

**Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Московский государственный технический университет имени Н.Э. Баумана**

**(национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)**

ФАКУЛЬТЕТ *Робототехники и комплексной автоматизации*

КАФЕДРА *Системы автоматизированного проектирования (РК-6)*

# ОТЧЕТ О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ

Студент Журавлев Н.В.

Группа РК6-42Б

Тип задания Лабораторная работа №3

Студент

**Журавлев Н.В.**

*подпись, дата фамилия, и.о.*

Преподаватель

**Волосатова Т.М.**

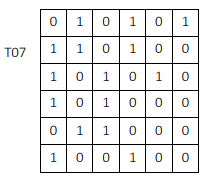
*подпись, дата фамилия, и.о.*

*Москва, 2021*

**Вариант 5**

**Задание**

В заданной [0,1] – матрице М, размеров 6х6, требуется выбрать по одной 1 последовательно из каждой ее строки сверху вниз, которые должны иметь попарно различные индексы своих столбцов.



**Теоретическая часть**

Алгоритм Холла реализует итерационный процесс выборов и перевыборов представителей подмножеств заданного семейства в порядке роста их индексов, а номер каждого представителя в искомой СРП идентифицирует подмножество, откуда он выбран. На каждой итерации i от 1 до m представителем очередного подмножества 𝑆𝑖 выбирается его минимальный (по номеру) свободный элемент 𝐸𝑗, который не входит в текущую частичную трансверсаль 𝑇𝑖−1 из представителей ранее рассмотренных подмножеств:

𝑆𝑖=𝑇𝑖(𝐸𝑗=min{𝑆𝑖−𝑇𝑖−1}).

Прямой выбор представителя невозможен, если все элементы подмножества 𝑆𝑖 уже использованы в качестве представителей. Тогда из них образуют список 𝐿∅=𝑆𝑖, который должен просматриваться и дополняться. В ходе просмотра списка для каждого элемента 𝐸𝑝 по его месту в текущей трансверсали 𝑇𝑖−1 нужно определить подмножество 𝑆𝑉, представителем которого он является. Все элементы этого подмножества, которых нет в списке, нужно добавить в конец списка, расширяя его состав:

𝐿𝑝=⟨… 𝐸𝑖 𝐸𝑝∈𝑇𝑖−1…⟩+{𝑇𝑖−1(𝐸𝑝)=𝑆𝑣−𝐿𝑖 }.

Просмотр и дополнение списка завершается, когда достигнут элемент, который не входит в СРП. Пусть первый такой элемент 𝐸𝐾 добавлен из подмножества 𝑆𝑟, представителем которого является, например элемент 𝐸𝑞 (стоящий на месте r в частичной трансверсали 𝑇𝑖−1):

𝐿𝑘=⟨… 𝐸𝑘∉𝑇𝑖−1…⟩|𝐸𝑘∈𝐿∅∪{𝑆𝑟=𝑇𝑖−1(𝐸𝑞)}|.

Этот свободный элемент 𝐸𝑘 должен стать теперь представителем подмножества 𝑆𝑟, откуда он был добавлен в список, вместо его прежнего представителя 𝐸𝑞 в СРП. Таким образом происходит замена представителя в позиции r текущей трансверсали:

𝑇𝑖−1′=𝑇𝑖−1+{𝐸𝑘}−{𝐸𝑞}| 𝑆𝑟=𝑇𝑖−1′(𝐸𝑘).

Прежний представитель 𝐸𝑞 теперь свободен и должен быть выбран представителем того подмножества, откуда он попал в список, освобождая в свою очередь, его представителя. Такой рекурсивный процесс замены представителей завершается, когда будет освобождён любой элемент 𝐸𝑗 из начального списка 𝐿∅ текущей итерации i. Этот элемент 𝐸𝑗 выбирается представителем подмножества 𝑆𝑖=𝐿∅ и добавляется в конец текущей трансверсали (на позицию i):

𝑇𝑖=𝑇𝑖−1+{𝐸𝑗∈𝑆𝑖=𝐿∅}.

Аналогичные итерации нужно проделать для выбора представителей из всех последующих подмножеств заданного семейства до получения полной трансверсали 𝑇=𝑇𝑚, где все m её элементов 𝐸𝑗 будут перечислены по построению в порядке номеров подмножеств 𝑆𝑖, которые они представляют:

𝑇={𝐸𝑗|𝑆𝑖=𝑇(𝐸𝑗),𝑖=1,…,𝑚}

**Алгоритм**

Семейства:

S1 = {2, 4, 6}

S2 = {1, 2, 4}

S3 = {1, 3, 5}

S4 = {1, 3}

S5 = {2, 3}

S6 = {1, 4}

Считаем T0:

T0 = {2, 1, 3}

Получается 3 итерации:

1. L0 = {1, 3}  
   L1 = {1, 3, 2, 4}  
   L3 = {1, 3, 2, 4, 5, 6}  
   T= {4, 2, 3, 1}
2. L0 = {2, 3}  
   L1 = {2, 3, 1, 4}  
   L2 = {2, 3, 1, 4, 5}  
   L3 = {2, 3, 1, 4, 5}  
   L4 = {2, 3, 1, 4, 5, 6}  
   T = {4, 2, 5, 1, 3}
3. L0 = {1, 4}  
   L1 = {1, 4, 3}  
   L2 = {1, 4, 3, 2, 6}  
   L3 = {1, 4, 3, 2, 6}  
   L4 = {1, 4, 3, 2, 6}

Итоговое T = {6, 2, 5, 1, 3, 4}

**Код программы**

#include <iostream>

#include <fstream>

#include <vector>

#include <map>

#include <new>

#define FILENAME "/home/a2000/CLionProjects/cppproject/matrix.txt"

using namespace std;

int \*\* fill\_matrix(int rows, int cols) {

ifstream fin(FILENAME);

if (!fin.is\_open()) {

cout << "Error file name" << endl;

exit(1);

}

int \*\* matrix = new int \* [rows];

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

matrix[i] = new int[cols];

}

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

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

fin >> matrix[i][j];

}

}

fin.close();

return matrix;

}

void output\_matrix(int \*\* matrix, int rows, int cols) {

cout << "Матрица:" << endl;

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

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

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

}

cout << endl;

}

}

void delete\_matrix(int \*\* matrix, int rows) {

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

delete[] matrix[i];

}

}

vector<vector<int>> create\_family(int \*\* matrix, int rows, int cols) {

vector<vector<int>> S;

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

vector<int> family;

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

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

family.push\_back(j + 1);

}

}

S.push\_back(family);

}

return S;

}

bool find\_cell(int x, vector <int> v) {

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

if (x == v[i]) {

return true;

}

}

return false;

}

void output\_vector(vector < int > v) {

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

cout << v[i] << ' ';

}

cout << endl;

}

vector<int> create\_t0(vector < vector < int > > S) {

vector <int> T0;

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

bool set\_new = true;

bool can\_have = false;

for (int j = 0; j < S[i].size() && set\_new; j++) {

if (!find\_cell(S[i][j], T0)) {

T0.push\_back(S[i][j]);

set\_new = false;

can\_have = true;

} else {

can\_have = false;

}

}

if (!can\_have) {

return T0;

}

}

return T0;

}

vector<int> vectors\_st(vector <int> L, vector <int> S) {

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

if (!find\_cell(S[i], L)) {

L.push\_back(S[i]);

}

}

return L;

}

int find\_index(int x, vector < int > v) {

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

if (x == v[i]) {

return i;

}

}

return -1;

}

int index\_fam(map <int, vector<int>> &added\_families, int x, int after) {

for (auto it = added\_families.begin(); it != added\_families.end(); it++) {

if (it -> first > after) {

if (find\_cell(x, added\_families[it -> first])) {

return it -> first;

}

}

}

return -1;

}

vector<int> find\_L\_T(vector <int> &T, vector <vector<int>> &S) {

vector <int> L;

vector <int> T\_new = T;

int next\_agent = T.size();

L = S[next\_agent];

cout << endl << "L0 = ";

output\_vector(L);

int i = 0;

int j = find\_index(L[i], T);

map<int,vector<int>> added\_families;

while (j != -1) {

added\_families[j] = S[j];

L = vectors\_st(L, S[j]);

cout << "L" << i + 1 << " = ";

output\_vector(L);

i++;

j = find\_index(L[i], T);

}

int new\_agent = L[i];

int replace\_index = -1;

while (!find\_cell(new\_agent, S[next\_agent])) {

replace\_index = index\_fam(added\_families, new\_agent, replace\_index);

swap(new\_agent, T\_new[replace\_index]);

}

T\_new.push\_back(new\_agent);

cout << endl << "T = ";

output\_vector(T\_new);

return T\_new;

}

int main() {

int rows = 6;

int cols = 6;

int \*\* matrix = fill\_matrix(rows, cols);

output\_matrix(matrix, rows, cols);

vector<vector<int>> S = create\_family(matrix, rows, cols);

cout << endl << "Семейства:" << endl;

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

cout << "S" << i + 1 << " = ";

output\_vector(S[i]);

}

vector<int> T = create\_t0(S);

cout << endl << "T0 = ";

output\_vector(T);

for (int i = 0; T.size() != rows; i++) {

cout << endl << "Итерация " << i + 1 << endl;

T = find\_L\_T(T, S);

}

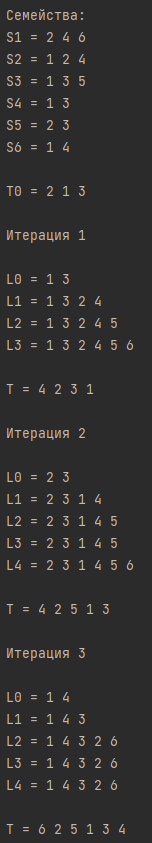
delete\_matrix(matrix, rows);

return 0;

}

**Результат работы программы**

Вывод:



**Литература**

1. Курс лекций «Методы комбинаторных вычислений» - Волосатова Т.М., Родионов С.В.