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)**

Кафедра программного обеспечения вычислительной

техники и автоматизированных систем

**Лабораторная работа №4.2**

по дисциплине: «Дискретная математика»

по теме: «Циклы»

Выполнил: ст. группы ВТ-231

Масленников Даниил Александрович

Проверили:

Рязанов Юрий Дмитриевич

Белгород, 2024 г.

**Цель работы:** изучить разновидности циклов в графах, научиться генерировать случайные графы, определять их принадлежность к множеству эйлеровых и гамильтоновых графов, находить все эйлеровы и гамильтоновы циклы в графах.

**Задания**

1. Разработать и реализовать алгоритм генерации случайного графа, содержащего n вершин и m ребер.

2. Написать программу, которая:

а) в течение десяти секунд генерирует случайные графы, содержащие n вершин и m ребер;

б) для каждого полученного графа определяет, является ли он эйлеровым или гамильтоновым;

в) подсчитывает общее количество сгенерированных графов и количество графов каждого типа.

Результат работы программы представить в виде таблицы (табл. 4). Величину h подобрать такой, чтобы в таблице количество строк было в диапазоне от 20 до 30.

*Таблица 4*

**Результат работы программы**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Количество вершин | Количество ребер | Количество графов | | |
| эйлеровых | гамильтоновых | всех |
| n | n |  | | |
| n | n+h |  | | |
| n | n+2h |  | | |
|  |  |  | | |
| n |  |  | | |

3. Выполнить программу при n = 8, 9, 10 и сделать выводы.

4. Привести пример диаграммы графа, который является эйлеровым, но не гамильтоновым. Найти в нем все эйлеровы циклы.

5. Привести пример диаграммы графа, который является гамильтоновым, но не эйлеровым. Найти в нем все гамильтоновы циклы.

6. Привести пример диаграммы графа, который является эйлеровым и гамильтоновым. Найти в нем все эйлеровы и гамильтоновы циклы.

7. Привести пример диаграммы графа, который не является ни эйлеровым, ни гамильтоновым.

**1. Разработать и реализовать алгоритм генерации случайного графа, содержащего n вершин и m ребер:**

typedef struct matrix {

int\*\* values; // элементы матрицы

int nRows; // количество рядов

int nCols; // количество столбцов

} matrix;

// размещает в динамической памяти матрицу размером nRows на nCols.

matrix getMemMatrix(int nRows, int nCols) {

int\*\* values = (int\*\*)malloc(sizeof(int\*) \* nRows);

for (int i = 0; i < nRows; i++)

values[i] = (int\*)malloc(sizeof(int) \* nCols);

return (matrix) { values, nRows, nCols };

}

// возвращает матрицу размера nRows на nCols, построенную из элементов массива a

matrix createMatrixFromArray(const int\* a, size\_t nRows, size\_t nCols) {

matrix m = getMemMatrix(nRows, nCols);

int k = 0;

for (int i = 0; i < nRows; i++)

for (int j = 0; j < nCols; j++)

m.values[i][j] = a[k++];

return m;

}

// Степень вершины

int degreeOfVertices(int\* a, int n) {

int c = 0;

for (int i = 0; i < n; i++)

c += a[i];

return c;

}

// Алгоритм генерации случайного графа

matrix generationGraph(int n, int m) {

srand(time(NULL));

matrix G = getMemMatrix(n, n);

for (int i = 0; i < n; i++)

for (int j = 0; j < n; j++)

G.values[i][j] = 0;

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

int r\_n = rand() % n;

int r\_m = rand() % n;

while (G.values[r\_n][r\_m] || (r\_n == r\_m)) {

r\_n = rand() % n;

r\_m = rand() % n;

}

G.values[r\_n][r\_m] = 1;

G.values[r\_m][r\_n] = 1;

}

for (int i = 0; i < n; i++)

if (!degreeOfVertices(G.values[i], n)) {

int r\_m = rand() % n;

G.values[i][r\_m] = 1;

G.values[r\_m][i] = 1;

}

return G;

}

**2. Написать программу, которая:**

**а) в течение десяти секунд генерирует случайные графы, содержащие n вершин и m ребер;**

**б) для каждого полученного графа определяет, является ли он эйлеровым или гамильтоновым;**

**в) подсчитывает общее количество сгенерированных графов и количество графов каждого типа.**

**Результат работы программы представить в виде таблицы. Величину h подобрать такой, чтобы в таблице количество строк было в диапазоне от 20 до 30:**

typedef struct matrix {

int\*\* values; // элементы матрицы

int nRows; // количество рядов

int nCols; // количество столбцов

} matrix;

// размещает в динамической памяти матрицу размером nRows на nCols.

matrix getMemMatrix(int nRows, int nCols) {

int\*\* values = (int\*\*)malloc(sizeof(int\*) \* nRows);

for (int i = 0; i < nRows; i++)

values[i] = (int\*)malloc(sizeof(int) \* nCols);

return (matrix) { values, nRows, nCols };

}

// возвращает матрицу размера nRows на nCols, построенную из элементов массива a

matrix createMatrixFromArray(const int\* a, size\_t nRows, size\_t nCols) {

matrix m = getMemMatrix(nRows, nCols);

int k = 0;

for (int i = 0; i < nRows; i++)

for (int j = 0; j < nCols; j++)

m.values[i][j] = a[k++];

return m;

}

// освобождает память, выделенную под хранение матрицы m.

void freeMemMatrix(matrix\* m) {

for (int i = 0; i < m->nRows; i++)

free(m->values[i]);

free(m->values);

m->values = NULL;

m->nRows = 0;

m->nCols = 0;

}

// Определяет, является ли граф гамильтоновым

bool isHamiltonGraph(matrix\* G) {

if (G->nRows < 3)

return 0;

else

for (int i = 0; i < G->nRows; i++)

for (int j = 0; j < G->nCols; j++)

if (!G->values[i][j] && (i != j) && (degreeOfVertices(G->values[i], G->nRows) + degreeOfVertices(G->values[j], G->nRows) < G->nRows))

return 0;

return 1;

}

// Определяет, является ли граф эйлеровым

bool isEulerGraph(matrix\* G) {

for (int i = 0; i < G->nRows; i++){

int d = degreeOfVertices(G->values[i], G->nRows);

if (!d || d % 2 != 0)

return 0;

}

return 1;

}

// Степень вершины

int degreeOfVertices(int\* a, int n) {

int c = 0;

for (int i = 0; i < n; i++)

c += a[i];

return c;

}

// Алгоритм генерации случайного графа

matrix generationGraph(int n, int m) {

srand(time(NULL));

matrix G = getMemMatrix(n, n);

for (int i = 0; i < n; i++)

for (int j = 0; j < n; j++)

G.values[i][j] = 0;

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

int r\_n = rand() % n;

int r\_m = rand() % n;

while (G.values[r\_n][r\_m] || (r\_n == r\_m)) {

r\_n = rand() % n;

r\_m = rand() % n;

}

G.values[r\_n][r\_m] = 1;

G.values[r\_m][r\_n] = 1;

}

for (int i = 0; i < n; i++)

if (!degreeOfVertices(G.values[i], n)) {

int r\_m = rand() % n;

G.values[i][r\_m] = 1;

G.values[r\_m][i] = 1;

}

return G;

}

// В течении 10 секунд генерирует случайные графы, определяет кол-во эйлеровых, гамильтоновых, всего

void getCharacteristicGraph(int n, int m) {

int count\_of\_Euler\_graphs = 0;

int count\_of\_Hamilton\_graphs = 0;

int count\_of\_graphs = 0;

time\_t start = time(NULL);

while (time(NULL) - start < 10) {

matrix G = generationGraph(n, m);

count\_of\_Euler\_graphs += isEulerGraph(&G);

count\_of\_Hamilton\_graphs += isHamiltonGraph(&G);

count\_of\_graphs++;

freeMemMatrix(&G);

}

printf("При %d вершинах и %d ребрах, получилось %d эйлеровых графов и %d гамильтоновых графов, всего вышло %d графов\n", n, m, count\_of\_Euler\_graphs, count\_of\_Hamilton\_graphs, count\_of\_graphs);

}

**3. Выполнить программу при n = 8, 9, 10 и сделать выводы:**

При n = 8:

**Результат работы программы**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Количество вершин | Количество ребер | Количество графов | | |
| эйлеровых | гамильтоновых | всех |
| 8 | 8 | 0 | 0 | 4609387 |
| 8 | 9 | 0 | 0 | 4959361 |
| 8 | 10 | 0 | 0 | 4784027 |
| 8 | 11 | 0 | 0 | 4488896 |
| 8 | 12 | 0 | 0 | 4229834 |
| 8 | 13 | 0 | 0 | 4151570 |
| 8 | 14 | 0 | 0 | 4139124 |
| 8 | 15 | 0 | 0 | 3821654 |
| 8 | 16 | 0 | 0 | 3654798 |
| 8 | 17 | 0 | 0 | 3758797 |
| 8 | 18 | 0 | 0 | 3354382 |
| 8 | 19 | 0 | 0 | 3357758 |
| 8 | 20 | 0 | 0 | 3062470 |
| 8 | 21 | 0 | 1064254 | 2903493 |
| 8 | 22 | 0 | 720741 | 2661827 |
| 8 | 23 | 0 | 1915773 | 2437575 |
| 8 | 24 | 0 | 1657659 | 2421169 |
| 8 | 25 | 0 | 1355378 | 2405257 |
| 8 | 26 | 0 | 1773484 | 2214903 |
| 8 | 27 | 0 | 2289733 | 2289733 |
| 8 | 28 | 0 | 2147102 | 2147102 |

При n = 9:

**Результат работы программы**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Количество вершин | Количество ребер | Количество графов | | |
| эйлеровых | гамильтоновых | всех |
| 9 | 9 | 0 | 0 | 4516382 |
| 9 | 10 | 0 | 0 | 4377778 |
| 9 | 11 | 0 | 0 | 4231080 |
| 9 | 12 | 0 | 0 | 4137622 |
| 9 | 13 | 380410 | 0 | 4052963 |
| 9 | 14 | 0 | 0 | 3973422 |
| 9 | 15 | 0 | 0 | 3890698 |
| 9 | 16 | 0 | 0 | 3650115 |
| 9 | 17 | 0 | 0 | 3573869 |
| 9 | 18 | 0 | 0 | 3382099 |
| 9 | 19 | 0 | 0 | 3296458 |
| 9 | 20 | 0 | 0 | 3281682 |
| 9 | 21 | 0 | 0 | 3115067 |
| 9 | 22 | 0 | 0 | 2997793 |
| 9 | 23 | 0 | 0 | 2956885 |
| 9 | 24 | 0 | 0 | 2725089 |
| 9 | 25 | 0 | 0 | 2624243 |
| 9 | 26 | 0 | 0 | 2509415 |
| 9 | 27 | 0 | 0 | 2548405 |
| 9 | 28 | 0 | 605092 | 2288503 |
| 9 | 29 | 0 | 195085 | 2240600 |
| 9 | 30 | 0 | 999083 | 2063357 |
| 9 | 31 | 0 | 899557 | 1958827 |
| 9 | 32 | 0 | 904011 | 1949163 |
| 9 | 33 | 0 | 1498851 | 1869429 |
| 9 | 34 | 0 | 1592592 | 1744620 |
| 9 | 35 | 0 | 1722022 | 1722022 |
| 9 | 36 | 0 | 1358801 | 1757877 |

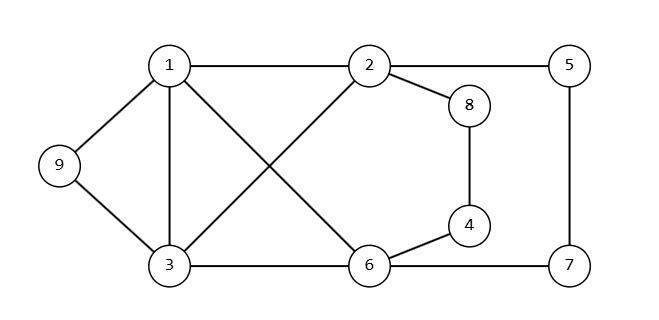
При n = 10:

**Результат работы программы**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Количество вершин | Количество ребер | Количество графов | | |
| эйлеровых | гамильтоновых | всех |
| 10 | 10 | 0 | 0 | 3995001 |
| 10 | 11 | 0 | 0 | 3917604 |
| 10 | 12 | 0 | 0 | 3853457 |
| 10 | 13 | 0 | 0 | 3854793 |
| 10 | 14 | 0 | 0 | 3642999 |
| 10 | 15 | 0 | 0 | 3500671 |
| 10 | 16 | 0 | 0 | 3437625 |
| 10 | 17 | 0 | 0 | 3271048 |
| 10 | 18 | 0 | 0 | 3202233 |
| 10 | 19 | 0 | 0 | 3253638 |
| 10 | 20 | 0 | 0 | 3130427 |
| 10 | 21 | 0 | 0 | 3126881 |
| 10 | 22 | 0 | 0 | 2908959 |
| 10 | 23 | 0 | 0 | 2883568 |
| 10 | 24 | 0 | 0 | 2808118 |
| 10 | 25 | 0 | 0 | 2752091 |
| 10 | 26 | 0 | 0 | 2622863 |
| 10 | 27 | 0 | 0 | 2553861 |
| 10 | 28 | 0 | 0 | 2481398 |
| 10 | 29 | 0 | 0 | 2473545 |
| 10 | 30 | 0 | 0 | 2348171 |
| 10 | 31 | 0 | 0 | 2200146 |
| 10 | 32 | 0 | 344431 | 2076450 |
| 10 | 33 | 0 | 355051 | 2023981 |
| 10 | 34 | 0 | 1119845 | 1681863 |
| 10 | 35 | 0 | 467923 | 1827987 |
| 10 | 36 | 0 | 1274961 | 1654500 |
| 10 | 37 | 167208 | 1280177 | 1623430 |
| 10 | 38 | 0 | 1117884 | 1651591 |
| 10 | 39 | 0 | 1523940 | 1523940 |
| 10 | 40 | 0 | 1220605 | 1546596 |
| 10 | 41 | 0 | 1299649 | 1448411 |
| 10 | 42 | 0 | 1429370 | 1429370 |
| 10 | 43 | 0 | 1397303 | 1397303 |
| 10 | 44 | 0 | 1425560 | 1425560 |
| 10 | 45 | 0 | 1350874 | 1350874 |

Вывод: редкая генерация гамильтоновых и еще более редкая генерация эйлеровых графов, скорее всего связанна с несовершенством алгоритма генерации случайных графов.

**4. Привести пример диаграммы графа, который является эйлеровым, но не гамильтоновым. Найти в нем все эйлеровы циклы:**



typedef struct matrix {

int\*\* values; // элементы матрицы

int nRows; // количество рядов

int nCols; // количество столбцов

} matrix;

// размещает в динамической памяти матрицу размером nRows на nCols.

matrix getMemMatrix(int nRows, int nCols) {

int\*\* values = (int\*\*)malloc(sizeof(int\*) \* nRows);

for (int i = 0; i < nRows; i++)

values[i] = (int\*)malloc(sizeof(int) \* nCols);

return (matrix) { values, nRows, nCols };

}

// возвращает матрицу размера nRows на nCols, построенную из элементов массива a

matrix createMatrixFromArray(const int\* a, size\_t nRows, size\_t nCols) {

matrix m = getMemMatrix(nRows, nCols);

int k = 0;

for (int i = 0; i < nRows; i++)

for (int j = 0; j < nCols; j++)

m.values[i][j] = a[k++];

return m;

}

// освобождает память, выделенную под хранение матрицы m.

void freeMemMatrix(matrix\* m) {

for (int i = 0; i < m->nRows; i++)

free(m->values[i]);

free(m->values);

m->values = NULL;

m->nRows = 0;

m->nCols = 0;

}

// Функция линейного поиска.

int linearSearch(int\* a, int n, int x) {

for (int i = 0; i < n; i++)

if (a[i] == x)

return i;

return n;

}

// Рекурсивная функция нахождения всех эйлеровых циклов в матрице инциндентности графа

void findAllEulerCyclesRecursionGraph(matrix\* G, int v, int\* W, int i, int\* E, int \*count\_of\_edges) {

for (int j = 0; j < G->nCols; j++)

if (G->values[W[i - 1] - 1][j] && (linearSearch(E, G->nCols, j + 1) == G->nCols)) {

for (int k = 0; k < G->nRows; k++)

if (G->values[k][j] && (k != (W[i - 1] - 1))){

W[i] = k + 1;

break;

}

if ((W[i] == v + 1) && (i == G->nCols)){

printf("При v = %d, вывод цикла W: ", v + 1);

for (int m = 0; m <= i; m++)

printf("%d ", W[m]);

printf("\n\n");

}

else{

E[\*count\_of\_edges] = j + 1;

\*count\_of\_edges += 1;

findAllEulerCyclesRecursionGraph(G, v, W, i + 1, E, count\_of\_edges);

}

}

}

// Функция нахождения всех эйлеровых циклов в матрице инциндентности графа, через эту функцию подаются данные в рекурсивную функцию нахождения всех эйлеровых циклов в матрице инциндентности графа

void findAllEulerCyclesGraph(matrix\* G, int v) {

int\* W = (int\*)malloc(sizeof(int) \* (G->nCols + 1));

int\* E = (int\*)malloc(sizeof(int) \* G->nCols);

int count\_of\_edges = 0;

W[0] = v + 1;

findAllEulerCyclesRecursionGraph(G, v, W, 1, E, &count\_of\_edges);

free(W);

free(E);

}

int main() {

setlocale(LC\_ALL, "Rus");

matrix G\_A = createMatrixFromArray((int[])

{

0, 1, 1, 0, 0, 1, 0, 0, 1,

1, 0, 1, 0, 1, 0, 0, 1, 0,

1, 1, 0, 0, 0, 1, 0, 0, 1,

0, 0, 0, 0, 0, 1, 0, 1, 0,

0, 1, 0, 0, 0, 0, 1, 0, 0,

1, 0, 1, 1, 0, 0, 1, 0, 0,

0, 0, 0, 0, 1, 1, 0, 0, 0,

0, 1, 0, 1, 0, 0, 0, 0, 0,

1, 0, 1, 0, 0, 0, 0, 0, 0

}, 9, 9);

printf("Пример матрицы инцидентности графа, который является эйлеровым, но не гамильтоновым:\n");

matrix G\_I = fromAdjacencyMatrixToIncidenceMatix(&G\_A);

outputMatrixN(G\_I);

printf("\nВсе эйлеровы циклы:\n");

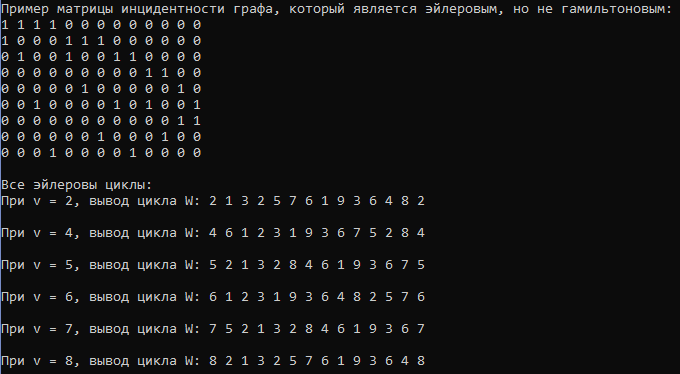
for (int v = 0; v < G\_I.nRows; v++)

findAllEulerCyclesGraph(&G\_I, v);

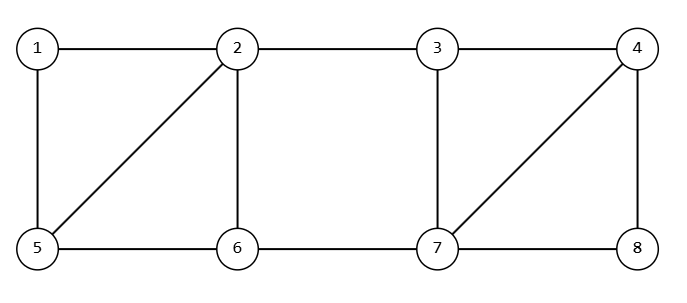
return 0;

}

Вывод программы:



**5. Привести пример диаграммы графа, который является гамильто новым, но не эйлеровым. Найти в нем все гамильтоновы циклы:**



typedef struct matrix {

int\*\* values; // элементы матрицы

int nRows; // количество рядов

int nCols; // количество столбцов

} matrix;

// размещает в динамической памяти матрицу размером nRows на nCols.

matrix getMemMatrix(int nRows, int nCols) {

int\*\* values = (int\*\*)malloc(sizeof(int\*) \* nRows);

for (int i = 0; i < nRows; i++)

values[i] = (int\*)malloc(sizeof(int) \* nCols);

return (matrix) { values, nRows, nCols };

}

// возвращает матрицу размера nRows на nCols, построенную из элементов массива a

matrix createMatrixFromArray(const int\* a, size\_t nRows, size\_t nCols) {

matrix m = getMemMatrix(nRows, nCols);

int k = 0;

for (int i = 0; i < nRows; i++)

for (int j = 0; j < nCols; j++)

m.values[i][j] = a[k++];

return m;

}

// освобождает память, выделенную под хранение матрицы m.

void freeMemMatrix(matrix\* m) {

for (int i = 0; i < m->nRows; i++)

free(m->values[i]);

free(m->values);

m->values = NULL;

m->nRows = 0;

m->nCols = 0;

}

// Функция линейного поиска.

int linearSearch(int\* a, int n, int x) {

for (int i = 0; i < n; i++)

if (a[i] == x)

return i;

return n;

}

// Рекурсивная функция нахождения всех гамильтоновых циклов в матрице смежности графа

void findAllHamiltonCyclesRecursionGraph(matrix\* G, int v, int\* W, int i, int\* V, int\* count\_of\_vertices) {

for (int j = 0; j < G->nCols; j++)

if (G->values[W[i - 1] - 1][j] && (linearSearch(V, G->nCols, j + 1) == G->nCols)) {

W[i] = j + 1;

if ((W[i] == v + 1) && (i == G->nCols)) {

printf("При v = %d, вывод цикла W: ", v + 1);

for (int m = 0; m <= i; m++)

printf("%d ", W[m]);

printf("\n\n");

}

else {

V[\*count\_of\_vertices] = W[i];

\*count\_of\_vertices += 1;

findAllHamiltonCyclesRecursionGraph(G, v, W, i + 1, V, count\_of\_vertices);

}

}

}

// Функция нахождения всех гамильтоновых циклов в матрице смежности графа, через эту функцию подаются данные в рекурсивную функцию нахождения всех гамильтоновых циклов в матрице смежности графа

void findAllHamiltonCyclesGraph(matrix\* G, int v) {

int\* W = (int\*)malloc(sizeof(int) \* (G->nCols + 1));

int\* V = (int\*)malloc(sizeof(int) \* G->nCols);

int count\_of\_vertices = 0;

W[0] = v + 1;

findAllHamiltonCyclesRecursionGraph(G, v, W, 1, V, &count\_of\_vertices);

free(W);

free(V);

}

int main() {

setlocale(LC\_ALL, "Rus");

matrix G\_A = createMatrixFromArray((int[])

{

0, 1, 0, 0, 1, 0, 0, 0,

1, 0, 1, 0, 1, 1, 0, 0,

0, 1, 0, 1, 0, 0, 1, 0,

0, 0, 1, 0, 0, 0, 1, 1,

1, 1, 0, 0, 0, 1, 0, 0,

0, 1, 0, 0, 1, 0, 1, 0,

0, 0, 1, 0, 0, 0, 0, 1,

0, 0, 0, 1, 0, 0, 1, 0

}, 8, 8);

printf("Пример матрицы смежности графа, который является гамильтоновым, но не эйлеровым:\n");

outputMatrixN(G\_A);

printf("\nВсе гамильтоновы циклы:\n");

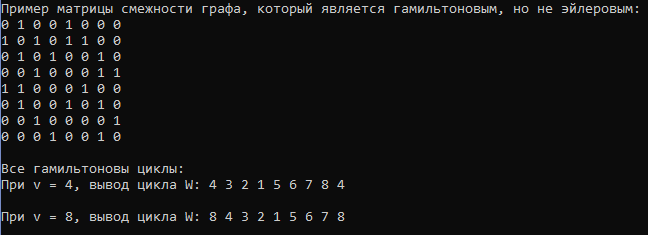
for (int v = 0; v < G\_A.nRows; v++)

findAllHamiltonCyclesGraph(&G\_A, v);

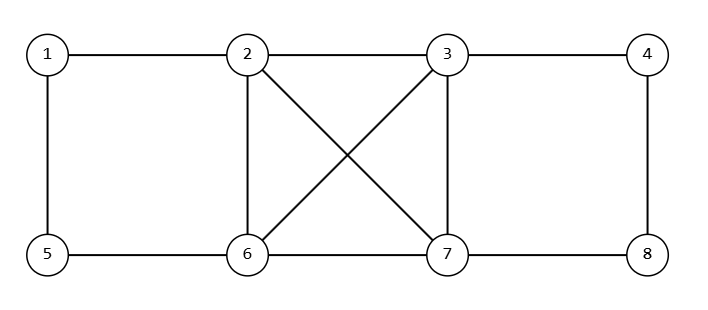
return 0;

}

Вывод программы:



**6. Привести пример диаграммы графа, который является эйлеровым и гамильтоновым. Найти в нем все эйлеровы и гамильтоновы циклы:**



typedef struct matrix {

int\*\* values; // элементы матрицы

int nRows; // количество рядов

int nCols; // количество столбцов

} matrix;

// размещает в динамической памяти матрицу размером nRows на nCols.

matrix getMemMatrix(int nRows, int nCols) {

int\*\* values = (int\*\*)malloc(sizeof(int\*) \* nRows);

for (int i = 0; i < nRows; i++)

values[i] = (int\*)malloc(sizeof(int) \* nCols);

return (matrix) { values, nRows, nCols };

}

// возвращает матрицу размера nRows на nCols, построенную из элементов массива a

matrix createMatrixFromArray(const int\* a, size\_t nRows, size\_t nCols) {

matrix m = getMemMatrix(nRows, nCols);

int k = 0;

for (int i = 0; i < nRows; i++)

for (int j = 0; j < nCols; j++)

m.values[i][j] = a[k++];

return m;

}

// освобождает память, выделенную под хранение матрицы m.

void freeMemMatrix(matrix\* m) {

for (int i = 0; i < m->nRows; i++)

free(m->values[i]);

free(m->values);

m->values = NULL;

m->nRows = 0;

m->nCols = 0;

}

// Функция линейного поиска.

int linearSearch(int\* a, int n, int x) {

for (int i = 0; i < n; i++)

if (a[i] == x)

return i;

return n;

}

// Рекурсивная функция нахождения всех эйлеровых циклов в матрице инциндентности графа

void findAllEulerCyclesRecursionGraph(matrix\* G, int v, int\* W, int i, int\* E, int \*count\_of\_edges) {

for (int j = 0; j < G->nCols; j++)

if (G->values[W[i - 1] - 1][j] && (linearSearch(E, G->nCols, j + 1) == G->nCols)) {

for (int k = 0; k < G->nRows; k++)

if (G->values[k][j] && (k != (W[i - 1] - 1))){

W[i] = k + 1;

break;

}

if ((W[i] == v + 1) && (i == G->nCols)){

printf("При v = %d, вывод цикла W: ", v + 1);

for (int m = 0; m <= i; m++)

printf("%d ", W[m]);

printf("\n\n");

}

else{

E[\*count\_of\_edges] = j + 1;

\*count\_of\_edges += 1;

findAllEulerCyclesRecursionGraph(G, v, W, i + 1, E, count\_of\_edges);

}

}

}

// Функция нахождения всех эйлеровых циклов в матрице инциндентности графа, через эту функцию подаются данные в рекурсивную функцию нахождения всех эйлеровых циклов в матрице инциндентности графа

void findAllEulerCyclesGraph(matrix\* G, int v) {

int\* W = (int\*)malloc(sizeof(int) \* (G->nCols + 1));

int\* E = (int\*)malloc(sizeof(int) \* G->nCols);

int count\_of\_edges = 0;

W[0] = v + 1;

findAllEulerCyclesRecursionGraph(G, v, W, 1, E, &count\_of\_edges);

free(W);

free(E);

}

// Рекурсивная функция нахождения всех гамильтоновых циклов в матрице смежности графа

void findAllHamiltonCyclesRecursionGraph(matrix\* G, int v, int\* W, int i, int\* V, int\* count\_of\_vertices) {

for (int j = 0; j < G->nCols; j++)

if (G->values[W[i - 1] - 1][j] && (linearSearch(V, G->nCols, j + 1) == G->nCols)) {

W[i] = j + 1;

if ((W[i] == v + 1) && (i == G->nCols)) {

printf("При v = %d, вывод цикла W: ", v + 1);

for (int m = 0; m <= i; m++)

printf("%d ", W[m]);

printf("\n\n");

}

else {

V[\*count\_of\_vertices] = W[i];

\*count\_of\_vertices += 1;

findAllHamiltonCyclesRecursionGraph(G, v, W, i + 1, V, count\_of\_vertices);

}

}

}

// Функция нахождения всех гамильтоновых циклов в матрице смежности графа, через эту функцию подаются данные в рекурсивную функцию нахождения всех гамильтоновых циклов в матрице смежности графа

void findAllHamiltonCyclesGraph(matrix\* G, int v) {

int\* W = (int\*)malloc(sizeof(int) \* (G->nCols + 1));

int\* V = (int\*)malloc(sizeof(int) \* G->nCols);

int count\_of\_vertices = 0;

W[0] = v + 1;

findAllHamiltonCyclesRecursionGraph(G, v, W, 1, V, &count\_of\_vertices);

free(W);

free(V);

}

int main() {

setlocale(LC\_ALL, "Rus");

matrix G\_A = createMatrixFromArray((int[])

{

0, 1, 0, 0, 1, 0, 0, 0,

1, 0, 1, 0, 0, 1, 1, 0,

0, 1, 0, 1, 0, 1, 1, 0,

0, 0, 1, 0, 0, 0, 0, 1,

1, 0, 0, 0, 0, 1, 0, 0,

0, 1, 1, 0, 1, 0, 1, 0,

0, 1, 1, 0, 0, 1, 0, 1,

0, 0, 0, 1, 0, 0, 1, 0

}, 8, 8);

printf("Пример матрицы смежности графа, который является эйлеровым и гамильтоновым:\n");

outputMatrixN(G\_A);

printf("\nВсе гамильтоновы циклы:\n");

for (int v = 0; v < G\_A.nRows; v++)

findAllHamiltonCyclesGraph(&G\_A, v);

matrix G\_I = fromAdjacencyMatrixToIncidenceMatix(&G\_A);

printf("\nПример матрицы инцидентности графа, который является эйлеровым и гамильтоновым:\n");

outputMatrixN(G\_I);

printf("\nВсе эйлеровы циклы:\n");

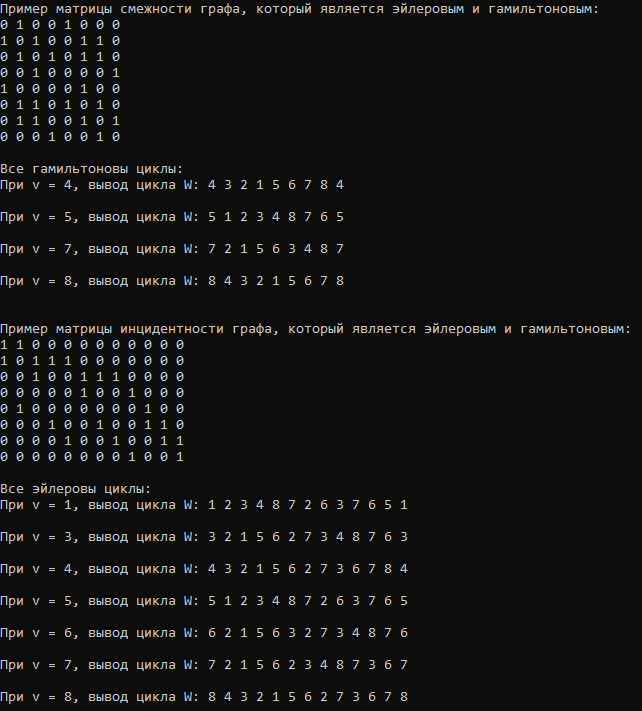
for (int v = 0; v < G\_I.nRows; v++)

findAllEulerCyclesGraph(&G\_I, v);

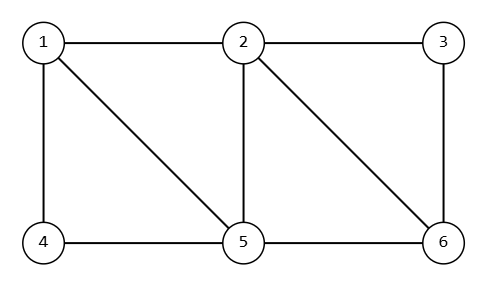
return 0;

}

Вывод программы:



**7. Привести пример диаграммы графа, который не является ни эйлеровым, ни гамильтоновым:**



**Вывод:** изучили разновидности циклов в графах, научились генерировать случайные графы, определять их принадлежность к множеству эйлеровых и гамильтоновых графов, находить все эйлеровы и гамильтоновы циклы в графах.