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 ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ**

на тему:

**Нахождение максимального независимого**

**множества вершин**

Выполнил: студент группы

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

\_\_Овсянников Сергей Олегович \_

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

подпись

Нижний Новгород

2019

**Содержание**

* Введение
* Постановка задачи
  + Исходные данные
  + Математическая модель задачи
  + Критерий задачи и система ограничений
  + Математическая сложность задачи
* Обзор методов решения задачи
  + Обзор и классификация методов решения.
  + Выбор алгоритма
* Алгоритм решения задачи
* Программная реализация и эксперимент
* Заключение и список источников
* Приложение

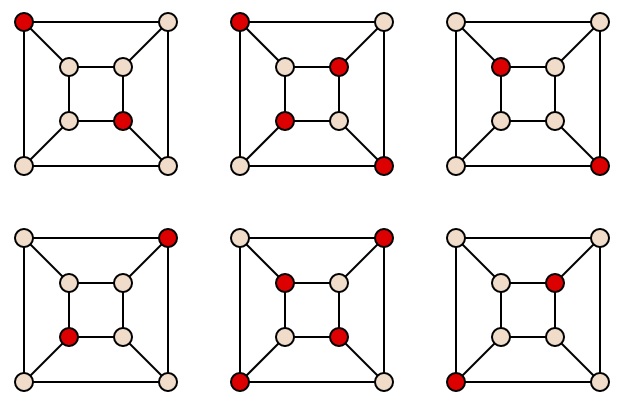
1. **Введение**

Целью моей дипломной работы является описание, нахождение и обзор алгоритма поиска в графе независимого множества вершин максимальной мощности.

Задача поиска максимального множества независимых вершин(оно же внутренне устойчивое множество) относится к классу NP-полных и, как и для других NP-полных задач, эффективного алгоритма для поиска множества независимых вершин на данный момент не найдено. Эквивалентной для неё считается задача о клике. Фактически клика на графе G является максимальным независимым множеством вершин на дополнении этого графа G’.

Внутренне устойчивое множество – это множество вершин графа G такое, что любые две вершины в нём не являются смежным, то есть никакая пара вершин не соединена ребром.

Следующий рисунок показывает примеры наибольших независимых множеств вершин (отмечены красным) в данном случае их 6.



Задача поиска множества независимых вершин максимальной мощности имеет применение и в реальной жизни, например для оптимизации рабочего процесса команды специалистов. Например, каждая вершина графа обозначает один проект, соответственно ребро между двумя вершинами – обозначает средства необходимые для выполнения обоих этих задач.

1. **Формальная постановка задачи:**
   1. **Исходные данные:**

Пусть дан связный граф G=(V,E), где V – множество вершин, Е – множество ребер. Необходимо найти такое подмножество вершин 𝑉1, что для любой пары вершин из этого подмножества не существует связывающего их ребра, т.е. найти 𝑉1 такое, что ∀𝑣𝑖∈𝑉1 ,𝑣𝑗∈𝑉1 ∄𝑒∈𝐸,что 𝑒 = (𝑣𝑖,𝑣𝑗).

Задача состоит в поиске подмножества максимальной мощности.

* 1. **Математическая модель задачи:**

Для построения математической модели я использовал задачу выбора проекта. Предположим, мы имеем n проектов, которые должны быть выполнены. Для выполнения проекта vi необходимо некоторое подмножество Ri имеющихся ресурсов из множества {1, … , p}. Пусть каждый проект, в нашей задаче, может быть выполнен за один и тот же промежуток времени. Нам нужно построить граф G, таким образом, что каждая вершина графа соответствует некоторому проекту, а ребро {vi, vj} наличию общих средств необходимых для реализации проекта vi и vj. Максимальное независимое множество графа G представляет максимальное множество проектов, которое можно выполнять параллельно. Итак, нам понадобится:

* Список проектов, обозначенный V={vi}, i = {1, … , n}, как множество вершин матрицы, где n – количество вершин графа.
* Список общих средств, обозначенный E={Xi} – множество рёбер, где {0, 1}, в зависимости от наличия общих средств выполнения между проектами vi и vj и i={1, … , n}.

Первое требование, которое нам выдвигает условия – все проекты не должны использовать общие методы решения. То есть никакая пара вершин не соединена ребром:  
**S ∩ E(S) = Ø**

* 1. **Критерий задачи:**

Система ограничений задачи:

* **S ∩ E(S) = Ø –** в графе нет ни одной пары смежных вершин
* **H ∩ E(H) ≠ Ø   ∀ H ⊃ S  –** в графе нет другого независимого множества, в которое бы оно входило.
  1. **Математическая сложность задачи:**

Как уже было сказано выше, нахождение наибольшего независимого множества вершин – относится к классу NP-полных задач, что значит, что последовательный перебор всех возможных подграфов размера k - эффективен только в случае небольшого количества вершин, т.к. число таких подграфов в графе с v вершинами равно биномиальному коэффициенту 

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.

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*)

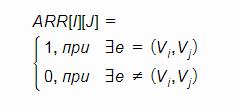
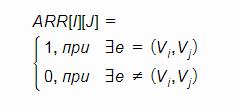
6 Удаляем v из *compsub* и *candidates*, и помещаем в *not*

* 1. **Выбор алгоритма**

Для решения поставленной задачи мной был выбран жадный алгоритм, то есть 3й по счёту вариант. В отличии от первого, он не ищет все независимые множества вершин, но этого в поставленной задаче и не требуется. Плюсом в копилку жадного алгоритма является и его скорость работы, алгоритм способен работать за полиномиальное время O(N^k).

1. **Алгоритм решения задачи:**

Итак, в пункте 3.2 нами был выбран алгоритм с использованием максимального паросочетания. Из этого следует, что наш алгоритм будет состоять их двух частей: нахождение множества вершин и его сортировка, а после жадный алгоритм для добавления непокрытых вершин.

Пусть у нас есть ARR - матрица смежности графа размером N x N. Элемент матрицы может принимать значения [0,1] в зависимости от наличия смежности между двумя вершинами: 0, если две вершины не смежны (не имеют общего ребра), 1 - иначе.   


Для нахождения максимального по мощности множества вершин я выбрал простой "жадный" алгоритм по нескольким причинам:

* Жадный алгоритм является одновременно простым и эффективным. Несмотря, на классификацию как рандомизированного, он часто является оптимальным решением, в особенности для канонических систем исчисления.
* Выбор алгоритма никак не влияет на ход и время нахождения максимального множества вершин, т.е. последующее включение вершин должно работать одинаково, вне зависимости от реализации первого пункта.

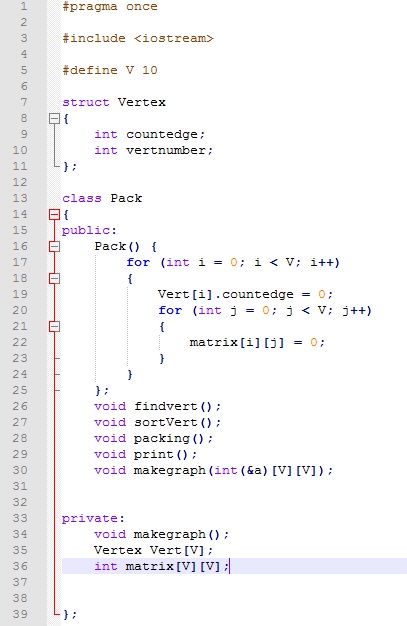
Итак, для начала опишем ход "жадного алгоритма".

Сделаем несколько итераций для нашей матрицы. В каждой итерации будем проводить следующий алгоритм:

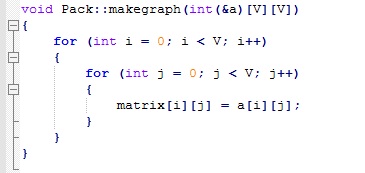
* Составим множество всех вершин.
* Отсортируем это множество.
* Обнуляем в множестве вершин те вершины, что смежны с выбранной

1. **Программная реализация и эксперимент**

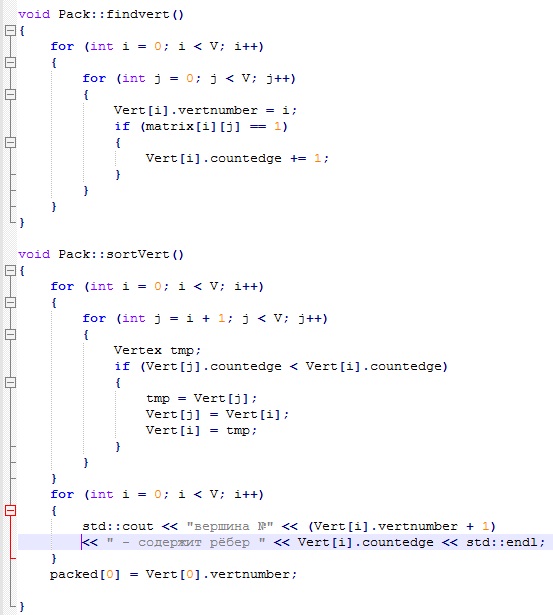
В первую очередь опишем класс, в котором будет храниться наш граф.



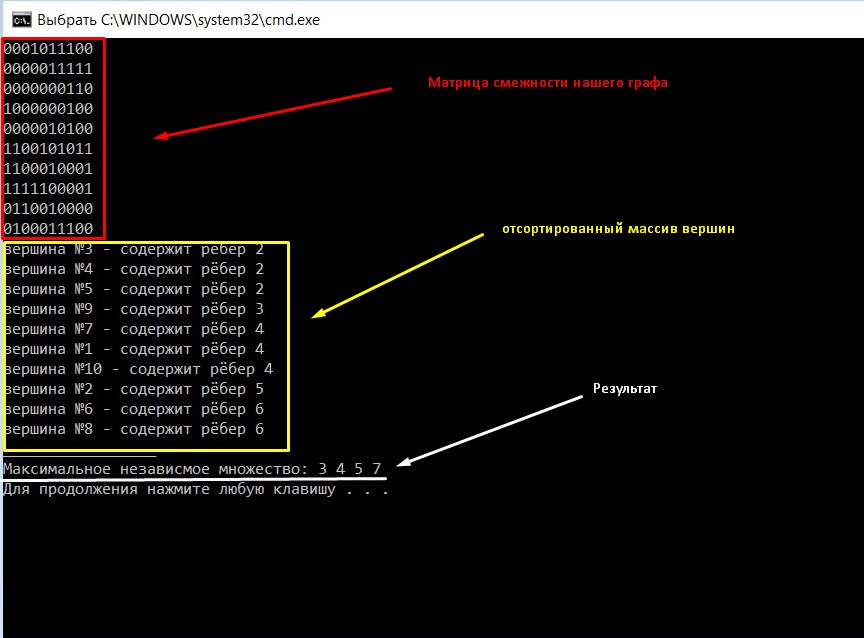
Функция для создания графа



Функции поиска количества рёбер в вершинах и их сортировки.



Пропускаем нашу тестовую матрицу 10х10:



**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 не является оптимальным, т.к. оптимальных алгоритмов для поиска максимального независимого множества – не существует.  
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();

}