**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ**

**РОССИЙСКОЙ ФЕДЕРАЦИИ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ**

**ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»**

**(БГТУ им. В. Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и

автоматизированных систем

Лабораторная работа № 4.3

Дисциплина: Дискретная Математика

тема: «Связность»

Выполнил: ст. группы ВТ-22

Богатырёв Валентин В.

Белгород 2018

**Цель**: изучить алгоритм Краскала построения покрывающего леса, научиться использовать его при решении различных задач.

**Задания**

1. Реализовать алгоритм Краскала построения покрывающего леса.

2. Используя алгоритм Краскала, разработать и реализовать алго-

ритм решения задачи (см. варианты заданий).

3. Подобрать тестовые данные. Результат представить в виде диа-

граммы графа.

**Выполнение**

Найти все k-элементные множества ребер, исключение которых из связного графа разбивает его на две связные компоненты.

1. Реализовать алгоритм Краскала построения покрывающего леса.

2. Используя алгоритм Краскала, разработать и реализовать алгоритм решения задачи (см. варианты заданий).

#include <stdio.h>

#include <stdlib.h>

int\*\* InitMem(int N)

{

int\*\* G, i;

G = (int\*\*)malloc(N\*sizeof(int\*));

for(i = 0; i < N; i++)

G[i] = (int\*)malloc(N\*sizeof(int));

return G;

}

void FreeMem(int\*\* G, int M)

{

int i;

for(i = 0; i < M; i++)

free(G[i]);

free(G);

}

void WriteGraph(int\*\* G, int M, int N)

{

int i, j;

for(i = 0; i < M; i++)

{

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

printf("%i ", G[i][j]);

printf("\n");

}

printf("\n");

}

void ReadGraph(FILE \*f, int\*\* G, int N)

{

int i, j;

for(i = 0; i < N; i++)

{

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

fscanf(f, "%i", &G[i][j]);

}

printf("\n");

}

void WriteArr(int\* A, int N)

{

int i;

for(i = 0; i < N; i++)

printf("%i ", A[i]+1);

printf("\n");

}

//Возвращает количество букетов/деревьев в остовном лесе

//В B записаны букеты

//N - количество вершин

int Kruskal(int\*\* G, int N)

{

int count = N, i, j, k;

int\* B = (int\*)malloc(N\*sizeof(int));

//Изначально каждая вершина представляет собой букет

for(i = 0; i < N; i++)

{

B[i] = i;

}

for(i = 0; i < N; i++)

{

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

{

//Если две вершины связаны, и принадлежат разным букетам,

if(G[i][j] && B[i] != B[j])

{

//объединим букеты

for(k = 0; k < N; k++)

{

if(B[k] == B[j]){

B[k] = B[i];

}

}

count--; //количество букетов уменьшается

}

}

}

return count;

}

void GenComb(int\*\* G, int\*\* E, int\* C, int N, int M, int k, int i, int b)

// G - граф, E - матрица рёбер, N - размер матрицы смежности, M - кол-во ребёр, i - заполняемая позиция

// C - формируемое сочтание, k -кол-во элементов в сочетании, b - минимальный элемент который можно поставить на i место

{

int x;

for(x = b; x <= (M-k+i); x++)

{

C[i] = x; //ставим элемент на iтое место

G[E[0][x]][E[1][x]] = 0; //исключаем из букета (графа)

G[E[1][x]][E[0][x]] = 0; //вершины связанные соответствующими ребрами

if(i == (k-1))//т.к. первый раз передаем i=0

{

if(Kruskal(G, N) == 2) //разбиение равно двум

WriteArr(C, k);

}

else

GenComb(G, E, C, N, M,k,i+1,x+1); //иначе генерируем следующую комбинацию

G[E[0][x]][E[1][x]] = 1; //выбрано ребро и включено в лес

G[E[1][x]][E[0][x]] = 1; // сформирован букет с кольцевыми вершинами

}

}

int\*\* GenMatrReb(int\*\* G, int N, int\* M) //создание матрицы рёбер

{

int\*\* E, i, j; //е - матрица ребёр

E = (int\*\*)malloc(2\*sizeof(int\*)); //2 строки

for(i = 0; i < 2; i++)

E[i] = (int\*)malloc((N\*N-N)/2\*sizeof(int)); //при n = 5 будет 10 N\*N-N -максимум ребер, делим на два т.к. симметрия

int count = 0; //кол-во ребер

for(i = 0; i < N; i++)

{

for(j = i; j < N; j++)

if(G[i][j]) //если это вершина записываем

{

E[0][count] = i;

E[1][count] = j;

printf("(%d, %d): %d\n", i+1, j+1, count+1);

count++;

}

}

for(i = 0; i < 2; i++)

E[i] = (int\*)realloc(E[i],count\*sizeof(int)); //усекание по кол-ву ребер

(\*M) = count; //кол-во вершин

return E;

}

int main()

{

FILE \*f;

int\*\* G;

int\*\* E;

int N, M;

int\* C;

int k;

f = fopen("D:/lab.txt","r");

fscanf(f,"%i",&N);

G = InitMem(N);

ReadGraph(f,G,N);

WriteGraph(G,N,N);

fclose(f);

printf("Enter k: ");

scanf("%i",&k); //ввод кол-ва элементов множества ребёр

E = GenMatrReb(G, N, &M); //создание матрицы ребёр

printf("\n");

C = (int\*)malloc(k\*sizeof(int)); //массив для генерации сочетаний из k элементов

GenComb(G,E,C,N,M,k,0,0);

free(C);

FreeMem(G,N);

FreeMem(E,2);

return 0;

}

3. Подобрать тестовые данные. Результат представить в виде диа-

граммы графа.

4

3

2

1

