**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ**

**УНИВЕРСИТЕТ им. В. Г. ШУХОВА»**

**(БГТУ им. В.Г. Шухова)**



ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И УПРАВЛЯЮЩИХ СИСТЕМ

**Лабораторная работа №4.1**

по дисциплине: Дискретная математика

тема: «Графы»

Выполнил: ст. группы ВТ-231

Масленников Даниил Александрович

Проверили:

Бондаренко Татьяна Владимировна

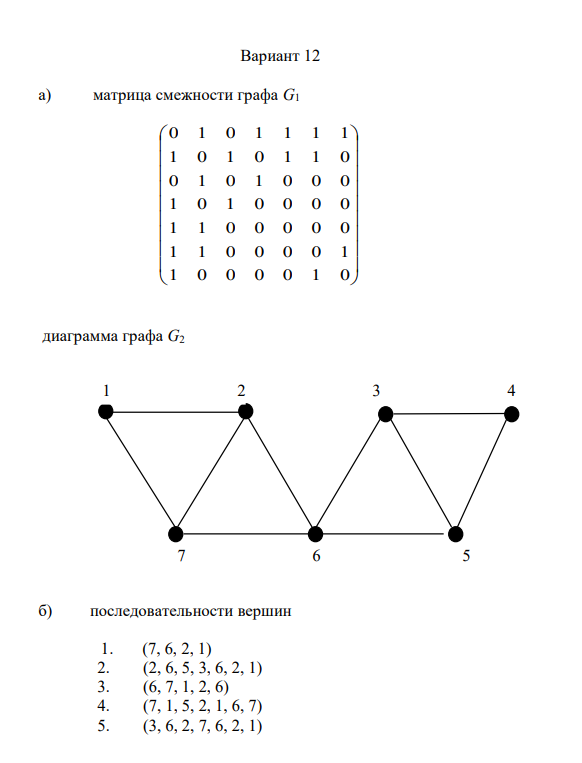
Рязанов Юрий Дмитриевич

Белгород 2023 г.

Л а б о р а т о р н а я р а б о т а № 4.1

**Цель работы:** изучить основные понятия теории графов, способы задания графов, научиться программно реализовывать алгоритмы получения и анализа маршрутов в графах.

**Вариант 12**



**Задания**

1. Представить графы G1 и G2 (см. ”Варианты заданий”, п.а) матрицей смежности, матрицей инцидентности, диаграммой.

Граф G1:

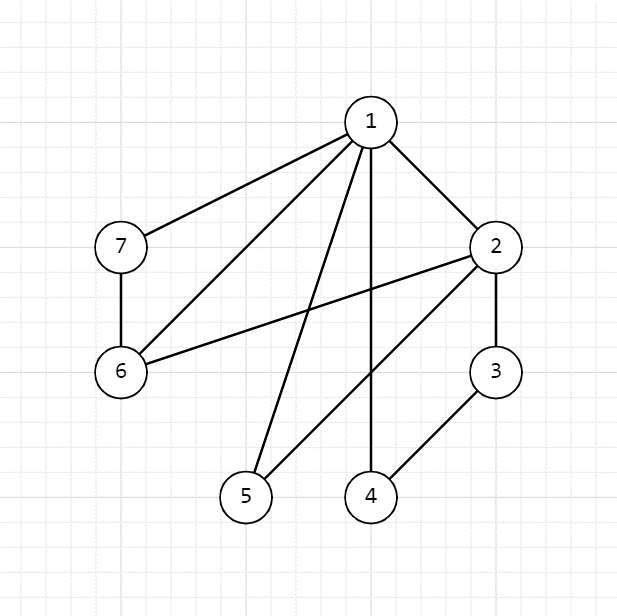
Матрица смежности:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| G1 | v1 | v2 | v3 | v4 | v5 | v6 | v7 |
| v1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |
| v2 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| v3 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| v4 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| v5 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| v6 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| v7 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |

Матрица инцидентности:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| v1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| v2 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| v3 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| v4 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| v5 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| v6 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| v7 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |

Диаграмма:



Граф G2:

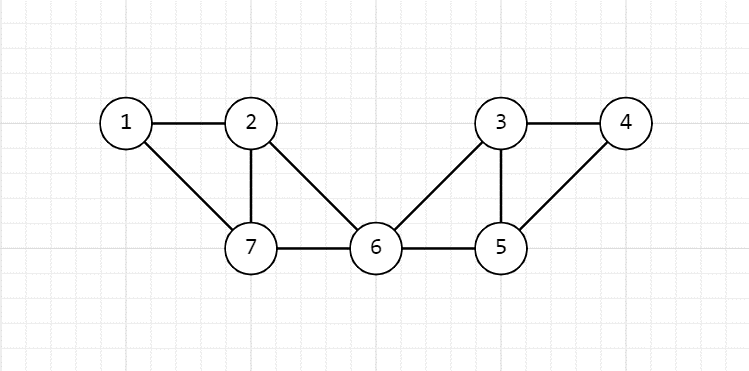
Матрица смежности:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| G2 | v1 | v2 | v3 | v4 | v5 | v6 | v7 |
| v1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| v2 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| v3 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| v4 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| v5 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| v6 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| v7 | 1 | 1 | 0 | 0 | 0 | 1 | 0 |

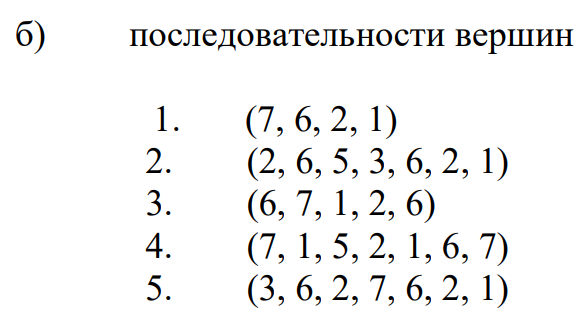
Матрица инцидентности:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| v1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| v2 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| v3 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| v4 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| v5 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 |
| v6 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| v7 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |

Диаграмма:



1. Определить, являются ли последовательности вершин (см. ”Варианты заданий”, п.б) маршрутом, цепью, простой цепью, циклом, простым циклом в графах G1 и G2 (см. ”Варианты заданий”, п.а).



|  |  |  |
| --- | --- | --- |
| Последовательность: | Графы: | |
| (7, 6, 2, 1) | G1: | G2: |
| Маршрут: | + | + |
| Цепь: | + | + |
| Простая цепь: | + | + |
| Цикл: | - | - |
| Простой цикл: | - | - |

|  |  |  |
| --- | --- | --- |
| Последовательность: | Графы: | |
| (2, 6, 5, 3, 6, 2, 1) | G1: | G2: |
| Маршрут: | - | + |
| Цепь: | - | - |
| Простая цепь: | - | - |
| Цикл: | - | - |
| Простой цикл: | - | - |

|  |  |  |
| --- | --- | --- |
| Последовательность: | Графы: | |
| (6, 7, 1, 2, 6) | G1: | G2: |
| Маршрут: | + | + |
| Цепь: | + | + |
| Простая цепь: | - | - |
| Цикл: | + | + |
| Простой цикл: | + | + |

|  |  |  |
| --- | --- | --- |
| Последовательность: | Графы: | |
| (7, 1, 5, 2, 1, 6, 7) | G1: | G2: |
| Маршрут: | + | - |
| Цепь: | + | - |
| Простая цепь: | - | - |
| Цикл: | + | - |
| Простой цикл: | - | - |

|  |  |  |
| --- | --- | --- |
| Последовательность: | Графы: | |
| (3, 6, 2, 7, 6, 2, 1) | G1: | G2: |
| Маршрут: | - | + |
| Цепь: | - | - |
| Простая цепь: | - | - |
| Цикл: | - | - |
| Простой цикл: | - | - |

1. Написать программу, определяющую, является ли заданная последовательность вершин (см. ”Варианты заданий”, п.б) маршрутом, цепью, простой цепью, циклом, простым циклом в графах G1 и G2 (см.”Варианты заданий”, п.а).

Программа:

#include <stdio.h>

#include <stdbool.h>

#include <stdlib.h>

#define MAX\_SIZE 7

void matrixOutput(bool matrix[MAX\_SIZE][MAX\_SIZE]) {

for (int i = 0; i < MAX\_SIZE; i++) {

for (int j = 0; j < MAX\_SIZE; j++) {

printf("%d ", matrix[i][j]);

}

printf("\n");

}

printf("\n");

}

void matrixInput(bool matrix[MAX\_SIZE][MAX\_SIZE]) {

for (int i = 0; i < MAX\_SIZE; i++) {

for (int j = 0; j < MAX\_SIZE; j++) {

int buf;

scanf("%d", &buf);

matrix[i][j] = buf;

}

}

}

bool IsRout(bool matrix[MAX\_SIZE][MAX\_SIZE], int rout[], int size) {

for (int i = 0; i < size - 1; i++) {

if (!matrix[rout[i] - 1][rout[i + 1] - 1]) {

return false;

}

}

return true;

}

bool IsChain(bool matrix[MAX\_SIZE][MAX\_SIZE], int rout[], int size) {

bool chain[MAX\_SIZE][MAX\_SIZE] = { false };

bool isNotChain = false;

for (int i = 0; i < size - 1; i++) {

isNotChain = false;

for (int j = 0; j < MAX\_SIZE; j++) {

if (chain[j][0] == rout[i] && chain[j][1] == rout[i + 1] ||

chain[j][1] == rout[i] && chain[j][0] == rout[i + 1]) {

isNotChain = true;

break;

}

}

if (!matrix[rout[i] - 1][rout[i + 1] - 1] && !isNotChain) {

return false;

}

chain[i][0] = rout[i];

chain[i][1] = rout[i + 1];

}

return !isNotChain;

}

bool IsPrimeChain(bool matrix[MAX\_SIZE][MAX\_SIZE], int rout[], int size) {

if (!IsRout(matrix, rout, size)) {

return false;

}

for (int i = 0; i < size - 1; i++) {

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

if (rout[i] == rout[j]) {

return false;

}

}

}

return true;

}

bool IsCycle(bool matrix[MAX\_SIZE][MAX\_SIZE], int rout[], int size) {

if (rout[0] != rout[size - 1]) {

return false;

}

return IsChain(matrix, rout, size);

}

bool IsPrimeCycle(bool matrix[MAX\_SIZE][MAX\_SIZE], int rout[], int size) {

if (rout[0] != rout[size - 1]) {

return false;

}

for (int i = 0; i < size - 1; i++) {

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

if (i == 0 && j == size - 1) {

continue;

} else if (rout[i] == rout[j]) {

return false;

}

}

}

return true;

}

void outputVector(int v[], int size) {

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

printf("%d ", v[i]);

}

printf("\n");

}

void outputSequenceProperty(bool matrix[MAX\_SIZE][MAX\_SIZE], int sequence[], int size) {

printf("For ");

outputVector(sequence, size);

printf("IsRout: %d\n", IsRout(matrix, sequence, size));

printf("IsChain: %d\n", IsChain(matrix, sequence, size));

printf("IsPrimeChain: %d\n", IsPrimeChain(matrix, sequence, size));

printf("IsCycle: %d\n", IsCycle(matrix, sequence, size));

printf("IsPrimeCycle: %d\n", IsPrimeCycle(matrix, sequence, size));

printf("\n");

}

int main() {

bool matrix[MAX\_SIZE][MAX\_SIZE] = { false };

matrixInput(matrix);

int rout\_1[] = { 7, 6, 2, 1 };

int rout\_2[] = { 2, 6, 5, 3, 6, 2, 1 };

int rout\_3[] = { 6, 7, 1, 2, 6 };

int rout\_4[] = { 7, 1, 5, 2, 1, 6, 7 };

int rout\_5[] = { 3, 6, 2, 7, 6, 2, 1 };

outputSequenceProperty(matrix, rout\_1, 4);

outputSequenceProperty(matrix, rout\_2, 7);

outputSequenceProperty(matrix, rout\_3, 5);

outputSequenceProperty(matrix, rout\_4, 7);

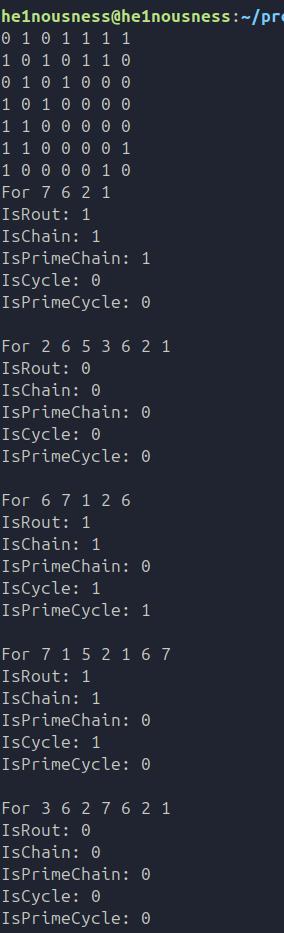
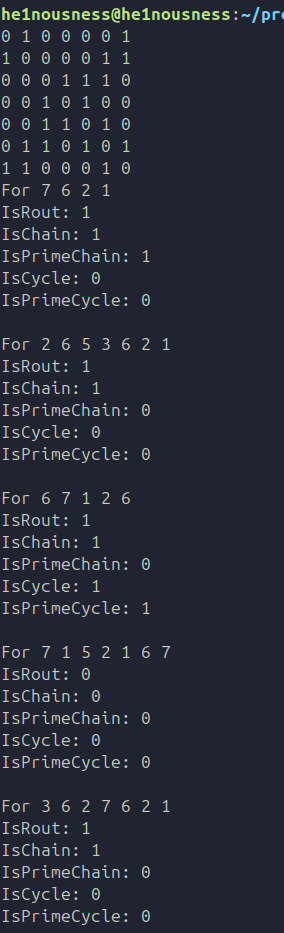
outputSequenceProperty(matrix, rout\_5, 7);

return 0;

}

Вывод программы:

G1: G2:

1. Написать программу, получающую все маршруты заданной длины, выходящие из заданной вершины. Использовать программу для получения всех маршрутов заданной длины в графах G1 и G2 (см. ”Варианты заданий”, п.а).

Программа:

#include <stdio.h>

#include <stdbool.h>

#define MAX\_SIZE 7

void matrixInput(bool matrix[MAX\_SIZE][MAX\_SIZE]) {

for (int i = 0; i < MAX\_SIZE; i++) {

for (int j = 0; j < MAX\_SIZE; j++) {

int buf;

scanf("%d", &buf);

matrix[i][j] = (buf != 0);

}

}

}

void outputVector(int v[], int size) {

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

printf("%d ", v[i]);

}

printf("\n");

}

void outputVertex(int v[][MAX\_SIZE], int count, int maxLen) {

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

outputVector(v[i], maxLen);

}

}

void GetRoutes\_(bool matrix[MAX\_SIZE][MAX\_SIZE], int maxLen, int G[], int gSize, int res[][MAX\_SIZE], int\* count) {

for (int i = 0; i < MAX\_SIZE; i++) {

if (matrix[i][G[gSize - 1] - 1]) {

G[gSize] = i + 1; // добавляем вершину к пути

if (gSize + 1 == maxLen) {

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

res[\*count][j] = G[j];

}

(\*count)++;

} else {

GetRoutes\_(matrix, maxLen, G, gSize + 1, res, count);

}

}

}

}

int GetRoutes(bool matrix[MAX\_SIZE][MAX\_SIZE], int v, int maxLen, int res[][MAX\_SIZE]) {

int count = 0;

int G[MAX\_SIZE];

G[0] = v; // начинаем с вершины v

GetRoutes\_(matrix, maxLen, G, 1, res, &count);

return count;

}

int GetAllRoutes(bool matrix[MAX\_SIZE][MAX\_SIZE], int maxLen, int res[][MAX\_SIZE]) {

int count = 0;

for (int i = 1; i <= MAX\_SIZE; i++) {

int buf[MAX\_SIZE][MAX\_SIZE];

int bufCount = GetRoutes(matrix, i, maxLen, buf);

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

for (int k = 0; k < maxLen; k++) {

res[count][k] = buf[j][k];

}

count++;

}

}

return count;

}

int main() {

bool matrix[MAX\_SIZE][MAX\_SIZE] = { false };

matrixInput(matrix);

int routes[MAX\_SIZE][MAX\_SIZE]; // массив для хранения маршрутов

int routeCount = GetRoutes(matrix, 1, 3, routes); // 1 - начальная вершина, 2 - максимальная длина маршрута

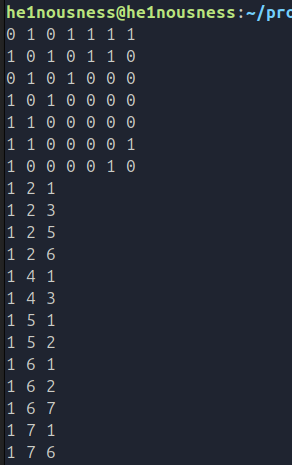
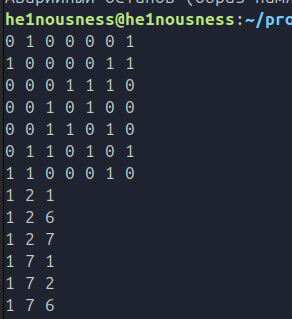
outputVertex(routes, routeCount, 3); // выводим маршруты

return 0;

}

Вывод программы:

G1: G2:

1. Написать программу, определяющую количество маршрутов заданной длины между каждой парой вершин графа. Использовать программу для определения количества маршрутов заданной длины между каждой парой вершин в графах G1 и G2 (см. ”Варианты заданий”, п.а).

Программа:

#include <stdio.h>

#include <stdbool.h>

#define MAX\_SIZE 7

void matrixInput(bool matrix[MAX\_SIZE][MAX\_SIZE]) {

for (int i = 0; i < MAX\_SIZE; i++) {

for (int j = 0; j < MAX\_SIZE; j++) {

int buf;

scanf("%d", &buf);

matrix[i][j] = buf != 0; // Записываем true, если buf не 0

}

}

}

int GetSumRoutes\_(bool matrix[MAX\_SIZE][MAX\_SIZE], int maxLen, int G[], int gSize, int vEnd) {

int sumRoutes = 0;

for (int i = 0; i < MAX\_SIZE; i++) {

if (matrix[i][G[gSize - 1] - 1]) {

G[gSize] = i + 1;

if (gSize == maxLen && vEnd == i + 1) {

sumRoutes += 1;

} else if (gSize < maxLen) {

sumRoutes += GetSumRoutes\_(matrix, maxLen, G, gSize + 1, vEnd);

}

G[gSize] = 0; // Отмена добавления вершины

}

}

return sumRoutes;

}

int GetSumRoutes(bool matrix[MAX\_SIZE][MAX\_SIZE], int v, int maxLen, int vEnd) {

int G[MAX\_SIZE] = {0};

G[0] = v; // Начинаем с вершины v

return GetSumRoutes\_(matrix, maxLen, G, 1, vEnd);

}

void GetAllSumRoutes(bool matrix[MAX\_SIZE][MAX\_SIZE], int maxLen) {

for (int i = 1; i <= MAX\_SIZE; i++) {

for (int j = 1; j <= MAX\_SIZE; j++) {

int sum = GetSumRoutes(matrix, i, maxLen, j);

printf("%d -> %d : %d\n", i, j, sum);

}

}

}

int main() {

bool matrix[MAX\_SIZE][MAX\_SIZE] = { false };

matrixInput(matrix);

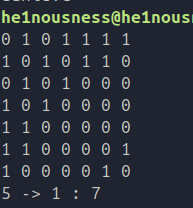
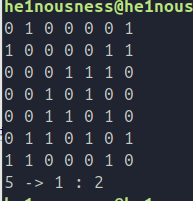
printf("5 -> 1 : %d\n", GetSumRoutes(matrix, 5, 3, 1));

return 0;

}

Вывод программы:

G1: G2:

1. Написать программу, определяющую все маршруты заданной длины между заданной парой вершин графа. Использовать программу для определения всех маршрутов заданной длины между заданной парой вершин в графах G1 и G2 (см. ”Варианты заданий”, п.а).

Программа:

#include <stdio.h>

#include <stdbool.h>

#define MAX\_SIZE 7

#define MAX\_PATH\_LENGTH 10 // Задаем максимальную длину пути

void matrixInput(bool matrix[MAX\_SIZE][MAX\_SIZE]) {

for (int i = 0; i < MAX\_SIZE; i++) {

for (int j = 0; j < MAX\_SIZE; j++) {

int buf;

scanf("%d", &buf);

matrix[i][j] = buf != 0; // Если buf не равен 0, устанавливаем true

}

}

}

void outputVector(int v[], int size) {

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

printf("%d ", v[i]);

}

printf("\n");

}

void outputVertex(int res[][MAX\_PATH\_LENGTH], int sizes[], int sizeCount) {

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

outputVector(res[i], sizes[i]);

}

}

void GetPairRoutes\_(bool matrix[MAX\_SIZE][MAX\_SIZE], int maxLen, int G[], int gSize, int vEnd, int res[][MAX\_PATH\_LENGTH], int \*resultCount, int sizes[]) {

for (int i = 0; i < MAX\_SIZE; i++) {

if (matrix[i][G[gSize - 1] - 1]) {

G[gSize] = i + 1;

if (gSize == maxLen && vEnd == (i + 1)) {

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

res[\*resultCount][j] = G[j];

}

sizes[\*resultCount] = gSize + 1;

(\*resultCount)++;

} else if (gSize < maxLen) {

GetPairRoutes\_(matrix, maxLen, G, gSize + 1, vEnd, res, resultCount, sizes);

}

}

}

}

void GetPairRoutes(bool matrix[MAX\_SIZE][MAX\_SIZE], int v, int maxLen, int vEnd, int res[][MAX\_PATH\_LENGTH], int \*resultCount, int sizes[]) {

int G[MAX\_PATH\_LENGTH] = {0};

G[0] = v; // Начинаем с вершины v

GetPairRoutes\_(matrix, maxLen, G, 1, vEnd, res, resultCount, sizes);

}

int main() {

bool matrix[MAX\_SIZE][MAX\_SIZE] = { false };

matrixInput(matrix);

int res[MAX\_SIZE \* MAX\_SIZE][MAX\_PATH\_LENGTH] = {0}; // Получаем массив путей

int sizes[MAX\_SIZE \* MAX\_SIZE] = {0}; // Для хранения длин путей

int resultCount = 0; // Количество найденных путей

GetPairRoutes(matrix, 5, 3, 1, res, &resultCount, sizes);

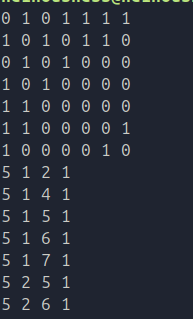
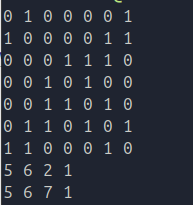
outputVertex(res, sizes, resultCount);

return 0;

}

Вывод программы:

G1: G2:

1. Написать программу, получающую все простые максимальные цепи, выходящие из заданной вершины графа. Использовать программу для получения всех простые максимальных цепей, выходящих из заданной вершины в графах G1 и G2 (см. ”Варианты заданий”, п.а).

Программа:

#include <stdio.h>

#include <stdbool.h>

#define MAX\_SIZE 7

#define MAX\_PATH\_LENGTH 10 // Задаем максимальную длину пути

void matrixInput(bool matrix[MAX\_SIZE][MAX\_SIZE]) {

for (int i = 0; i < MAX\_SIZE; i++) {

for (int j = 0; j < MAX\_SIZE; j++) {

int buf;

scanf("%d", &buf);

matrix[i][j] = buf != 0; // Если buf не равен 0, устанавливаем true

}

}

}

void outputVector(int v[], int size) {

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

printf("%d ", v[i]);

}

printf("\n");

}

void outputVertex(int res[][MAX\_PATH\_LENGTH], int sizes[], int sizeCount) {

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

outputVector(res[i], sizes[i]);

}

}

void GetMaxChains\_(bool matrix[MAX\_SIZE][MAX\_SIZE], int G[], int gSize, int res[][MAX\_PATH\_LENGTH], int \*resultCount, int sizes[]) {

bool areNoWaysToAnyPoint = true;

for (int i = 0; i < MAX\_SIZE && gSize <= MAX\_SIZE; i++) {

if (matrix[i][G[gSize - 1] - 1]) {

bool isNoPointInG = true;

for (int j = 0; j < gSize && isNoPointInG; j++) {

if (G[j] == (i + 1)) {

isNoPointInG = false;

}

}

areNoWaysToAnyPoint = areNoWaysToAnyPoint && !isNoPointInG;

if (isNoPointInG) {

G[gSize] = i + 1;

GetMaxChains\_(matrix, G, gSize + 1, res, resultCount, sizes);

}

}

}

if (areNoWaysToAnyPoint) {

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

res[\*resultCount][j] = G[j];

}

sizes[\*resultCount] = gSize;

(\*resultCount)++;

}

}

void GetMaxChains(bool matrix[MAX\_SIZE][MAX\_SIZE], int v, int res[][MAX\_PATH\_LENGTH], int \*resultCount, int sizes[]) {

int G[MAX\_PATH\_LENGTH] = {0};

G[0] = v; // Начинаем с вершины v

GetMaxChains\_(matrix, G, 1, res, resultCount, sizes);

}

int main() {

bool matrix[MAX\_SIZE][MAX\_SIZE] = { false };

matrixInput(matrix);

int res[MAX\_SIZE \* MAX\_SIZE][MAX\_PATH\_LENGTH] = {0}; // Массив для хранения цепочек

int sizes[MAX\_SIZE \* MAX\_SIZE] = {0}; // Для хранения длин цепочек

int resultCount = 0; // Количество найденных цепочек

GetMaxChains(matrix, 1, res, &resultCount, sizes);

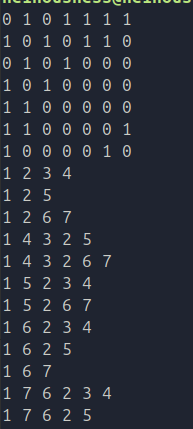
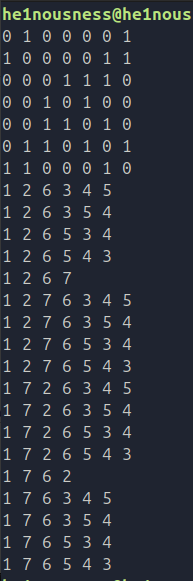
outputVertex(res, sizes, resultCount);

return 0;

}

Вывод программы:

G1: G2:

**Вывод**

**Вывод**: в ходе работы я изучил основные понятия теории графов, способы задания графов, научился программно реализовывать алгоритмы получения и анализа маршрутов в графах.