique(set <int> clique, set <int> candidates, set <int> used); // функция рекурсивного поиска макс. клик

int main(int argc, char \*argv[])

{

ifstream in(argv[1]); // входной поток из файла

in >> dim; // считываем размерность матрицы (количество вершин)

// выделяем память под матрицу смежности, заполняем её из файла

matrix = new int \*[dim];

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

{

matrix[i] = new int[dim];

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

in >> matrix[i][j];

}

in.close();

/////////////////////

// вводим множества

set <int> clique; // искомые вершины макс. клики

set <int> candidates; // вершины, которые могут увеличить clique

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

{

candidates.insert(i);

}

set <int> used; // вершины, уже использованные для расширения клики на предыдущий этапах

maxclique(clique, candidates, used); // запускаем рекурсивный поиск макс. клик, списки вершин запишутся в вектор results

/////////////////////

char color[5][20] = { " Red", " Yellow", " Magenta", " DarkGreen", " Gray" }; // массив с обозначениями цветов

int \*nodes = new int[dim]; // массив для соотнесения вершин с номерами макс. клик

int \*\*edges = new int\*[dim]; // массив для соотнесения рёбер с номерами макс. клик

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

{

nodes[i] = 0;

edges[i] = new int[dim];

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

edges[i][j] = 0;

}

for (unsigned int i = 0; i < result.size(); ++i)

{

for (set <int> ::iterator j = result[i].begin(); j != result[i].end(); ++j)

{

// сохранение номера макс. клики, в которую входит вершина

if (nodes[\*j] != 0)

{

if (result[i].size() > result[nodes[\*j] - 1].size())

nodes[\*j] = i + 1;

}

else

nodes[\*j] = i + 1;

for (set <int> ::iterator k = j; k != result[i].end(); ++k)

{

if (matrix[\*j][\*k]) // если есть ребро

{

// сохранение номера макс. клики, в которую входит ребро

if (edges[\*j][\*k] > 0)

{

if (result[i].size() > result[edges[\*j][\*k] - 1].size())

{

edges[\*j][\*k] = i + 1;

edges[\*k][\*j] = i + 1;

}

}

else

{

edges[\*j][\*k] = i + 1;

edges[\*k][\*j] = i + 1;

}

}

}

}

}

/////////////////////

ofstream out(argv[1]);

out << dim << endl; // записываем размерность матрицы

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

{

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

out << matrix[i][j] << ' ';

out << endl;

}

out << "Colors\_Nodes:" << endl;

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

{

out << i;

if (nodes[i])

out << color[(nodes[i] - 1) % 5];

else

out << " Green";

out << endl;

}

out << "Colors\_Edges:" << endl;

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

{

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

{

if (matrix[i][j])

{

out << i << ' ' << j;

if (edges[i][j] != 0)

out << color[(edges[i][j] - 1) % 5];

else

out << " Blue";

out << endl;

}

}

}

// заполняем текстовое поля

out << "Text:" << endl << "Максимальные клики графа (из 3 и более вершин):" << endl;

for (unsigned int i = 0; i < result.size(); ++i)

{

for (set <int> ::iterator j = result[i].begin(); j != result[i].end(); ++j)

{

out << \*j << ' ';

}

out << endl;

}

out.close();

// освобождение памяти

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

{

delete[] matrix[i];

delete[] edges[i];

}

delete[] matrix;

delete[] nodes;

delete[] edges;

return 0;

}

bool no\_node\_connected\_to\_all(set <int> candidates, set <int> used)

{

bool ans;

for (set <int> ::iterator i = used.begin(); i != used.end(); ++i)

{

ans = true;

for (set <int> ::iterator j = candidates.begin(); j != candidates.end(); ++j)

{

if (!matrix[\*i][\*j]) // если где-то нет связи

{

ans = false;

break;

}

}

if (ans)

return false; // если флаг не поменялся, найдена вершина, соединённая со всеми в candidates

}

return true;

}

set <int> maxclique(set <int> clique, set <int> candidates, set <int> used)

{

// пока candidates не пустое и used НЕ содержит вершины, соединённой со всеми вершинами в candidates

while (candidates.size() != 0 && no\_node\_connected\_to\_all(candidates, used))

{

// выбираем вершину

int node = \*candidates.begin();

clique.insert(node);

// формируем new\_candidates и new\_not, удаляя из candidates и not вершины, не соединённые с node

set <int> new\_candidates = candidates;

set <int> new\_used = used;

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

{

if (!matrix[i][node])

{

new\_candidates.erase(i);

new\_used.erase(i);

}

}

// если new\_candidates и new\_not пусты

if (new\_candidates.size() == 0 && new\_used.size() == 0 && clique.size() > 2)

result.push\_back(clique);

else

clique = maxclique(clique, new\_candidates, new\_used);

// удаляем v из clique и candidates, и помещаем в not

clique.erase(node);

candidates.erase(node);

used.insert(node);

}

return clique;

}