Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное

Учреждение высшего профессионального образования

Кубанский государственный технологический университе

(ФБГОУ ВПО КубГТУ)

Кафедра Информационных систем и программирования

Факультет Компьютерных технологий и автоматизированных систем

**КУРСОВАЯ РАБОТА**

По дисциплине Дискретная математика

На тему Задачи и алгоритмы дискретной математики

Выполнил студент группы 12-КБ-ПР1

Семёнов Виталий Петрович\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Допущен к защите

Руководитель работы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись, дата, расшифровка подписи)

Защищён \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Оценка\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата)

Члены комиссии Симоненко. Е.А.

Волик. А.Г

Краснодар

2013

Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное

Учреждение высшего профессионального образования

Кубанский государственный технологический университет

(ФБГОУ ВПО КубГТУ)

Кафедра Информационных систем и программирования

Факультет Компьютерных технологий и автоматизированных систем

УТВЕРЖДАЮ

Зав. Кафедрой ИСП

профессор,

д.т.н Видовский Л.А.

(дата, подпись, расшифровка подписи)

**ЗАДАНИЕ**

на курсовую работу

Студенту Семёнову Виталию Петровичу группы 12-КБ-ПР1

факультета Компьютерных технологий и автоматизированных систем

направления 231000 – Программная инженерия

Тема работы Задачи и алгоритмы дискретной математики

Содержание задания: Изучить темы «Потоки в сетях» и «Компоненты связности в ориентированных графах», провести исследования алгоритмов работы с этими объектами, составить программы, которые демонстрируют указанные алгоритмы, провести тестирование программ.

Объём курсовой работы:

а) пояснительная записка 18 стр.;

б) Программы: 2 шт.;

Рекомендуемая литература Скиена С. Алгоритмы. Руководство по разработке, Седжвик Р. Алгоритмы на C++.

Срок выполнения : с 2 сентября 2013г. по 23 декабря 2013г.

Срок защиты: с 2 декабря 2013г. по 23 декабря 2013г.

Дата выдачи задания: с 2 сентября 2013г. по 10 сентября 2013г.

Дата сдачи работы на кафедру: с 2 декабря 2013г. по 23 декабря 2013г.

Руководитель работы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

Задание принял студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

РЕФЕРАТ

ПОТОКИ, АЛГОРИТМ ГОЛЬДБЕРГА-ТАРЬЯНА, C++, ОРИЕНТИРОВАННЫЕ ГРАФЫ, VISUAL STUDIO 2010, СВЯЗНОСТЬ.

Стр. 18, табл. 0, рис. 4, библ. 3.

В данной курсовой работе были рассмотрены вопросы, касающиеся двух тем: «Потоки в сетях: алгоритм Гольдберга - Тарьяна», «Алгоритм поиска компонент связности ориентированного графа. Проверка на сильную связность».

Основными моментами проведённого исследования были практическое исследование алгоритмов.

Следовательно, данная работа сделала возможным оценить каждый алгоритм с практической точки зрения.

Содержание.

Введение………………………………………………………………………4

Глава 1 Потоки в сетях: Алгоритм Гольдберга-Тарьяна……………….... 6

1.1 Постановка задачи………………………………………………….……6

1.2 Решение задачи………………………………………………………...….6

1.3 Тестирование программы…………………………………………….….7

Глава 2 Компоненты связности…………………………………………..…9

2.1 Постановка задачи………………………………………………………9

2.2 Решение задачи…………………………………………………………..9

2.3 Тестирование программы……………………………………………...10

Заключение…………………………………………………………………. 12

Список использованной литературы…………………………..………….13

Приложение A. Код программы нахождения максимального потока...14

Приложение B. Код программы нахождения компонент связности……17

**Введение**

В ходе выполнения курсовой работы будут рассмотрены две задачи из дискретной математики и написаны программы, реализующие алгоритмы решения этих задач: Нахождение максимального потока с помощью алгоритма Гольдберга – Тарьяна и нахождение компонент сильной связности в ориентированном графе.

Данные задачи относятся к числу основных задач курса дискретной математики. В теории оптимизации и теории графов, задача о максимальном потоке заключается в нахождении такого потока по транспортной сети, что сумма потоков из истока, или, что то же самое, сумма потоков в сток максимальна. Эта задача имеет большую практическую ценность. С её помощью, например, можно определить максимальное количество жидкости, которое может пройти через трубопровод.

Задача нахождения компонент сильной связности так же является одной из центральных в теории графов. Очень часто, именно с неё начинают исследование ориентированного графа.

**Глава 1 Потоки в сетях. Алогитм Гольдберга – Тарьяна.**

**1.1. Постановка задачи.**

Составить программу на языке программирования C++, которая по алгоритму Гольдберга – Тарьяна методом проталкивания предпотока искала бы максимальный поток в сетях.

Входные данные:

В первой строке входного файла содержится одно число: d (2 ≤ d). Это количество вершин в ориентированном графе. Далее следует матрица смежности размером d \* d, где пересечение (i,j) отличное от нуля означает, что вершину i c вершиной j соединяет ребро, чья пропускная способность равна значению a [i,j] в матрице. При этом, первый столбец и первая строка всегда нулевые, т.к первая вершина является стоком, а вторая истоком. То есть, из последней вершины не выходит не одно ребро, а в первую ничего не входит.

Выходные данные:

В единственную строку выходного файла выведите одно число - размер максимального потока из истока в вершину сток.

**1.2 Решение задачи**

Для нахождения максимального потока в сети используется несколько алгоритмов. Метод выталкивания превосходящего потока был разработан Гольдбергом и Тарьяном в 1986 г. на основе ранее открытых алгоритмов. Алгоритм получил широкое распространение, благодаря своей простоте, гибкости и эффективности. Временная сложность алгоритма составляет O(N**4)**, или, точнее  O (N2M).

Общая схема алгоритма такова. На каждом шаге будем рассматривать некоторый предпоток - т.е. функцию, которая по свойствам напоминает поток, но не обязательно удовлетворяет закону сохранения потока. На каждом шаге будем пытаться применить какую-либо из двух операций: проталкивание потока или поднятие вершины. Если на каком-то шаге станет невозможно применить какую-либо из двух операций, то мы нашли требуемый поток.

Для каждой вершины определена её высота Hu, причём HS = N, HT = 0, и для любого остаточного ребра (u, v) имеем Hu <= Hv + 1.

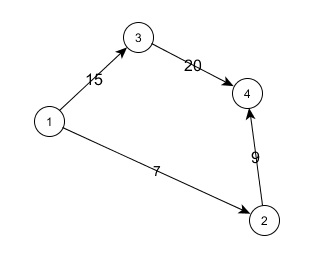
Для каждой вершины (кроме S) можно определить её избыток: Eu = FV, u. Вершина с положительным избытком называется переполненной.

Операция проталкивания Push (u, v) применима, если вершина u переполнена, остаточная пропускная способность Cfu, v > 0 и Hu = Hv + 1. Операция проталкивания заключается в максимальном увеличении потока из u в v, ограниченном избытком Eu и остаточной пропускной способностью Cfu, v.

Операция поднятия Lift (u) поднимает переполненную вершину u на максимально допустимую высоту. Т.е. Hu = 1 + min { Hv }, где (u, v) - остаточное ребро.

Осталось только рассмотреть инициализацию потока. Нужно инициализировать только следующие значения: FS, v = CS, v, Fu, S = - Cu, S, остальные значения положить равными нулю.

**1.3 Тестирование программы**

****

**Рисунок 1 – Граф 1**

Тест 1:

Input

4

0 7 15 0

0 0 0 9

0 0 0 20

0 0 0 0

Output

22

На рисунке мы видим, что через путь 1→2→4 можно пропустить **15** единиц, а через путь 1→3→4 можно пропустить **7** единиц. Суммируя, получаем общую пропускную способность 22.

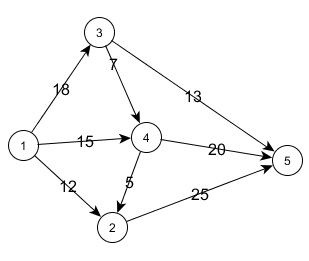


Рисунок 2 – Граф 2

Тест 2

Input

5

0 12 18 15 0

0 0 0 0 25

0 0 0 7 13

0 5 0 0 20

0 0 0 0 0

Output

45

Посчитаем на графе 2 пропускные способности всех путей и просуммируем. Путь 1→3→5 «донесёт» до стока **13** единиц. 5 единиц спуститься в вершину 3. Путь 1→4→5 пропустит **20** единиц. Путь 1→2→5 пропустит ещё **12** единиц. В сумме пропускная способность сети равна 45.

**Глава 2Алгоритм поиска компонент связности ориентированного графа. Проверка на сильную связность.**

**2.1 Постановка задачи**

Составить программу на языке программирования C++, которая будет искать в ориентированном графе компоненты связи, и на этой основе определять, является ли граф сильно связным.

Входные данные:

В первой строке входного файла содержится два числа: n и m (2 ≤ n ≤ 10, 1 ≤ m ≤ n·(n-1)). Это количество вершин и рёбер в графах, которые нужно проверить на сильную связность. Далее следуют список смежности графа, в котором описываются ребра графа. Описание ребра состоит из двух чисел: a, b (0 ≤ a, b ≤ n, a ≠ b). Эти числа означают, между вершинам a и b есть ребро, которое выходит из вершины a и приходит в вершину b.

Выводные данные.

В выходной файл необходимо вывести найденные компоненты связности. Где одна компонента связности выводится в формате номеров вершин a, b, … , n , где – n число вершин графа. Если выведется ровно одна компонента связности, содержащая все вершины графа – значит граф является сильно связным

**2.2 Решение задачи.**

[Орграф](http://ru.wikipedia.org/wiki/%D0%9E%D1%80%D0%B3%D1%80%D0%B0%D1%84) называется сильно связным ([англ.](http://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) strongly connected), если любые две его вершины сильно связаны. Две вершины a и b любого графа сильно связаны, если существует ориентированный путь из a в b и ориентированный путь из t в s. Компонентами сильной связности орграфа называются его максимальные по включению сильно связные подграфы. Орграф, не принадлежащий к классу сильно связных графов, содержит некоторый набор сильно связных компонент, и некоторый набор ориентированных ребер, идущих от одной компоненты к другой. Любая вершина орграфа сильно связана сама с собой.

Поиск компонент связности осуществляем следующим образом: из первой вершины (0) начинаем выполнять обход графа в глубину. Все вершины, которые мы при этом посещаем – становятся помеченными. Затем начинаем вновь обход в глубину с непомеченных вершин. Если весь граф будет пройден за один раз – значит он является сильно связным. Сильной компонентой связности для графа будет являться его связный подграф. Итоговая асимптотика составит O(n + m).

**2.3 Тестирование программы**

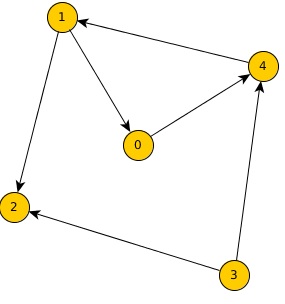
****

Рисунок 3 – Граф 3

Тест 1

Input

5 6

0 4

1 0

1 2

3 2

3 4

4 1

Output

0, 1, 4

2

3

Совершив первый обход в глубину, мы пройдем по вершинам 0, 1 и 4. Это и будет наша первая компонента сильной связности. Оставшиеся компоненты – это одиночные вершины 2 и 3.

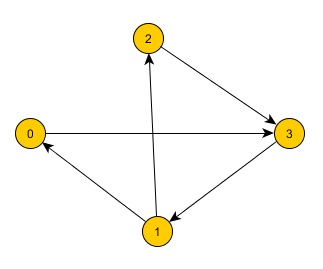


Рисунок 4 – Граф 4

Тест 4

Input

4 5

0 3

1 0

2 3

1 2

3 1

Output

0, 1, 3, 2

Выполнив обход в глубину 1 раз, мы посетим все вершины. Кроме того мы видим, что из каждой вершины нашего графа мы можем попасть в любую другую. Следовательно – граф сильно связный и имеет 1 компоненту связности.

**Заключение**

В ходе выполнения курсовой работы были рассмотрены две задачи из курса дискретной математики: нахождение максимального потока с помощью алгоритма Голдберга – Тарьяна и нахождение компонент связности в ориентированном графе. Была изучена соответствующая литература. Алгоритмы решения задач были реализованы на языке программирования C++.

**Список использованной литературы.**

1. Скиена С. Алгоритмы. Руководство по разработке. – 2-е изд.: пер. с англ. – СПб.: БХВ-Петербург, 2011. – 720 с.
2. Седжвик Р. Алгоритмы на C++. – Пер. с англ. – М.Вильямс, 2011. – 1056 с.

3. Свободная энциклопедия [Электронный ресурс] – Режим доступа: [http://ru.wikipedia.org/](http://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%BE%D0%BD%D0%B5%D0%BD%D1%82%D0%B0_%D1%81%D0%B8%D0%BB%D1%8C%D0%BD%D0%BE%D0%B9_%D1%81%D0%B2%D1%8F%D0%B7%D0%BD%D0%BE%D1%81%D1%82%D0%B8_%D0%B2_%D0%BE%D1%80%D0%B3%D1%80%D0%B0%D1%84%D0%B5)wiki/Компонента \_сильной\_связности\_в\_орграфе

**Приложение A. Код программы нахождения максимального потока.**

#include <iostream>

#include <vector>

using namespace std;

const int inf = 1000\*1000\*1000;

typedef vector<int> graf\_line;

typedef vector<graf\_line> graf;

typedef vector<int> vint;

typedef vector<vint> vvint;

void push (int u, int v, vvint & f, vint & e, const vvint & c)

{

int d = min (e[u], c[u][v] - f[u][v]);

f[u][v] += d;

f[v][u] = - f[u][v];

e[u] -= d;

e[v] += d;

}

void lift (int u, vint & h, const vvint & f, const vvint & c)

{

int d = inf;

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

if (c[u][i]-f[u][i] > 0)

d = min (d, h[i]);

if (d == inf)

return;

h[u] = d + 1;

}

int main()

{

int n;

cin >> n;

vvint c (n, vint(n));

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

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

cin >> c[i][j];

// исток - вершина 0, сток - вершина n-1

vvint f (n, vint(n));

for (int i=1; i<n; i++)

{

f[0][i] = c[0][i];

f[i][0] = -c[0][i];

}

vint h (n);

h[0] = n;

vint e (n);

for (int i=1; i<n; i++)

e[i] = f[0][i];

for ( ; ; )

{

int i;

for (i=1; i<n-1; i++)

if (e[i] > 0)

break;

if (i == n-1)

break;

int j;

for (j=0; j<n; j++)

if (c[i][j]-f[i][j] > 0 && h[i]==h[j]+1)

break;

if (j < n)

push (i, j, f, e, c);

else

lift (i, h, f, c);

}

int flow = 0;

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

if (c[0][i])

flow += f[0][i];

cout << max(flow,0);

}

**Приложение Б. .Код программы нахождения компонент связности.**

#include <iostream>

#include <vector>

using namespace std;

/\* Вывод будет в следующем виде: каждая строка будет описывать одну компоненту связности.

В одной строке будут перечислены через запяту вершины, которые образуют одну компоненту сильной связности.

\*/

vector < vector<int> > g, gr; // сам граф и транспонированный граф

vector<bool> used; // метки

vector<int> order, component; // компоненты связности и

void dfs1 (int v) {

used[v] = true;

for (size\_t i=0; i<g[v].size(); ++i)

if (!used[ g[v][i] ])

dfs1 (g[v][i]);

order.push\_back (v);

}

void dfs2 (int v) {

used[v] = true;

component.push\_back (v);

for (size\_t i=0; i<gr[v].size(); ++i)

if (!used[ gr[v][i] ])

dfs2 (gr[v][i]);

}

int main() {

int n, m; // количество вершин и количество ребер соответсвенно

cin >> n >> m;

g.assign(n, vector<int>()); // выделяем память для графов

gr.assign(n, vector<int>());

int u, v;

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

cin >> u >> v; // описание ребра. вершина выхода и вершина входа ребра соответсвенно

g[u].push\_back (v);

gr[v].push\_back (u);

}

used.assign (n, false); // метка. здесь отмечается, если мы достигли уже како-то вершины

for (int i=0; i<n; ++i) // из всех вершин, где еще не были, запускаем DFS

if (!used[i])

dfs1 (i);

used.assign (n, false); // обнуление меток

for (int i=0; i<n; ++i) { // еще раз запуск DFS, но уже учитывая времена входа и выхода сильных связностей.

int v = order[n-1-i];

if (!used[v]) {

dfs2 (v);

for (int i = 0; i < component.size() - 1; ++i) // вывод компоненты связности

cout << component[i] << ", ";

cout << component[component.size() - 1] << endl;

component.clear();

}

}

system("pause");

}