**МИНОБРНАУКИ РОССИИ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ**

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

**(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

дисциплина: Дискретная математика

Лабораторная работа № 4.3

тема: «Связность»

|  |  |
| --- | --- |
|  | Выполнил: ст. группы ВТ-22  Макаров Даниил Сергеевич  Проверил: Рязанов Ю.Д. |

Белгород

2018г.

**Цель занятия:** 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 алгоритм решения задачи.
3. Подобрать тестовые данные. Результат представить в виде диаграммы графа.

**Задание варианта 7**

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

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

typedef struct S\_edge{

int x;

int y;

}edge;

//Возвращает количество букетов/деревьев в остовном лесе

//В B записаны букеты

//N - количество вершин

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

int count = N, i, j, k;

//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++){

int t = B[j];

if(B[k] == t){

B[k] = B[i];

}

}

count--; //количество букетов уменьшается

}

}

}

return count;

}

//Подсчитывает количество ребер по матрице и записывает их в массиив edge\_arr

edge\* EdgesCount(int \*\*G, int N,int \*count\_of\_edge){

int i = 0;

edge \*edge\_arr=(edge\*) malloc(sizeof(edge)\*100);

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

for (int j = i+1; j<N; j++){

if (G[i][j]){

i++;

edge\_arr[i].x=i;

edge\_arr[i].y=j;

}

}

}

\*count\_of\_edge=i;

return edge\_arr;

}

//выводит массив граней

void print\_edge\_arr(edge \*arr, int size){

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

printf("(%d,%d),",arr[i].x,arr[i].y);

}

printf("\b \n");

}

//Копирует массив в другой

void Copy1(const int \*A, int \*B, int N){

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

B[i] = A[i];

}

}

//Копирует двумерный массив в другой

void Copy2(int \*\*A, int \*\*B, int N){

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

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

B[i][j] = A[i][j];

}

}

}

//Поэлементно вычитает из A B

void Minus(int \*\*A, int \*\*B,int\*\* result, int N){

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

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

result[i][j]=A[i][j] - B[i][j];

}

}

}

void clear\_matr(int \*\*A, int N){

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

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

A[i][j] = 0;

}

}

}

//Выводит ребра графа G

void PrintEdges(int \*\*G, int N){

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

for (int j = i+1; j<N; j++){

if (G[i][j]){

printf("%i <-> %i\n", i+1, j+1);

}

}

}

}

//Выводит граф в формате DOT

void PrintDot(FILE \*f, int \*\*G, int N){

fprintf(f, "graph auto{\n");

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

fprintf(f, "%i;", i+1);

for (int j = i+1; j<N; j++){

if (G[i][j]){

fprintf(f, "%i--%i;\n", i+1, j+1);

}

}

}

fprintf(f, "}\n");

}

int \*\*GetMem(int N){

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

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

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

}

return r;

}

void FreeMem(int \*\*G, int N){

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

free(G[i]);

}

free(G);

}

void clear\_edge\_arr(edge\* arr){

free(arr);

arr=(edge\*)malloc(sizeof(edge)\*100);

}

//Выводит матрицу графа G

void PrintAdjG(int \*\*G, int N){

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

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

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

}

printf("\n");

}

printf("\n");

}

//Вводит матрицу графа G

void ReadAdjG(FILE \*f, int \*\*G, int N){

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

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

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

}

}

}

void task(int \*\*G, int N){

int \*B; //массив для Краскала

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

int \*\*temp\_matr1=GetMem(N);

int \*\*temp\_matr2=GetMem(N);

int count\_of\_edge;

edge \*sub\_edge\_arr=(edge\*)malloc(sizeof(edge)\*100);

int count\_of\_edge\_in\_sub =0;

int k=0;

edge \*edge\_arr=EdgesCount(G,N,&count\_of\_edge);

unsigned count\_of\_sub\_arr = pow(2, count\_of\_edge);

for (int i = 0; i < count\_of\_sub\_arr;i++){

clear\_matr(temp\_matr1,N);

clear\_edge\_arr(sub\_edge\_arr);

k=0;

for (int j = 0;j<count\_of\_edge;j++){

if ( i & (1 << j) ){ //--если j-й бит установлен

temp\_matr1[edge\_arr[j].x][edge\_arr[j].y]=1;

temp\_matr1[edge\_arr[j].y][edge\_arr[j].x]=1;

sub\_edge\_arr[k]=edge\_arr[j];

k++;

}

}

Minus(G,temp\_matr1,temp\_matr2,N);

if (Kruskal(temp\_matr2, B, N) == 2){

print\_edge\_arr(sub\_edge\_arr,k-1);

}

}

}

int main(){

FILE \*f;

FILE \*g\_out;

int\*\* G;

int\*\* E;

int N;

f = fopen("graph.txt","r");

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

printf("%i\n", N);

G = GetMem(N);

ReadAdjG(f,G,N);

PrintAdjG(G,N);

g\_out = fopen("gin.gv", "w");

PrintDot(g\_out, G, N);

fclose(g\_out);

fclose(f);

task(G,N);

FreeMem(G, N);

return 0;

}