МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им. Н. Э. Баумана

КАФЕДРА ПРОЕКТИРОВАНИЕ И ТЕХНОЛОГИЯ ПРОИЗВОДСТВА ЭЛЕКТРОННОЙ АППАРАТУРЫ

Отчет о выполнении

практического задания №3

«Графы»

по курсу «Функциональная логика и теория алгоритмов»

Выполнил: студент каф. ИУ4-21Б

Горохов Иван Алексеевич

Проверил: д.т.н., профессор, профессор

каф. ИУ-4

Л.А. Зинченко

Москва 2024

**Цель работы**

Создание программы, проверяющей граф на связанность.

**Исходные данные**

Написать на языке Си и отладить программу, реализующую следующую функцию:

Проверка графа заданного матрицей смежности на связанность

**Выполнение**

Программа принимает на вход путь до файла, содержащего матрицу смежности (поддерживается перетаскивание файла в программу), и с помощью graphviz создает svg изображение построенного графа, затем проверяет, удовлетворяет ли граф условию связанности. Программный код, добавленный после выполнения задания №2, выделен жирным.

#include <stdio.h>

#include <stdlib.h>

#include <locale.h>

unsigned long getlen(FILE \*fp, int separator, int ending);

int \*edgeptr(void \*graph, unsigned long x, unsigned long y);

void \*readfrom(char \*fname);

unsigned long long size(len){ return ((len \* (len+1ULL))>>1); }

void main(int argc, char \*argv[]){

setlocale(LC\_ALL, "Russian");

//printf("sizes %i %i\n", sizeof(unsigned long), sizeof(int));

if (argc < 2) {

printf("No filename provided");

} else if (argc > 2) {

printf("To many arguments");

} else {

void \*graph = readfrom(argv[1]);

if (graph == NULL){

printf("Error while reading adjacency matrix from \"%s\"\n", argv[1]);

} else{

writeto(argv[1], graph);

call\_dot(argv[1]);

char \*svg = malloc(strlen(argv[1]) + 5);

strcpy(svg, argv[1]);

strcat(svg, ".svg");

printf("Success: file \"%s\" created\n", svg);

system(svg);

**unsigned long long edgecount = count(graph);**

**unsigned long n = \*((\*unsigned long) graph);**

**if (edgecount >= ((n-1)\*(n-2)/2)){**

**printf("Inequality %i >= (%i-1)\*(%i-2)/2 holds\n", edgecount, n ,n);**

**printf("Graph is connected\n");**

**} else {**

**printf("Inequality %i >= (%i-1)\*(%i-2)/2 does not hold\n", edgecount, n ,n);**

**printf("Graph could be disconnected\n");**

**}**

}

}

printf("Any key to continue\n");

getchar();

}

void call\_dot(char path[]){

char \*command\_ = malloc(strlen(path)\*2 + 40);

strcpy(command\_, "dot -Tsvg ");

strcat(command\_, path);

strcat(command\_, ".gv > ");

strcat(command\_, path);

strcat(command\_, ".svg");

//printf(command\_);

system(command\_);

free(command\_);

}

void \*readfrom(char \*fname){

FILE \*fp;

if ((fp = fopen(fname, "r")) == NULL) {

printf("Can't open the file\n");

return NULL;

}

unsigned long len = getlen(fp, ' ', '\n');

void \*graph = malloc(sizeof(unsigned long) + size(len)\*sizeof(int));

\*((unsigned long \*) graph) = len;

if (fillfrom(graph, fp) < 0){

printf("Wrong file structure\n");

return NULL;

}

//printf("fillfrom: %i\n", fillfrom(graph, fp));

//printgraph(graph);

fclose(fp);

return(graph);

}

int writeto(char \*fname\_, void \*graph){

char \*fname = malloc(strlen(fname\_) + 4);

strcpy(fname, fname\_);

strcat(fname, ".gv");

FILE \*fp;

if ((fp = fopen(fname, "w")) == NULL)

{

printf("Не удалось открыть файл \"%s\"\n", fname);

return -1;

}

fprintf(fp, "graph {\n");

unsigned long len = \*((unsigned long\*) graph);

for (unsigned long i=0; i<len; i++) fprintf(fp, "%i\n", i);

for (unsigned long i=0; i<len; i++){

for (unsigned long j=i; j<len; j++){

int edge = \*edgeptr(graph, i, j);

for (int e=0; e<edge; e++) fprintf(fp, "%i -- %i\n", i, j);

}

}

fprintf(fp, "}");

fclose(fp);

return(graph);

}

int \*edgeptr(void \*graph, unsigned long x, unsigned long y){

unsigned long len = \*((unsigned long\*) graph);

if (x>y) {

unsigned long temp = x;

x=y;

y=temp;

}

int \*ptr = (graph + sizeof(unsigned long) + (len\*x - ((x-1)\*x)/2 + y-x)\*sizeof(int));

//printf("ePtr:%p:%p", graph, ptr);

return ptr;

}

int fillfrom(void \*graph, FILE \*fp){

unsigned long len = \*((unsigned long\*) graph);

int current = -1;

//printf("fillfrom of lenght %i\n", len);

for(unsigned long i=0; i<len; i++){

for(unsigned long j=0; j<i; j++){ // only checks

current = readuint(fp);

if (current<0) return(current);

if (\*edgeptr(graph, i, j) != current) return(-2);

}

for(unsigned long j=i; j<len; j++){ // fills

current = readuint(fp);

if (current<0) return(current);

\*edgeptr(graph, i, j) = current;

}

if (fgetc(fp) != '\n') return(-3);

}

//printf("\n");

return 0;

}

int readuint(FILE \*fp){

int current = fgetc(fp);

int value = -1;

while (('0' <= current) && (current <= '9')){

if (value < 0) value = 0;

value \*= 10;

value += current - '0';

current = fgetc(fp);

}

//printf("\n%i:",value);

return value;

}

unsigned long getlen(FILE \*fp, int separator, int ending){

unsigned long counter = 0;

int current = fgetc(fp);

while (current != ending){

counter += current == separator;

current = fgetc(fp);

}

fseek(fp, 0, SEEK\_SET);

return counter;

}

void printfile(FILE \*fp){

int current = fgetc(fp);

while (current > 0){

printf("%c",current);

current = fgetc(fp);

}

}

void printgraph(void \*graph){

unsigned long len = \*((unsigned long\*) graph);

unsigned long long s = size(len);

printf("printgraph of size %i\n",s);

int \*edges = (int \*) (graph + sizeof(unsigned long));

int toskip = 0;

int fromlastskip = 0;

for (unsigned long long i=0; i<s; i++){

printf("%2i ",edges[i]);

fromlastskip++;

if (fromlastskip == len){

toskip++;

printf("\n");

for(int j=0;j<toskip;j++) printf(" ");

fromlastskip = toskip;

}

}

printf("\n");

}

**unsigned long long count(void \*graph){**

**unsigned long len = \*((unsigned long\*) graph);**

**unsigned long long s = size(len);**

**printf("printgraph of size %i\n",s);**

**int \*edges = (int \*) (graph + sizeof(unsigned long));**

**unsigned long long counter = 0;**

**for (unsigned long i=0; i<len; i++){**

**for (unsigned long j = i+1; j<len; j++){**

**counter += (\*edgeptr(graph, i, j) >0);**

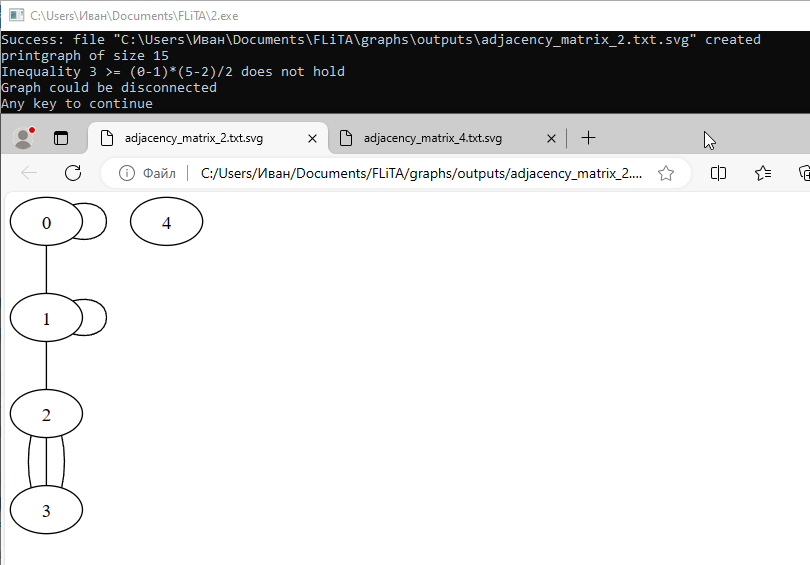
**}**

**}**

**return(counter);**

**}**

**3 Результат работы**Программа создает и открывает .svg файл, содержащий изображение графа, заданного данной матрицей смежности. В случае удовлетворения (рис. 1) или неудовлетворения (рис.2) условию смежности выводит соответствующее сообщение.

  
Рисунок 1

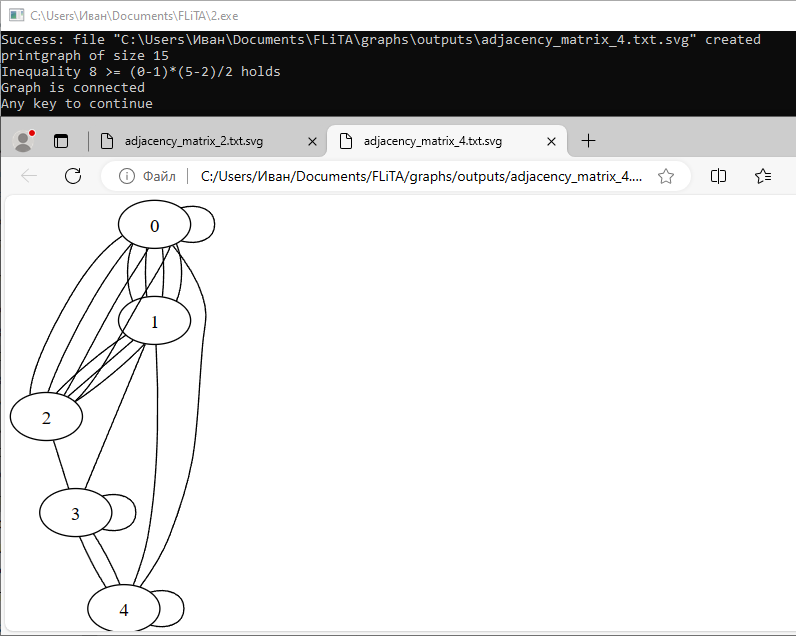


Рисунок 2

**4. Вывод**

Программа корректно исполняет поставленные задачи, а именно проверяет граф на соответствие условию связанности.