МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Кафедра «Вычислительная техника»

ОТЧЁТ

По лабораторной работе №5

По курсу «Логика и основы алгоритмизации в инженерных задачах»

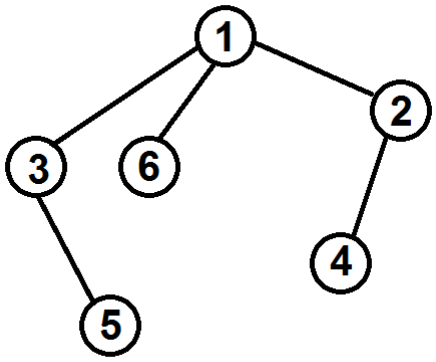
На тему «Определение характеристик графов»

Выполнили
студенты
группы
23ВВВ4:
Святов И.Ю.
Епинин Д.В.

Приняли:
Юрова О.В.
Деев М.В.

Пенза 2024

Общие сведения:
Если *G* – граф (рисунок 1), содержащий непустое множество *n* вершин *V* и множество ребер *E*, где $e(v_i, v_j)$ – ребро между двумя произвольными вершинами v_i и v_j, тогда **размер** графа G есть мощность множества ребер |E(G)| или, количество ребер графа.
Степенью вершины графа *G* называется число инцидентных ей ребер. Степень вершины v_i обозначается через $deg(v_i)$.



$$G = \begin{Bmatrix} 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{Bmatrix}$$

Рисунок 1 – Граф

Вершина v_i со степенью 0 называется **изолированной**, со степенью 1 – **концевой**.
Вершина графа, смежная с каждой другой его вершиной, называется **доминирующей**.

Задание:

1. Сгенерируйте (используя генератор случайных чисел) матрицу смежности для неориентированного графа *G*. Выведите матрицу на экран.

Ответ:

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream>
#include <string.h>
#include <locale.h>

int main() {
    setlocale(LC_ALL, "Russian");
    srand(time(NULL));
    const int G = 5;  // Размер графа
    int M[G][G];      // Матрица смежности

    // Инициализация матрицы смежности
    for (int i = 0; i < G; i++) {
        for (int j = 0; j < G; j++) {
            if (i == j) {
                M[i][j] = 0;  // Нет петли
            }
            else {
                M[i][j] = 0;  // Изначально нет ребер
            }
        }
    }
```

```
    // Генерация случайных рёбер для неориентированного графа
    for (int i = 0; i < G; i++) {
        for (int j = i + 1; j < G; j++) {
            M[i][j] = M[j][i] = rand() % 2;  // Случайное ребро между i и j
        }
    }
    // Вывод матрицы смежности на экран
    printf("Матрица смежности:\n");
    for (int i = 0; i < G; i++) {
        for (int j = 0; j < G; j++) {
            printf("%d ", M[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

2. Определите размер графа *G*, используя матрицу смежности графа.

Ответ:

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream>
#include <string.h>
#include <locale.h>
const int G = 5;  // Размер графа
int M[G][G];      // Матрица смежности
// Функция для определения размера графа
int getGraphSize(int matrix[][G]) {
    int size = 0;
    while (size < G && matrix[size][size] == 0) {
        size++;
    }
    return size;
}
int main() {
    setlocale(LC_ALL, "Russian");
    srand(time(NULL));
    // Инициализация матрицы смежности
    for (int i = 0; i < G; i++) {
        for (int j = 0; j < G; j++) {
            if (i == j) {
                M[i][j] = 0;  // Нет петли
            }
            else {
                M[i][j] = 0;  // Изначально нет рёбер
            }
        }
    }
    // Генерация случайных рёбер для неориентированного графа
    for (int i = 0; i < G; i++) {
        for (int j = i + 1; j < G; j++) {
            M[i][j] = M[j][i] = rand() % 2;  // Случайное ребро между i и j
        }
    }
```

```
    // Вывод матрицы смежности на экран
    printf("Матрица смежности:\n");
    for (int i = 0; i < G; i++) {
        for (int j = 0; j < G; j++) {
            printf("%d ", M[i][j]);
        }
        printf("\n");
    }
    // Определение размера графа
    int size = getGraphSize(M);
    printf("Размер графа: %d\n", size);
    return 0;      }
```

3. Найдите изолированные, концевые и доминирующие вершины.

Ответ:

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream>
#include <string.h>
#include <locale.h>

const int G = 5;  // Размер графа
int M[G][G];      // Матрица смежности
// Функция для определения размера графа
int getGraphSize(int matrix[][G]) {
    int size = 0;
    while (size < G && matrix[size][size] == 0) {
        size++;
    }
    return size;
}
// Функция для нахождения изолированных вершин
void findIsolatedVertices(int matrix[][G]) {
    printf("Изолированные вершины: ");
    for (int i = 0; i < G; i++) {
        int isIsolated = 1;  // Предполагаем, что вершина изолирована
        for (int j = 0; j < G; j++) {
            if (matrix[i][j] == 1) {
                isIsolated = 0;  // Вершина не изолирована
                break;
            }
        }
        if (isIsolated) {
            printf("%d ", i);
        }
    }
    printf("\n");
}
// Функция для нахождения концевых вершин
void findEndVertices(int matrix[][G]) {
    printf("Концевые вершины: ");
    for (int i = 0; i < G; i++) {
        int degree = 0;
        for (int j = 0; j < G; j++) {
            degree += matrix[i][j];  // Считаем степень вершины
```

```
        }
        if (degree == 1) {
            printf("%d ", i);  // Вершина с одной связью является концевой
        }
    }
    printf("\n");
}

// Функция для нахождения доминирующих вершин
void findDominatingVertices(int matrix[][G]) {
    printf("Доминирующие вершины: ");
    for (int i = 0; i < G; i++) {
        int isDominating = 1;  // Предполагаем, что вершина доминирующая
        for (int j = 0; j < G; j++) {
            if (j != i && matrix[i][j] == 0) {
                isDominating = 0;  // Если есть вершина, не связанная с i, то она не
доминирующая
                break;
            }
        }
        if (isDominating) {
            printf("%d ", i);
        }
    }
    printf("\n");
}
int main() {
    setlocale(LC_ALL, "Russian");
    srand(time(NULL));
    // Инициализация матрицы смежности
    for (int i = 0; i < G; i++) {
        for (int j = 0; j < G; j++) {
            if (i == j) {
                M[i][j] = 0;  // Нет петли
            }
            else {
                M[i][j] = 0;  // Изначально нет рёбер
            }
        }
    }
    // Генерация случайных рёбер для неориентированного графа
    for (int i = 0; i < G; i++) {
        for (int j = i + 1; j < G; j++) {
            M[i][j] = M[j][i] = rand() % 2;  // Случайное ребро между i и j
        }
    }
    // Вывод матрицы смежности на экран
    printf("Матрица смежности:\n");
    for (int i = 0; i < G; i++) {
        for (int j = 0; j < G; j++) {
            printf("%d ", M[i][j]);
        }
        printf("\n");
    }
    // Определение размера графа
    int size = getGraphSize(M);
    printf("Размер графа: %d\n", size);
    // Поиск и вывод изолированных, концевых и доминирующих вершин
```

```
    findIsolatedVertices(M);
    findEndVertices(M);
    findDominatingVertices(M);
    return 0;
}
```

Задание 2*

1. Постройте для графа G матрицу инцидентности.

Ответ:

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream>
#include <string.h>
#include <locale.h>

const int G = 5;  // Размер графа
int M[G][G];      // Матрица смежности
// Функция для создания и вывода матрицы инцидентности
void createIncidenceMatrix(int matrix[][G]) {
    // Подсчёт количества рёбер
    int edgeCount = 0;
    for (int i = 0; i < G; i++) {
        for (int j = i + 1; j < G; j++) {
            if (matrix[i][j] == 1) {
                edgeCount++;
            }
        }
    }
    // Динамическое выделение памяти для матрицы инцидентности
    int** incidenceMatrix = (int**)malloc(G * sizeof(int*));
    for (int i = 0; i < G; i++) {
        incidenceMatrix[i] = (int*)malloc(edgeCount * sizeof(int));
    }
    // Инициализация матрицы инцидентности
    for (int i = 0; i < G; i++) {
        for (int j = 0; j < edgeCount; j++) {
            incidenceMatrix[i][j] = 0;  // Изначально все значения 0
        }
    }
    // Заполнение матрицы инцидентности
    int edgeIndex = 0;
    for (int i = 0; i < G; i++) {
        for (int j = i + 1; j < G; j++) {
            if (matrix[i][j] == 1) {
                incidenceMatrix[i][edgeIndex] = 1;  // Вершина i инцидентна ребру
                incidenceMatrix[j][edgeIndex] = 1;  // Вершина j инцидентна ребру
                edgeIndex++;  // Переход к следующему ребру
            }
        }
    }

    // Вывод матрицы инцидентности
    printf("Матрица инцидентности:\n");
    for (int i = 0; i < G; i++) {
        for (int j = 0; j < edgeCount; j++) {
```

```
            printf("%d ", incidenceMatrix[i][j]);
        }
        printf("\n");
    }
    // Освобождение памяти
    for (int i = 0; i < G; i++) {
        free(incidenceMatrix[i]);
    }
    free(incidenceMatrix);
}
int main() {
    setlocale(LC_ALL, "Russian");
    srand(time(NULL));
    // Инициализация матрицы смежности
    for (int i = 0; i < G; i++) {
        for (int j = 0; j < G; j++) {
            if (i == j) {
                M[i][j] = 0;  // Нет петли
            }
            else {
                M[i][j] = 0;  // Изначально нет рёбер
            }
        }
    }
    // Генерация случайных рёбер для неориентированного графа
    for (int i = 0; i < G; i++) {
        for (int j = i + 1; j < G; j++) {
            M[i][j] = M[j][i] = rand() % 2;  // Случайное ребро между i и j
        }
    }
    // Вывод матрицы смежности на экран
    printf("Матрица смежности:\n");
    for (int i = 0; i < G; i++) {
        for (int j = 0; j < G; j++) {
            printf("%d ", M[i][j]);
        }
        printf("\n");
    }
    // Создание и вывод матрицы инцидентности
    createIncidenceMatrix(M);
    return 0;
}
```

2. Определите размер графа G, используя матрицу инцидентности графа.

Ответ:

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream>
#include <string.h>
#include <locale.h>

const int G = 7;  // Размер графа
int M[G][G];      // Матрица смежности
```

```
// Функция для определения размера графа на основе матрицы инцидентности
int getGraphSizeFromIncidenceMatrix(int** incidenceMatrix, int edgeCount) {
    return G;  // В текущем случае размер графа известен заранее как константа G
}

// Функция для создания и вывода матрицы инцидентности
void createIncidenceMatrix(int matrix[][G]) {
    // Подсчёт количества рёбер
    int edgeCount = 0;
    for (int i = 0; i < G; i++) {
        for (int j = i + 1; j < G; j++) {
            if (matrix[i][j] == 1) {
                edgeCount++;
            }
        }
    }
    // Динамическое выделение памяти для матрицы инцидентности
    int** incidenceMatrix = (int**)malloc(G * sizeof(int*));
    for (int i = 0; i < G; i++) {
        incidenceMatrix[i] = (int*)malloc(edgeCount * sizeof(int));
    }

    // Инициализация матрицы инцидентности
    for (int i = 0; i < G; i++) {
        for (int j = 0; j < edgeCount; j++) {
            incidenceMatrix[i][j] = 0;  // Изначально все значения 0
        }
    }

    // Заполнение матрицы инцидентности
    int edgeIndex = 0;
    for (int i = 0; i < G; i++) {
        for (int j = i + 1; j < G; j++) {
            if (matrix[i][j] == 1) {
                incidenceMatrix[i][edgeIndex] = 1;  // Вершина i инцидентна ребру
                incidenceMatrix[j][edgeIndex] = 1;  // Вершина j инцидентна ребру
                edgeIndex++;  // Переход к следующему ребру
            }
        }
    }

    // Вывод матрицы инцидентности
    printf("Матрица инцидентности:\n");
    for (int i = 0; i < G; i++) {
        for (int j = 0; j < edgeCount; j++) {
            printf("%d ", incidenceMatrix[i][j]);
        }
        printf("\n");
    }
    // Определение размера графа на основе матрицы инцидентности
    int     calculatedSize     =     getGraphSizeFromIncidenceMatrix(incidenceMatrix,
edgeCount);
    printf("Размер   графа,   определённый   из   матрицы   инцидентности:   %d\n",
calculatedSize);

    // Освобождение памяти
    for (int i = 0; i < G; i++) {
        free(incidenceMatrix[i]);
```

```
    }
    free(incidenceMatrix);
}

int main() {
    setlocale(LC_ALL, "Russian");
    srand(time(NULL));

    // Инициализация матрицы смежности
    for (int i = 0; i < G; i++) {
        for (int j = 0; j < G; j++) {
            if (i == j) {
                M[i][j] = 0;  // Нет петли
            }
            else {
                M[i][j] = 0;  // Изначально нет рёбер
            }
        }
    }

    // Генерация случайных рёбер для неориентированного графа
    for (int i = 0; i < G; i++) {
        for (int j = i + 1; j < G; j++) {
            M[i][j] = M[j][i] = rand() % 2;  // Случайное ребро между i и j
        }
    }

    // Вывод матрицы смежности на экран
    printf("Матрица смежности:\n");
    for (int i = 0; i < G; i++) {
        for (int j = 0; j < G; j++) {
            printf("%d ", M[i][j]);
        }
        printf("\n");
    }


    // Создание и вывод матрицы инцидентности
    createIncidenceMatrix(M);

    return 0;
}
```

3. Найдите изолированные, концевые и доминирующие вершины.

Ответ:

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream>
#include <string.h>
#include <locale.h>

const int G = 7;  // Размер графа
int M[G][G];      // Матрица смежности
// Функция для создания и вывода матрицы инцидентности
void createIncidenceMatrix(int matrix[][G]) {
```

```
// Подсчёт количества рёбер
int edgeCount = 0;
for (int i = 0; i < G; i++) {
    for (int j = i + 1; j < G; j++) {
        if (matrix[i][j] == 1) {
            edgeCount++;
        }
    }
}
// Динамическое выделение памяти для матрицы инцидентности
int** incidenceMatrix = (int**)malloc(G * sizeof(int*));
for (int i = 0; i < G; i++) {
    incidenceMatrix[i] = (int*)malloc(edgeCount * sizeof(int));
}
// Инициализация матрицы инцидентности
for (int i = 0; i < G; i++) {
    for (int j = 0; j < edgeCount; j++) {
        incidenceMatrix[i][j] = 0;  // Изначально все значения 0
    }
}
// Заполнение матрицы инцидентности
int edgeIndex = 0;
for (int i = 0; i < G; i++) {
    for (int j = i + 1; j < G; j++) {
        if (matrix[i][j] == 1) {
            incidenceMatrix[i][edgeIndex] = 1;  // Вершина i инцидентна ребру
            incidenceMatrix[j][edgeIndex] = 1;  // Вершина j инцидентна ребру
            edgeIndex++;  // Переход к следующему ребру
        }
    }
}
// Вывод матрицы инцидентности
printf("Матрица инцидентности:\n");
for (int i = 0; i < G; i++) {
    for (int j = 0; j < edgeCount; j++) {
        printf("%d ", incidenceMatrix[i][j]);
    }
    printf("\n");
}
// Поиск изолированных, концевых и доминирующих вершин
for (int i = 0; i < G; i++) {
    int edgeCountForVertex = 0;
    for (int j = 0; j < edgeCount; j++) {
        edgeCountForVertex += incidenceMatrix[i][j];
    }
    if (edgeCountForVertex == 0) {
        printf("Вершина %d: Изолированная\n", i);
    }
    else if (edgeCountForVertex == 1) {
        printf("Вершина %d: Концевая\n", i);
    }
    else {
        printf("Вершина %d: Доминирующая\n", i);
    }
}

// Освобождение памяти
for (int i = 0; i < G; i++) {
```

```
        free(incidenceMatrix[i]);
    }
    free(incidenceMatrix);
}

int main() {
    setlocale(LC_ALL, "Russian");
    srand(time(NULL));
    // Инициализация матрицы смежности
    for (int i = 0; i < G; i++) {
        for (int j = 0; j < G; j++) {
            if (i == j) {
                M[i][j] = 0;  // Нет петли
            }
            else {
                M[i][j] = 0;  // Изначально нет рёбер
            }
        }
    }
    // Генерация случайных рёбер для неориентированного графа
    for (int i = 0; i < G; i++) {
        for (int j = i + 1; j < G; j++) {
            M[i][j] = M[j][i] = rand() % 2;  // Случайное ребро между i и j
        }
    }
    // Вывод матрицы смежности на экран
    printf("Матрица смежности:\n");
    for (int i = 0; i < G; i++) {
        for (int j = 0; j < G; j++) {
            printf("%d ", M[i][j]);
        }
        printf("\n");
    }
    // Создание и вывод матрицы инцидентности
    createIncidenceMatrix(M);
    return 0;
}
```

Задание:

Сделать так чтобы матрица смежности была динамическая и степени вершин.

Код:

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream>
#include <string.h>
#include <locale.h>

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <locale.h>

int** dinamic(int size) {
```

```
    int** matrix = (int**)malloc(size * sizeof(int*));
    for (int i = 0; i < size; i++) {
        matrix[i] = (int*)malloc(size * sizeof(int));
    }
    return matrix;
}

void freeMatrix(int** matrix, int size) {
    for (int i = 0; i < size; i++) {
        free(matrix[i]);
    }
    free(matrix);
}

int getGraphSizeFromIncidenceMatrix(int** incidenceMatrix, int edgeCount, int
vertexCount) {
    return vertexCount;  // Размер графа возвращается в качестве количества вершин
}

void createIncidenceMatrix(int** matrix, int vertexCount) {
    int edgeCount = 0;
    for (int i = 0; i < vertexCount; i++) {
        for (int j = i + 1; j < vertexCount; j++) {
            if (matrix[i][j] == 1) {
                edgeCount++;
            }
        }
    }

    int** incidenceMatrix = (int**)malloc(vertexCount * sizeof(int*));
    for (int i = 0; i < vertexCount; i++) {
        incidenceMatrix[i] = (int*)malloc(edgeCount * sizeof(int));
    }

    for (int i = 0; i < vertexCount; i++) {
        for (int j = 0; j < edgeCount; j++) {
            incidenceMatrix[i][j] = 0;
        }
    }

    int edgeIndex = 0;
    for (int i = 0; i < vertexCount; i++) {
        for (int j = i + 1; j < vertexCount; j++) {
            if (matrix[i][j] == 1) {
                incidenceMatrix[i][edgeIndex] = 1;
                incidenceMatrix[j][edgeIndex] = 1;
                edgeIndex++;
            }
        }
    }

    int calculatedSize = getGraphSizeFromIncidenceMatrix(incidenceMatrix, edgeCount,
vertexCount);
    printf("Размер графа, определённый из матрицы инцидентности: %d\n",
calculatedSize);

    for (int i = 0; i < vertexCount; i++) {
        int edgeCountForVertex = 0;
```

```
        for (int j = 0; j < edgeCount; j++) {
            edgeCountForVertex += incidenceMatrix[i][j];
        }
        if (edgeCountForVertex == 0) {
            printf("Вершина %d: Изолированная\n", i);
        }
        else if (edgeCountForVertex == 1) {
            printf("Вершина %d: Концевая\n", i);
        }
        else {
            printf("Вершина %d: Доминирующая\n", i);
        }
    }

    printf("Матрица инцидентности:\n");
    for (int i = 0; i < vertexCount; i++) {
        for (int j = 0; j < edgeCount; j++) {
            printf("%d ", incidenceMatrix[i][j]);
        }
        printf("\n");
    }

    freeMatrix(incidenceMatrix, vertexCount);
}

int main() {
    setlocale(LC_ALL, "Russian");
    srand(time(NULL));

    int G;
    printf("Введите количество вершин графа: ");
    scanf("%d", &G);

    // Динамическое выделение памяти для матрицы смежности
    int** M = dinamic(G);

    // Инициализация матрицы смежности
    for (int i = 0; i < G; i++) {
        for (int j = 0; j < G; j++) {
            if (i == j) {
                M[i][j] = 0;
            }
            else {
                M[i][j] = 0; // Или можно не инициализировать, если будете сразу
задавать rand()
            }
        }
    }

    int* degrees = (int*)malloc(G * sizeof(int));
    memset(degrees, 0, G * sizeof(int)); // Инициализация массива степеней нулями

    for (int i = 0; i < G; i++) {
        for (int j = i + 1; j < G; j++) {
            M[i][j] = M[j][i] = rand() % 2; // Генерация ребра

            if (M[i][j] == 1) {
                degrees[i]++;
```

```
                degrees[j]++;
            }
        }
    }

    printf("Матрица смежности:\n");
    for (int i = 0; i < G; i++) {
        for (int j = 0; j < G; j++) {
            printf("%d ", M[i][j]);
        }
        printf(" Степень: %d", degrees[i]); // Печать степени каждой вершины
        printf("\n");
    }

    createIncidenceMatrix(M, G);
    freeMatrix(M, G);
    free(degrees); // Освобождение памяти от массива степеней

    return 0;
}
```

Вывод: В ходе лабораторной работы была успешно реализована генерация случайной матрицы смежности и матрицы инцидентности, выяснили характеристики вершин графа, такие как изолированные, концевые и доминирующие. Через матрицы смежности и инцидентности, представления графа показывает что размер графа одинаковый, что подтверждает корректность работы программы.