МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования   
**«Национальный исследовательский   
Нижегородский государственный университет им. Н.И. Лобачевского»**

**(ННГУ)**

**Институт информационных технологий, математики и механики**

**Кафедра: Информатики и автоматизации научных исследований**

Направление подготовки: «Прикладная информатика»

Профиль подготовки: «Прикладная информатика в области обработки данных»

**ОТЧЕТ ПО ПРЕДДИПЛОМНОЙ ПРАКТИКЕ**

на тему:

**Нахождение максимального независимого**

**множества вершин**

Выполнил: студент группы

\_\_\_\_\_\_\_\_\_\_\_\_381507в\_\_\_\_\_\_\_\_

\_\_Овсянников Сергей Олегович \_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись

Нижний Новгород

2019

**Содержание**

* Обзор методов решения задачи
  + Обзор и классификация методов решения.
  + Выбор алгоритма
* Алгоритм решения задачи
* Программная реализация и эксперимент
* Заключение и список источников
* Приложение

1. **Обзор методов решения задачи:**
   1. **Обзор и классификация методов решения.**

* Один из вариантов – является полный перебор всех возможных подграфов размера k с проверкой того, является ли хотя бы один из них полным. Как уже было сказано выше – этот алгоритм неэффективен, на графе с количеством вершин более 20.
* Другой алгоритм работает так: две клики размера *n* и *m* «склеиваются» в большую клику размера *n+m*, причём кликой размера *1* полагается отдельная вершина графа. Алгоритм завершается, как только ни одного слияния больше произвести нельзя. Время работы данного алгоритма линейно, однако он является эвристическим, поскольку не всегда приводит к нахождению клики максимального размера. В качестве примера неудачного завершения можно привести случай, когда вершины, принадлежащие максимальной клике, оказываются разделены и находятся в кликах меньшего размера, причём последние уже не могут быть «склеены» между собой.
* Однако есть ещё и третий вариант. Он работает следующим образом: сортируем вершины по количеству выходящих из них рёбер. Выбираем вершину с наименьшим количество выходящих рёбер, зануляем вершины смежные выбранной
* Алгоритм Брона-Кербоша: Алгоритм использует тот факт, что всякая клика в графе является его максимальным по включению [полным подграфом](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D0%BD%D1%8B%D0%B9_%D0%B3%D1%80%D0%B0%D1%84). 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](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_%D0%B2%D0%B5%D1%82%D0%B2%D0%B5%D0%B9_%D0%B8_%D0%B3%D1%80%D0%B0%D0%BD%D0%B8%D1%86) 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.
  1. **Выбор алгоритма**

Для решения поставленной задачи мной был выбран алгоритм Брона-Кербоша. Он даёт точное решение и в отличии от полного перебора работает за линейное время O(n), где n – количество независимых множеств. В худшем случае данный алгоритм имеет время выполнения - *O*(3*n*/3), где n –количество вершин

1. **Алгоритм решения задачи:**

В предыдущем пункте мы выбрали алгоритм Брона-Кербоша.  
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 **not** — множество вершин, которые уже использовались для расширения **compsub** на предыдущих шагах алгоритма.

Алгоритм является [рекурсивной](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BA%D1%83%D1%80%D1%81%D0%B8%D1%8F) процедурой, применяемой к этим трем множествам.

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

**ПОКА** *candidates* НЕ пусто **И** *not* НЕ содержит вершины, СОЕДИНЕННОЙ СО ВСЕМИ вершинами из *candidates*,

**ВЫПОЛНЯТЬ**:

1 Выбираем вершину *v* из *candidates* и добавляем её 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*)

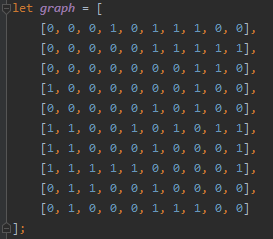
6 Удаляем v из *compsub* и *candidates*, и помещаем в *not*

В классическом алгоритме используется случайная выборка.

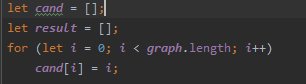
1. **Программная реализация и эксперимент**

Для реализации данного алгоритма мной используется язык программирования JavaScript, спецификация EcmaScript2015.

В начале зададим граф.



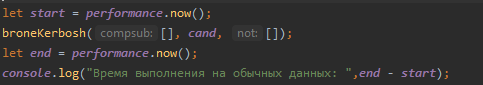
Создаём массив кандидатов и массив результатов работы нашей программы. Инициализируем массив кандидатов.

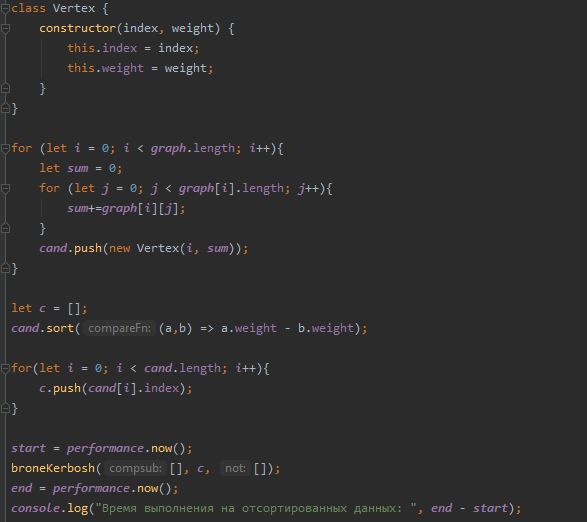


Реализация алгоритма Брона-Кербоша.



Вызываем функцию для нашей тестовой матрицы и выводим время выполнения:



Сортируем данные и запускаем наш алгоритм ещё раз:  


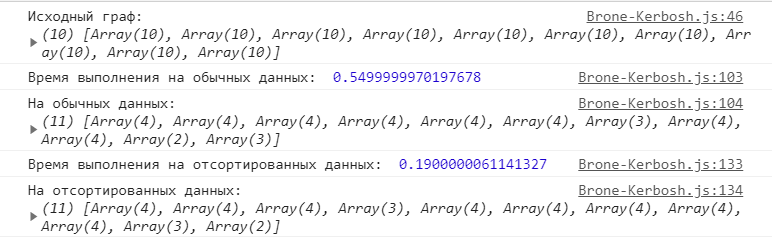
Результаты:

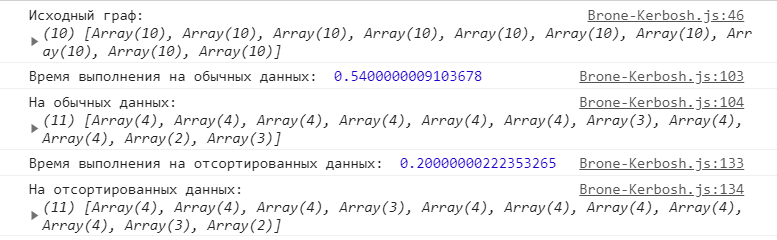
1)



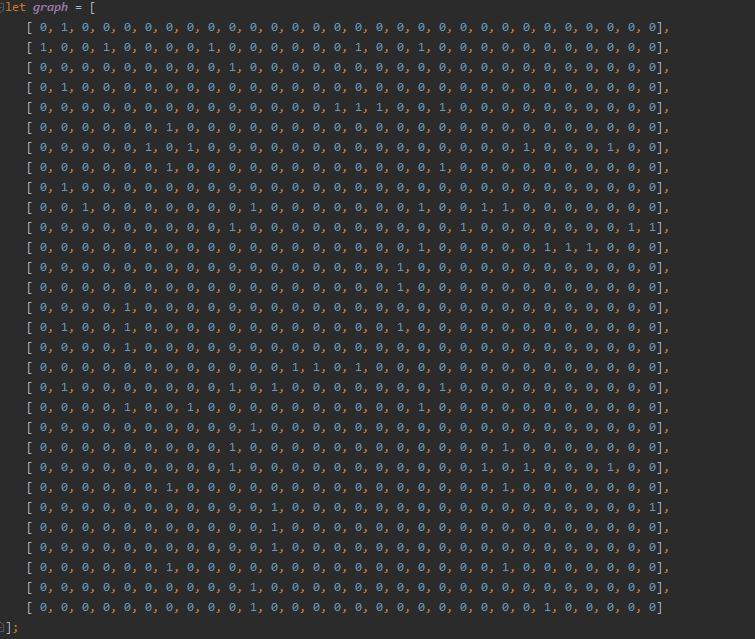


2)



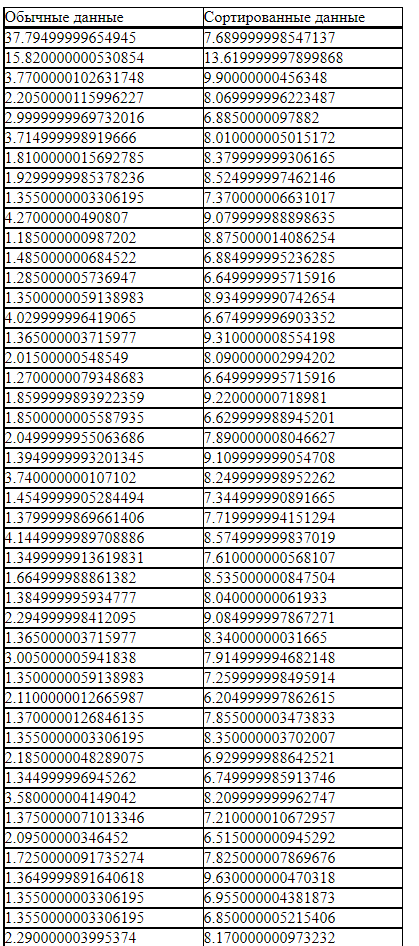
3) 

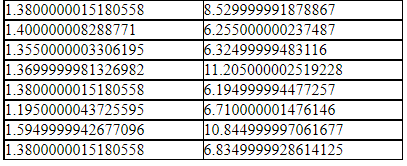
На графе с 30 вершинами:



****

**Результаты 100 запусков для графа из 30 вершин.**

****

****

**Заключение**

В рамках дипломной работы была поставлена проблема о решении задачи независимого множества вершин максимальной мощности. Во время разбора задачи было найдено несколько возможных алгоритмов решения. Один из этих алгоритмов был реализован и подробно разобран мной. Мною выбранный алгоритм является одним из самых оптимальных. В ходе работы было доказано, что алгоритм Брона-Кербоша не зависит от порядка входных данных и выполняется, приблизительно одинаковое время.  
В ходе работы была решена индивидуальная задача.

**Список источников**

Литература:

* Иванов Б. – «Дискретная математика. Алгоритмы и программы»
* N. Christofides "Теория графов. Алгоритмический подход."

Онлайн ресурсы:

* [http://rain.ifmo.ru](http://rain.ifmo.ru/cat/view.php/theory/graph-coloring-layout/covering-2004)
* ru.wikipedia.org

**Приложение:**

#pragma once

#include <iostream>

#define V 10

struct Vertex

{

int countedge;

int vertnumber;

};

class Pack

{

public:

Pack() {

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

{

Vert[i].countedge = 0;

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

{

matrix[i][j] = 0;

}

}

};

void findvert();

void sortVert();

void packing();

void print();

void makegraph(int(&a)[V][V]);

private:

void makegraph();

Vertex Vert[V];

int matrix[V][V];

};

#include "Packing.h"

void Pack::findvert()

{

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

{

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

{

Vert[i].vertnumber = i;

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

{

Vert[i].countedge += 1;

}

}

}

}

void Pack::sortVert()

{

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

{

for (int j = i + 1; j < V; j++)

{

Vertex tmp;

if (Vert[j].countedge < Vert[i].countedge)

{

tmp = Vert[j];

Vert[j] = Vert[i];

Vert[i] = tmp;

}

}

}

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

{

std::cout << "вершина №" << (Vert[i].vertnumber + 1)

<< " - содержит рёбер " << Vert[i].countedge << std::endl;

}

packed[0] = Vert[0].vertnumber;

}

void Pack::packing()

{

std::cout << "Максимальное независмое множество: ";

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

{

if (Vert[i].vertnumber != -1)

{

std::cout << Vert[i].vertnumber + 1 << " \n";

}

for (int j = 0; j < (V-1); j++)

{

if (matrix[Vert[i].vertnumber][Vert[j].vertnumber] == 1)

{

Vert[j] = Vert[j + 1];

Vert[j].vertnumber = -2;

Vert[V-1].vertnumber = -2;

for (int g = 0; g < V; g++)// вывод на каждом шаге

{

std::cout << "вершина №" << (Vert[g].vertnumber + 1) << " - содержит рёбер " << Vert[g].countedge << std::endl;

}

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

}

}

}

std::cout << std::endl;

}

void Pack::print()

{

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

{

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

{

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

}

std::cout << std::endl;

}

}

void Pack::makegraph(int(&a)[V][V])

{

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

{

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

{

matrix[i][j] = a[i][j];

}

}

}

#include "Packing.h"

void main()

{

setlocale(LC\_ALL, "RUS");

int matr[V][V] = {

{ 0, 0, 0, 1, 0, 1, 1, 1, 0, 0 },

{ 0, 0, 0, 0, 0, 1, 1, 1, 1, 1 },

{ 0, 0, 0, 0, 0, 0, 0, 1, 1, 0 },

{ 1, 0, 0, 0, 0, 0, 0, 1, 0, 0 },

{ 0, 0, 0, 0, 0, 1, 0, 1, 0, 0 },

{ 1, 1, 0, 0, 1, 0, 1, 0, 1, 1 },

{ 1, 1, 0, 0, 0, 1, 0, 0, 0, 1 },

{ 1, 1, 1, 1, 1, 0, 0, 0, 0, 1 },

{ 0, 1, 1, 0, 0, 1, 0, 0, 0, 0 },

{ 0, 1, 0, 0, 0, 1, 1, 1, 0, 0 },

};

Pack p;

p.makegraph(matr);

p.findvert();

p.sortVert();

p.print();

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

p.packing();

}