

**Оглавление**

[**Реферат** 5](#_Toc185410551)

[**Введение** 6](#_Toc185410552)

[**1.** **Постановка задачи** 8](#_Toc185410553)

[**2.** **Теоретическая часть задания** 9](#_Toc185410554)

[**3.** **Описание алгоритма программы** 11](#_Toc185410555)

[**4.** **Описание программы** 14](#_Toc185410556)

[**5.** **Тестирование** 23](#_Toc185410557)

[**6.**  **Ручной расчет задачи** 33](#_Toc185410558)

[**Заключение** 34](#_Toc185410559)

[**Список литературы** 35](#_Toc185410560)

[**Приложение А. Листинг программы** 36](#_Toc185410561)

# **Реферат**

Отчет 39 стр., 23 рисунка.

ГРАФ, ТЕОРИЯ ГРАФА, ПОИСК МАКСИМАЛЬНОГО ПАРОСОЧЕТАНИЯ, АЛГОРИТМ ХОПКРОФТА-КАРПА

Цель исследования – разработка программы, реализующей алгоритм поиска наибольшего паросочетания.

Теория графов — раздел дискретной математики, изучающий графы, одна из ветвей топологии.

В самом общем смысле граф — это множество точек (вершин, узлов), которые соединяются множеством линий (рёбер, дуг).

Поиск максимального паросочетания — это важная концепция в теории графов и дискретной математике, которая находит применение в различных областях программирования и компьютерных наук:

- решение задач совместимости;

- планирование и распределение ресурсов;

- оптимизация использования каналов связи;

- методы визуализации;

- параллельное вычисление;

- теория сложных систем.

Универсальность графов незаменима при проектировании и анализе коммуникационных сетей.

Максимальное паросочетание в графе — это такое подмножество рёбер, что никакие два ребра не имеют общих вершин. В задачах комбинаторики, теории графов и оптимизации часто возникает необходимость найти максимальное паросочетание в графе.

Задачи на раскраски графа возникают при проектировании коммуникаций, в радиоэлектронике, в планировании эксперимента и других областях.

В работе рассмотрен алгоритм Хопкрофта – Карпа, который не так уж прост, но эффективен.

# **Введение**

Алгоритм Хопкрофта Карпа – это довольно интуитивный и понятный метод. Он основан на понятии увеличивающего пути — пути, который начинается и заканчивается в свободной вершине, а внутри пути рёбра, принадлежащие и не принадлежащие паросочетанию, чередуются.

**Преимущества:**

* + - Эффективность: имеет | временную сложность, где E – множество ребер, V – множество его вершин.
    - Точность: алгоритм Хопкрофта-Карпа является одним из наиболее точных алгоритмов поиска максимального паросочетания в двудольном графе.
    - Универсален: алгоритм может быть легко расширен для работы с различными типами графов и задачами.

**Недостатки:**

* + Сложность реализации: реализация данного алгоритма может потребовать определённых знаний и опыта в области алгоритмов и структур данных.
  + Ограниченность: алгоритм Хопкрофта-Карпа работает только с двудольными графами, поэтому не подходит для решения задач с другими типами графов.
  + Требовательность к вычислительным ресурсам: в некоторых случаях алгоритм может потреблять большое количество памяти и вычислительных ресурсов, особенно при работе с большими графами.

В качестве среды разработки была выбрана среда Microsoft Visual Studio 2022, языки программирования – С++.

Целью данной курсовой работы является разработка программы на языке C++, который является широко используемым. Именно с его помощью была реализована программа целиком – логика алгоритма, вывод графа и т.п.

Алгоритм поиска максимального паросочетания является важным, так как он позволяет находить оптимальное совпадение между элементами двух различных множеств, что может быть полезно в различных задачах и приложениях, таких как планирование расписания, оптимизация транспортных маршрутов, распределение ресурсов и многие другие.

1. **Постановка задачи**

Требуется разработать программу, которая будет находить максимальное количество паросочетаний.

Исходный граф в программе должен задаваться двумя подмножествами U и V. Программа должна работать так, чтобы пользователь вводил количество вершин графа для формирования подмножеств. После обработки этих данных на экран должны выводится вершины подмножеств U и V, а затем и сам граф. Необходимо предусмотреть проверку ввода, чтобы программа не выдавала ошибок и работала правильно.

Алгоритм должен принимать на входе целое число, которое в последующем послужит для работы всей программы. Число должно задаваться без запятых, знаков и других символов. Программа также должна иметь возможность продолжения работы программы без повторного запуска.

Устройство ввода – клавиатура и мышь.

1. **Теоретическая часть задания**

Граф G задается подмножеством вершин U1, U2, …, Un и V1, V2, ..., Vn и множеством ребер, соединяющих между собой определенные вершины.

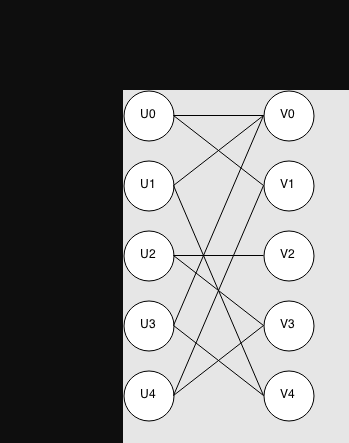
****

Рисунок 1 – Граф

При представлении графа матрицей смежности информация о ребрах графа хранится в квадратной матрице, где присутствие пути из одной вершины в другую обозначается единицей, иначе нулем.

Существует много алгоритмов на графах, в основе которых лежит систематический перебор вершин графа, такой, что каждая вершина графа просматривается только один раз, и переход от одной вершины к другой осуществляется по ребрам графа.

Поиск максимального паросочетания в графе является важной задачей в теории графов. Паросочетание в графе - это набор несмежных рёбер, такой, что каждая вершина графа является концом ровно одного ребра из паросочетания. Максимальное паросочетание - это паросочетание максимальной мощности.

Идея алгоритма заключается в последовательном увеличении паросочетания путём нахождения удлиняющего пути. Удлиняющий путь - это путь в графе, начинающийся и заканчивающийся на свободной вершине (не в паросочетании), и включающий чередующиеся рёбра (то есть чередуются рёбра из паросочетания и не из паросочетания). Путём последовательного нахождения удлиняющего пути в паросочетание, алгоритм находит максимальное паросочетание в графе.

1. **Описание алгоритма программы**

Функция алгоритм Хопкрофта Карпа

создать pairU = новый массив размером [количество вершин U + 1], pairV = новый массив размером [количество вершин V + 1], dist = новый массив размером [количество вершин U + 1]

для каждого элемента pairU

pairU[i] = 0

для каждого элемента pairV

pairV[i] = 0

result = 0

пока (BFS())

для каждого элемента pairU

Если вершина свободна и dfs(вершины)

result ++

вернуть result;

Функция BFS()

Функция bfs():

Создать пустую очередь q

Для каждого элемента i от 1 до размера u:

если pairU[i] равно 0:

dist[i] = 0

добавить i в очередь q

иначе:

dist[i] = ∞

dist[0] = ∞

Пока очередь q не пуста:

извлечь первый элемент s из очереди q

если dist[s] меньше чем dist[0]:

для каждого элемента i из u:

k = 0

для каждого элемента j от 0 до количества вершин:

если G[i][j] равно 1:

если dist[pairV[k]] равно ∞:

dist[pairV[k]] = dist[s] + 1

добавить pairV[k] в очередь q

увеличить k на 1

вернуть (dist[0] не равно ∞)

Функция DFS(целое t)

если t не равно 0:

для каждого элемента i из u:

k = 0

Для каждого элемента j от 0 до len:

если G[ss - 1][j] равно 1:

если dist[pairV[k]] равно dist[t - 1] + 1:

если вызов DFS(pairV[k]) равен true:

pairV[k] = t

pairU[t] = j

вернуть true

увеличить k на 1

dist[t] = 0

вернуть false

вернуть true

Программа начинает работу с запроса у пользователя количества вершин в графе. Ввод должен быть целым неотрицательным числом. Если пользователь вводит некорректное значение (например, символ или отрицательное число), программа выдаёт сообщение об ошибке и запрашивает ввод снова.

Сначала вводится длина графа и распределяются вершины по двум подмножествам (u и v) случайным образом. Затем создается и заполняется матрица смежности графа G, где 1 означает ребро между вершинами, а 0 - отсутствие ребра.

После создания графа программа переходит к поиску максимального паросочетания. Поиск реализуется с помощью алгоритма Хопкрофта Карпа.

Создается два массива pairU и pairV, а также массив dist размером. В этих массивах будут храниться пары вершин в сочетании и расстояние между левыми боковыми вершинами. Далее первые 2 массива заполняются нулями. Затем запускается цикл while до тех пор пока существует увеличивающий путь, который ищется с помощью функции обхода в ширину. После запускается цикл который идет по всем вершинам u, и проверяется, если вершина свободна и есть возможность расширить путь от нее с помощью функции обхода в глубину, то результат увеличивается на единицу. После завершение функции программа выводит результат.

После завершения завершения поиска программа предлагает пользователю выбрать, завершить ли алгоритм или начать заново. Если пользователь выбирает начать заново, то запускается функция обнуления массивов и вызывает заново функцию меню, иначе программа завершается.

1. **Описание программы**

Работа программы начинается с запроса количества вершин графа. Далее пользователю предлагается выбор: запуск алгоритма или завершить выполнение программы. Если пользователь выбирает запуск алгоритма, то запрашивается количество вершин. После выводится матрица смежности графа, вектора U и V, и сразу же выводится результат алгоритма.

Для написания данной программы использованы языки C++. Язык программирования C++ был использован для реализация алгоритма создания матрицы смежности, на основе введенного количества вершин, а также сам алгоритм раскраски графа. Были использованы функции printG, find\_part, creatG, bfs, dfs, Hopkroft\_Karp\_alg, zeroing, menu.

Общее представление программы на диаграмме модулей:

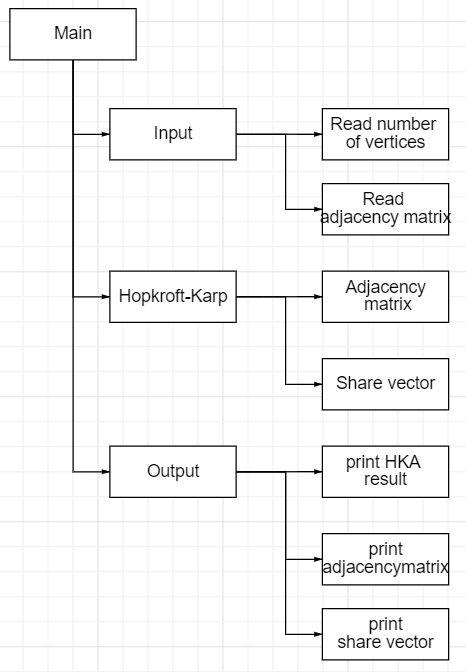
****

Рисунок 2- Диаграмма модулей

*Функция printG*

Код:

void printG() {

cout << "\tГраф G: \n\n";

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

cout << "\t";

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

cout << G[i][j] << " ";

}

cout << "\n";

}

cout << "\n\tВершины подмножества U: ";

for (int i = 0; i < u.size(); i++) cout << u[i] << " ";

cout << "\n\tВершины подмножества V: ";

for (int i = 0; i < v.size(); i++) cout << v[i] << " ";

}

Данная функция выводит матрицу смежности, а затем выводит вектора U и V.

Логика работы:

1. два вложенных цикла перебирают все элементы матрицы смежности и выводит их;

2. цикл выводит все значения вектора U, а затем так же все значения вектора V.

*Функция find\_part*

Код:

bool find\_part(int s) {

for (int i = 0; i < v.size(); i++) {

if (v[i] == s) return true;

}

return false;

}

Эта функция проверяет принадлежит ли вершина подмножеству.

Логика работы:

входной параметр: s – вершина, которую надо проверить;

1. цикл проходит по всем элементам подмножества V;

2. если вершина равна s какому-либо элементу V, вернуть true, иначе false;

*Функция creatG*

Код:

void creatG() {

//Ввод длины графа

while(1){

cout << "\n\tВведите длину графа: ";

cin >> len;

if (len <= 1) cout << "\n\tНекорректно введенные данные, попробуйте снова!\n";

else break;

}

srand(time(0));

//Распределение вершин по подмножествам

while(1){

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

if (rand() % 11 > 5) u.push\_back(i + 1);

else v.push\_back(i + 1);

}

if (v.size() < 1 || u.size() < 1 || u.size() < v.size()) { u.clear(); v.clear(); continue; }

else break;

}

//Инициализация и заполнение массива

G = new int\* [len];

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

G[i] = new int[len];

}

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

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

G[i][j] = 0;

//Проложение путей между вершинами

vector <int>::iterator j;

srand(time(0));

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

if (find\_part(i + 1)){

for (j = u.begin(); j != u.end(); ++j) {

int s = \*j;

if (s - 1 == i) continue;

if (rand() % 11 < 5) G[i][s - 1] = G[s - 1][i] = 1;

}

}

else {

for (j = v.begin(); j != v.end(); ++j) {

int s = \*j;

if (s - 1 == i) continue;

if (rand() % 11 < 5) G[i][s - 1] = G[s - 1][i] = 1;

}

}

}

printG();

}

Данная функция предназначена разбить вершины на 2 подмножества, инициализировать матрицу смежности и проложить пути между вершинами.

Логика работы:

1. в цикле с вероятностью 50% вершина попадает либо в вектор U, либо V;
2. инициализация массива и заполнение всех его элементов нулями;
3. в цикле если вершина принадлежит вектору V, то запускается цикл, который идет по вершинам вектора U, и с вероятностью 50% создаются ребра между вершинами вектора U и вершиной V;
4. иначе запускается цикл, который идет по вершинам вектора V, и с вероятностью 50% создаются ребра между вершинами вектора V и вершиной U;
5. вызывается функция печати графа.

*Функция bfs*

Код:

bool bfs(){

queue <int> q;

for (int i = 1; i <= u.size(); i++) {

//Если свободная вершина то добавляем в очередь

if (pairU[i] == 0) {

dist[i] = 0;

q.push(i);

}

else dist[i] = INT\_MAX;

}

dist[0] = INT\_MAX;

while (!q.empty()) {

int s = q.front();

q.pop();

//Если вершина не равна нулю и может обеспечить более короткий путь к нулю

if (dist[s] < dist[0]) {

vector <int>::iterator i;

for (i = u.begin(); i != u.end(); ++i) {

int ss = \*i, k = 0;

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

if (G[ss - 1][j] == 1) {

if (dist[pairV[k]] == INT\_MAX) {

dist[pairV[k]] = dist[s] + 1;

q.push(pairV[k]);

}

k++;

}

}

}

}

}

return (dist[0] != INT\_MAX);

}

Назначение функции выполнить поиск в ширину, чтобы найти увеличивающий путь.

Логика работы:

1. создается очередь q, в которую добавляются начальные вершины;
2. инициализируются расстояния dist от каждой вершины до вершины 0. Если вершина является свободной (pairU[i] == 0), то ее расстояние устанавливается как 0 и добавляется в очередь q. В противном случае расстояние устанавливается как INT\_MAX;
3. пока очередь не пуста, извлекаем вершину s;
4. если расстояние от вершины s до вершины 0 меньше, чем расстояние от вершины 0 до вершины 0, то для каждой вершины ss из списка смежности u вершины s, рассматриваем смежные вершины pairV[k]. Если расстояние до вершины pairV[k] равно INT\_MAX, обновляем расстояние до нее и добавляем в очередь q;
5. по завершении обработки всех вершин в очереди, возвращается результат (dist[0] не равно INT\_MAX). Если вершина 0 не достижима из других вершин, функция возвращает false.

*Функция dfs*

Код:

bool dfs(int t){

if (t != 0) {

vector <int>::iterator i;

for (i = u.begin(); i != u.end(); ++i) {

int ss = \*i, k = 0;

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

if (G[ss - 1][j] == 1) {

if(dist[pairV[k]] == dist[t - 1] + 1){

if (dfs(pairV[k]) == true) {

pairV[k] = t;

pairU[t] = j;

return true;

}

}

k++;

}

}

}

dist[t] = 0;

return false;

}

return true;

}

Назначение функции в том, что она выполняет поиск в глубину для нахождения увеличивающего пути в графе из заданной вершины

Логика работы:

входной параметр: t – вершина из которой нужно начать обход;

1. проверяется, если вершина t не равна 0;
2. для каждой вершины ss из U (вектор) проверяется смежная вершина в графе G;
3. если смежная вершина соответствует условию и еще не посещена, рекурсивно вызывается функция dfs для этой вершины;
4. если рекурсивный вызов возвращает true, то происходит обновление пар совпадающих вершин pairV и pairU, и функция возвращает true;
5. если ни одна из смежных вершин не подходит, обнуляется dist[t] и возвращается false;

6. если вершина t равна 0, функция возвращает true.

*Функция Hopkroft\_Karp\_alg*

Код:

int Hopkroft\_Karp\_alg(){

//1 и 2 массивы хранят пару в сочетании, 3 - расстояние между левыми боковыми вершинами

pairU = new int[u.size() + 1], pairV = new int[v.size() + 1], dist = new int[u.size() + 1];

for (int i = 0; i <= u.size(); i++) pairU[i] = 0;

for (int i = 0; i <= v.size(); i++) pairV[i] = 0;

//Инициализация результата

int result = 0;

//Пока есть увеличивающий путь

while (bfs()){

for (int i = 1; i <= u.size(); i++) //Если свободная вершина существует и свободна то расширяем путь от нее

if (pairU[i] == 0 && dfs(i))

result++;

}

return result;

}

Назначение функции в том что она выполняет запуск алгоритм Хопкрофта Карпа и возвращает результат

Логика работы:

1. создаются три массива: pairU (хранит пару в сочетании для вершин из первой доли), pairV (хранит пару в сочетании для вершин из второй доли) и dist (хранит расстояние между левыми боковыми вершинами);
2. инициализируется результат (result) равным 0;
3. пока существует увеличивающий путь (проверяется с помощью bfs), перебираются все вершины первой доли (U) и если вершина свободна и увеличивающий путь может быть найден с этой вершины, увеличивается результат и увеличивающий путь расширяется;
4. в конце выполнения цикла возвращается найденный результат - максимальное паросочетание в графе.

*Функция zeroing*

Код:

void zeroing() {

delete[]G;

u.clear();

v.clear();

}

Данная функция очищает вектора U и V, удаляет выделенную под матрицу смежности память.

Логика работы:

1. очистить вектора U и V с помощью функции clear();
2. удалить матрицу смежности с помощью функции delete.

*Функция menu*

Код:

void menu() {

int n;

cout << "\n\t1. Запустить алгоритм\n\t2. Выход";

while (1) {

cout << "\nВыберите пункт: ";

cin >> n;

switch (n) {

case(1): {

creatG();

cout << "\n\n\tМаксимальное паросочитание равно: " << Hopkroft\_Karp\_alg() << "\n\n";

cout << "\tПары из паросочетания: \n";

for (int i = 1; i <= u.size(); i++) {

if (pairU[i] != 0) {

cout << "\t(" << u[i - 1] << ", " << pairU[i] + 1 << ")\n";

}

}

cout << "\n\tЗапустить данный алгоритм ещё раз? \n\t1 - Да\n\t0 - Нет\n\tВаш выбор: ";

cin >> n;

if (n == 1) { zeroing(); menu(); }

else exit(0);

}

case(2): {

exit(0);

}

default: cout << "\nТакого пункта нет, попробуйте снова";

}

}

}

Назначение:

Данная функция позволяет пользователю выбрать одну из опций данной программы.

Логика работы:

1. выводится меню с опциями "Запустить алгоритм" и "Выход";
2. пользователь вводит свой выбор (1 или 2) с клавиатуры;
3. с помощью оператора switch происходит проверка введенного значения:

* если пользователь выбрал 1, то вызываются функции creatG() и Hopkroft\_Karp\_alg() для выполнения алгоритма. Затем пользователю выводится результат выполнения алгоритма. Далее спрашивается, хочет ли пользователь выполнить алгоритм еще раз. Если да, вызывается функция zeroing() для сброса данных и снова отображается меню с помощью функции menu(). Если нет, программа завершается.
* если пользователь выбрал 2, программа завершается.
* если пользователь ввел другое значение, выводится сообщение об ошибке и пользователь должен выбрать снова.

1. пункт 2-3 повторяется, пока пользователь не выберет опцию "Выход".

*Логика работы main*

Код:

int main() {

setlocale(LC\_ALL, "");

menu();

return 0;

}

1. подключается русская локализация с помощью команды setlocale;
2. запускает функцию menu.
3. **Тестирование**

Среда разработки Microsoft Visual Studio 2022 предоставляет все средства, необходимые при разработке и отладке программы.

Программа будет протестирована на разном наборе данных, проверки функциональности вывода сообщений вывода данных в консоль, а также на проверку ошибок ввода и вывода ошибок.

В ходе тестирования будет проверена вся функциональная часть программы: ввод количества вершин, вывод данных в консоль, вывод матрицы смежности, вывод векторов, поиск максимального паросочетания, проверка ввода.

Таблица 1- Описание ожидаемого поведения программы

|  |  |  |  |
| --- | --- | --- | --- |
| **Описание теста** | **Предусловие** | **Последовательность** | **Ожидаемый результат** |
| Запуск программы | ОС загружена, открыт каталог с программой в которой лежит код exe файл | Запустить kursach.ехе | Вывод пользователю списка с функциями программы |
| Задание размера матрицы | Программа запущена, выведено меню пользователя | Ввести ,согласно меню, пункт запуска алгоритма, потом ввести число не меньше 2. | Вывод пользователю сообщения о создании графа |
| Тестирование на вывод матрицы небольших размеров | Программа запущена, матрица правильно создана и не сгенерирована. | Ввести ,согласно меню, пункт запуска алгоритма, потом ввести число не меньше 2 | Вывод матрицы на экран |

Продолжение таблицы 1

|  |  |  |  |
| --- | --- | --- | --- |
| Тестирование на вывод матрицы больших размеров | Программа запущена, матрица правильно создана и не сгенерирована. | Ввести ,согласно меню, пункт запуска алгоритма, потом ввести число не меньше 2 | Вывод матрицы на экран |
| Тестирование на работу алгоритма Хокрофта-Карпа. | Программа запущена, матрица  создана и сгенерирована. | Ввести ,согласно меню, пункт запуска алгоритма, потом ввести число не меньше 2 | Вывод результата работы алгоритма на экран |

Тесты:

1. Ввод количества вершин
   1. Вывод сообщения о выводе вершин



Рисунок 3- Вывод вершин

* 1. Ввод вершин



Рисунок 4- Ввод вершин

Таблица 2- Тест 1

|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Вывод сообщения о выводе вершин | В консоли выйдет сообщение | Верно |
| Ввод вершин | Будет введено количество вершин | Верно |

1. Вывод данных в консоль.
   1. Вывод сообщения о выборе:

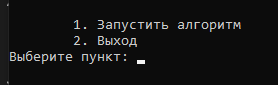


Рисунок 5 - Вывод данных

* 1. Выбор «Запустить алгоритм»



Рисунок 6 - Выведенные данные

* 1. Выбор «Выход»

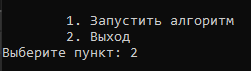


Рисунок 7- Выбор "Выход"

Таблица 3- Тест 2

|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Вывод сообщения | В консоли выведется сообщения о выборе | Верно |
| Выбор варианта «Запустить алгоритм» | В консоль выйдут данные:  Матрица смежности, вектора вершин, результат работы алгоритма | Верно |
| Выбор варианта «Выход» | В консоль ничего не выведется | Верно |

1. Вывод матрицы смежности
   1. Вывод матрицы в консоль на малых наборах данный:

****

Рисунок 8-Матрица смежности

* 1. Вывод матрицы в консоль на средних наборах данный:

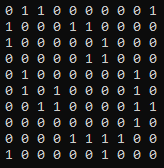
****

Рисунок 9 - Матрица смежности

* 1. Вывод матрицы в консоль на больших наборах данный:

****

Рисунок 10 - Матрица смежности

Таблица 4 - Тест 3

|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Вывод матрицы в консоль на малых наборах данный | В консоль выведется матрица смежности | Верно |
| Вывод матрицы в консоль на средних наборах данный | В консоль выведется матрица смежности | Верно |
| Вывод матрицы в консоль на больших наборах данный | В консоль выведется матрица смежности | Верно |

1. Вывод векторов U и V
   1. Вывод векторов на малых наборах данных

****

Рисунок 11 - вектора U и V

* 1. Вывод векторов на средних наборах данных

****

Рисунок 12 - вектора U и V

* 1. Вывод векторов на больших наборах данных

****

Рисунок 13- вектора U и V

Таблица 5 - Тест 4

|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Вывод векторов в консоль на малых наборах данный | В консоль выведется вектора U и V | Верно |
| Вывод векторов в консоль на средних наборах данный | В консоль выведется матрица U и V | Верно |
| Вывод векторов в консоль на больших наборах данный | В консоль выведется матрица U и V | Верно |

1. Поиск максимального паросочетания
   1. Вывод работы алгоритма на малых наборах данных(кол-во вершин – 4)

****

Рисунок 14 - максимальное паросочетание

* 1. Вывод работы алгоритма на средних наборах данных(кол-во вершин – 10)

****

Рисунок 15 - максимальное паросочетание

* 1. Вывод работы алгоритма на больших наборах данных(кол-во вершин – 25)



Рисунок 16 - максимальное паросочетание

Таблица 6 - Тест 5

|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Вывод графа на малых наборах данный | В консоль выведется максимальное паросочетание | Верно |
| Вывод графа на средних наборах данный | В консоль выведется максимальное паросочетание | Верно |
| Вывод графа на больших наборах данный | В консоль выведется максимальное паросочетание | Верно |

1. Проверка ввода
   1. Ввод букв

****

Рисунок 17- Ввод букв

* 1. Ввод отрицательных чисел



Рисунок 18 - Ввод отрицательного числа

* 1. Ввод нуля

****

Рисунок 19 - Ввод нуля

* 1. Ввод символов



Рисунок 20 - Ввод символов

* 1. Ввод пустой строки



Рисунок 21- Ввод пустой строки

* 1. Ввод дробного числа



Рисунок 22- Ввод дробного числа

Таблица 7 - некорректный ввод

|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Ввод букв | Выведется сообщение «Некорректно введенные данные, попробуйте снова!» | Верно |
| Ввод отрицательных чисел | Выведется сообщение «Некорректно введенные данные, попробуйте снова!» | Верно |
| Ввод нуля | Выведется сообщение «Некорректно введенные данные, попробуйте снова!» | Верно |
| Ввод символов | Выведется сообщение «Некорректно введенные данные, попробуйте снова!» | Верно |
| Ввод пустой строки | Выведется сообщение «Некорректно введенные данные, попробуйте снова!» | Верно |
| Ввод дробного числа | Выведется сообщение «Некорректно введенные данные, попробуйте снова!» | Верно |

Таблица 8 - Описание проведения программы при тестировании

|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Ввод количества вершин | Получение количества вершин графа | Верно |
| Вывод данных в консоль | В консоль выведутся данные о графе | Верно |
| Вывод матрицы смежности | В консоль выведется матрица смежности | Верно |
| Вывод векторов  U и V | В консоль выведется вектора U и V | Верно |
| Поиск максимального паросочетания | В консоль выведется результат поиска максимального паросочетания | Верно |
| Проверка ввода | При вводе некорректных данных выведется сообщение «Ошибка: Вы ввели некорректные данные!» | Верно |

# **6. Ручной расчет задачи**

Проведем проверку программы посредством ручных вычислений на примере графа с 5 вершинами.

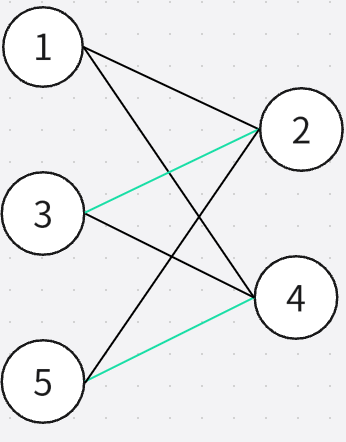
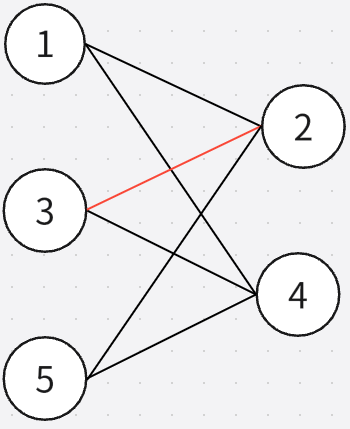
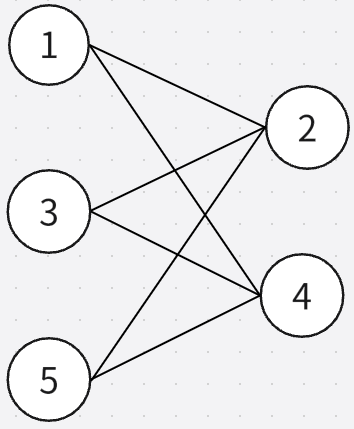
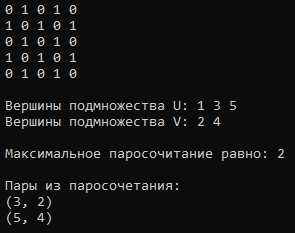
 

Рисунок 23 - работа на 5 вершинах

Вначале заносим вершины вектора U в очередь q. Вначале дистанция у всех вершин будет равна 0. Далее начинаем обход в глубину для всех вершин подмножества U. После первой итерации будет найдена пара 3 – 2. Далее будет найдена пара 5 – 4. Далее увеличивающего пути не будет, следовательно заканчиваем алгоритм.

# **Заключение**

Таким образом, в процессе создания данного проекта была разработана программа, реализующая алгоритм Хопкрофта Карпа в средах разработки Microsoft Visual Studio 2022.

При выполнение данной курсовой работы были получены навыки разработки алгоритмов и освоены приемы создания графов и вывода консольного отображения результатов. Углублены знания языков C++.

# **Список литературы**

1. Герберт Шилдт «Полный справочник по C++» - Вильямс, 2006
2. Уилсон Р. Введение в теорию графов. Пер. с анг. 1977. 208 с.
3. Т. А. Павловская “C/C++. Программирование на языке высокого уровня”
4. <https://ru.wikipedia.org/wiki/Алгоритм_Хопкрофта_—_Карпа>
5. <https://www.geeksforgeeks.org/hopcroft-karp-algorithm-for-maximum-matching-set-2-implementation>

# **Приложение А. Листинг программы**

**Main.cpp**

#include <iostream>

#include <vector>

#include <queue>

using namespace std;

int\*\* G, len, \*pairU, \*pairV, \*dist;

vector <int> u, v;

void printG() {

cout << "\tГраф G: \n\n";

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

cout << "\t";

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

cout << G[i][j] << " ";

}

cout << "\n";

}

cout << "\n\tВершины подмножества U: ";

for (int i = 0; i < u.size(); i++) cout << u[i] << " ";

cout << "\n\tВершины подмножества V: ";

for (int i = 0; i < v.size(); i++) cout << v[i] << " ";

}

//Функция проверки к какому подмножеству принадлежит вершина

bool find\_part(int s) {

for (int i = 0; i < v.size(); i++) {

if (v[i] == s) return true;

}

return false;

}

void creatG() {

//Ввод длины графа

while(1){

cout << "\n\tВведите длину графа: ";

cin >> len;

if (len <= 1) cout << "\n\tНекорректно введенные данные, попробуйте снова!\n";

else break;

}

srand(time(0));

//Распределение вершин по подмножествам

while(1){

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

if (rand() % 11 > 5) u.push\_back(i + 1);

else v.push\_back(i + 1);

}

if (v.size() < 1 || u.size() < 1 || u.size() < v.size()) { u.clear(); v.clear(); continue; }

else break;

}

//Инициализация и заполнение массива

G = new int\* [len];

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

G[i] = new int[len];

}

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

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

G[i][j] = 0;

//Проложение путей между вершинами

vector <int>::iterator j;

srand(time(0));

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

if (find\_part(i + 1)){

for (j = u.begin(); j != u.end(); ++j) {

int s = \*j;

if (s - 1 == i) continue;

if (rand() % 11 < 5) G[i][s - 1] = G[s - 1][i] = 1;

}

}

else {

for (j = v.begin(); j != v.end(); ++j) {

int s = \*j;

if (s - 1 == i) continue;

if (rand() % 11 < 5) G[i][s - 1] = G[s - 1][i] = 1;

}

}

}

printG();

}

bool bfs(){

queue <int> q;

for (int i = 1; i <= u.size(); i++) {

//Если свободная вершина то добавляем в очередь

if (pairU[i] == 0) {

dist[i] = 0;

q.push(i);

}

else dist[i] = INT\_MAX;

}

dist[0] = INT\_MAX;

while (!q.empty()) {

int s = q.front();

q.pop();

//Если вершина не равна нулю и может обеспечить более короткий путь к нулю

if (dist[s] < dist[0]) {

vector <int>::iterator i;

for (i = u.begin(); i != u.end(); ++i) {

int ss = \*i, k = 0;

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

if (G[ss - 1][j] == 1) {

if (dist[pairV[k]] == INT\_MAX) {

dist[pairV[k]] = dist[s] + 1;

q.push(pairV[k]);

}

k++;

}

}

}

}

}

return (dist[0] != INT\_MAX);

}

bool dfs(int t){

if (t != 0) {

vector <int>::iterator i;

for (i = u.begin(); i != u.end(); ++i) {

int ss = \*i, k = 0;

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

if (G[ss - 1][j] == 1) {

if(dist[pairV[k]] == dist[t - 1] + 1){

if (dfs(pairV[k]) == true) {

pairV[k] = t;

pairU[t] = j;

return true;

}

}

k++;

}

}

}

dist[t] = 0;

return false;

}

return true;

}

int Hopkroft\_Karp\_alg(){

//1 и 2 массивы хранят пару в сочетании, 3 - расстояние между левыми боковыми вершинами

pairU = new int[u.size() + 1], pairV = new int[v.size() + 1], dist = new int[u.size() + 1];

for (int i = 0; i <= u.size(); i++) pairU[i] = 0;

for (int i = 0; i <= v.size(); i++) pairV[i] = 0;

//Инициализация результата

int result = 0;

//Пока есть увеличивающий путь

while (bfs()){

for (int i = 1; i <= u.size(); i++) //Если свободная вершина существует и свободна то расширяем путь от нее

if (pairU[i] == 0 && dfs(i))

result++;

}

return result;

}

void zeroing() {

delete[]G;

u.clear();

v.clear();

}

void menu() {

int n;

cout << "\n\t1. Запустить алгоритм\n\t2. Выход";

while (1) {

cout << "\nВыберите пункт: ";

cin >> n;

switch (n) {

case(1): {

creatG();

cout << "\n\n\tМаксимальное паросочитание равно: " << Hopkroft\_Karp\_alg() << "\n\n";

cout << "\tЗапустить данный алгоритм ещё раз? \n\t1 - Да\n\t0 - Нет\n\tВаш выбор: ";

cin >> n;

if (n) { zeroing(); menu(); }

else exit(0);

}

case(2): {

exit(0);

}

default: cout << "\nТакого пункта нет, попробуйте снова";

}

}

}

int main() {

setlocale(LC\_ALL, "");

menu();

return 0;

}