Министерство науки и высшего образования Российской Федерации
Пензенский государственный университет
Кафедра «Вычислительная техника»

ОТЧЕТ

по лабораторной работе №5

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Определение характеристик графов»

Выполнили:
студенты группы 24ВВВ3
Азаров М.С.
Кукушкин А.А.

Приняли:
к.т.н., доцент Юрова О.В.
к.т.н., Деев М.В.

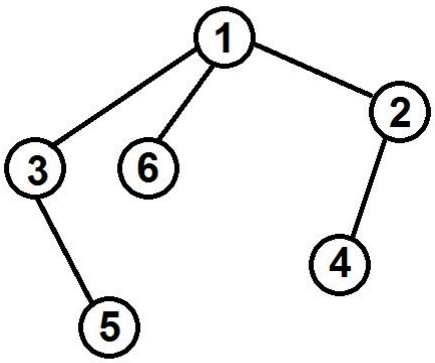
Пенза 2025

Цель работы – Освоение методов программной обработки графовых структур путем реализации алгоритмов преобразования матричных представлений графов и анализа их структурных характеристик.

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**.

Лабораторное задание:

Задание 1

1. Сгенерируйте (используя генератор случайных чисел) матрицу смежности для неориентированного графа *G*. Выведите матрицу на экран.
2. Определите размер графа *G*, используя матрицу смежности графа.
3. Найдите изолированные, концевые и доминирующие вершины.

Задание 2*

1. Постройте для графа G матрицу инцидентности.

---

2. Определите размер графа *G*, используя матрицу инцидентности графа.
3. Найдите изолированные, концевые и доминирующие вершины.

Код программы

C

Задание 1

```c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void generate_adjacency_matrix(int n, int** matrix) {
    for (int i = 0; i < n; i++) {
        for (int j = i; j < n; j++) {
            int r = rand() % 2;
            matrix[i][j] = matrix[j][i] = r;

        }
    }
}
```

```c
void print_matrix(int n, int** matrix) {
    printf("Matr smezh:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }
}

int graph_size(int n, int** matrix) {
    int edges = 0;
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            if (matrix[i][j]) edges++;
        }
    }
    for (int i = 0; i < n; i++) {
        if (matrix[i][i]) edges++;
    }
    return edges;
}


void analyze_vertices(int n, int** matrix) {
    printf("Isolated vershini: ");
    int found = 0;
    for (int i = 0; i < n; i++) {
        int degree = 0;
        for (int j = 0; j < n; j++) {
            if (i == j)
                degree += 2 * matrix[i][j];
            else
                degree += matrix[i][j];
        }
        if (degree == 0) {
            printf("%d ", i + 1);
            found = 1;
        }
    }
    if (!found) printf("no");
    printf("\n");

    printf("Konechnie vershini: ");
    found = 0;
    for (int i = 0; i < n; i++) {
        int degree = 0;
        for (int j = 0; j < n; j++) {
            if (i == j)
                degree += 2 * matrix[i][j];
            else
                degree += matrix[i][j];
```

```
        }
        if (degree == 1) {
            printf("%d ", i + 1);
            found = 1;
        }
    }
    if (!found) printf("net");
    printf("\n");

    printf("Dominating vershini: ");
    found = 0;
    for (int i = 0; i < n; i++) {
        int degree = 0;
        for (int j = 0; j < n; j++) {
            if (i == j)
                degree += 2 * matrix[i][j];
            else
                degree += matrix[i][j];
        }
        if (degree == n - 1) {
            printf("%d ", i + 1);
            found = 1;
        }
    }
    if (!found) printf("no");
    printf("\n");
}


int main() {
    int n;
    printf("Enter kolvo vershin of graph: ");
    scanf("%d", &n);

    int** matrix = malloc(n * sizeof(int*));
    for (int i = 0; i < n; i++)
        matrix[i] = malloc(n * sizeof(int));

    srand(time(NULL));
    generate_adjacency_matrix(n, matrix);
    print_matrix(n, matrix);
    printf("Size of graph (chislo reber): %d\n", graph_size(n, matrix));
    analyze_vertices(n, matrix);

    for (int i = 0; i < n; i++) free(matrix[i]);
    free(matrix);

    return 0;
}
```

Результаты работы программы

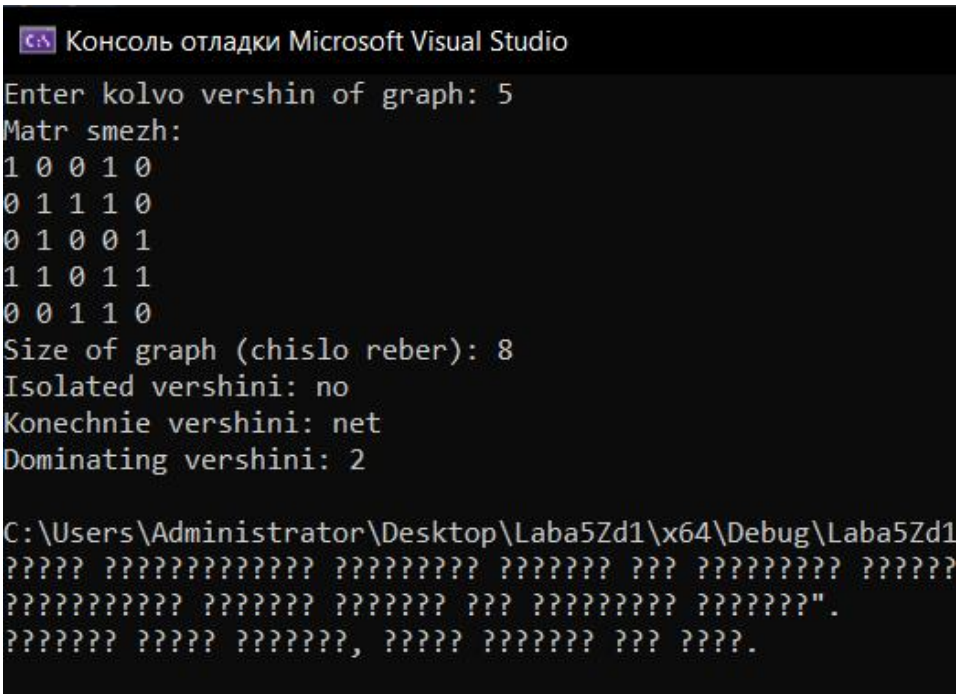


Рисунок 1 - Результат работы программы

Задание 2

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void generate_adjacency_matrix(int n, int** matrix) {
    for (int i = 0; i < n; i++)
        for (int j = i; j < n; j++)
            matrix[i][j] = matrix[j][i] = (i != j) ? rand() % 2 : 0;
}

int build_edge_list(int n, int** matrix, int (*edges)[2]) {
    int k = 0;
    for (int i = 0; i < n; i++)
        for (int j = i + 1; j < n; j++)
            if (matrix[i][j])
                edges[k][0] = i, edges[k++][1] = j;
    return k;
}

void build_incidence_matrix(int n, int m, int (*edges)[2], int** inc_matrix)
{
    for (int i = 0; i < n; i++)
```

```
        for (int j = 0; j < m; j++)
            inc_matrix[i][j] = (edges[j][0] == i || edges[j][1] == i) ? 1 : 0;
}

void print_incidence_matrix(int n, int m, int** inc_matrix) {
    printf("Intsidentnosti matrix:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++)
            printf("%d ", inc_matrix[i][j]);
        printf("\n");
    }
}

int graph_size_by_inc(int m) {
    return m;
}

void analyze_vertices_inc(int n, int m, int** inc_matrix) {
    printf("Isolated vershini: ");
    int found = 0;
    for (int i = 0; i < n; i++) {
        int deg = 0;
        for (int j = 0; j < m; j++) deg += inc_matrix[i][j];
        if (deg == 0) { printf("%d ", i + 1); found = 1; }
    }
    if (!found) printf("no");
    printf("\n");

    printf("Konechnie vershini: ");
    found = 0;
    for (int i = 0; i < n; i++) {
        int deg = 0;
        for (int j = 0; j < m; j++) deg += inc_matrix[i][j];
        if (deg == 1) { printf("%d ", i + 1); found = 1; }
    }
    if (!found) printf("net");
    printf("\n");

    printf("Dominating vershini: ");
    found = 0;
    for (int i = 0; i < n; i++) {
        int deg = 0;
        for (int j = 0; j < m; j++) deg += inc_matrix[i][j];
        if (deg == n - 1) { printf("%d ", i + 1); found = 1; }
    }
    if (!found) printf("no");
    printf("\n");
}

int main() {
    int n;
    printf("Enter kolvo vershin of graph: ");
```

```
    scanf("%d", &n);

    int** matrix = malloc(n * sizeof(int*));
    for (int i = 0; i < n; i++)
        matrix[i] = calloc(n, sizeof(int));

    srand(time(NULL));
    generate_adjacency_matrix(n, matrix);

    int max_edges = n * (n - 1) / 2;
    int (*edges)[2] = malloc(max_edges * sizeof(int[2]));
    int m = build_edge_list(n, matrix, edges);

    int** inc_matrix = malloc(n * sizeof(int*));
    for (int i = 0; i < n; i++)
        inc_matrix[i] = calloc(m, sizeof(int));
    build_incidence_matrix(n, m, edges, inc_matrix);

    print_incidence_matrix(n, m, inc_matrix);
    printf("Size of graph (chislo reber): %d\n", graph_size_by_inc(m));
    analyze_vertices_inc(n, m, inc_matrix);

    for (int i = 0; i < n; i++) { free(matrix[i]); free(inc_matrix[i]); }
    free(matrix); free(inc_matrix); free(edges);

    return 0;
}
```

Результаты работы программы

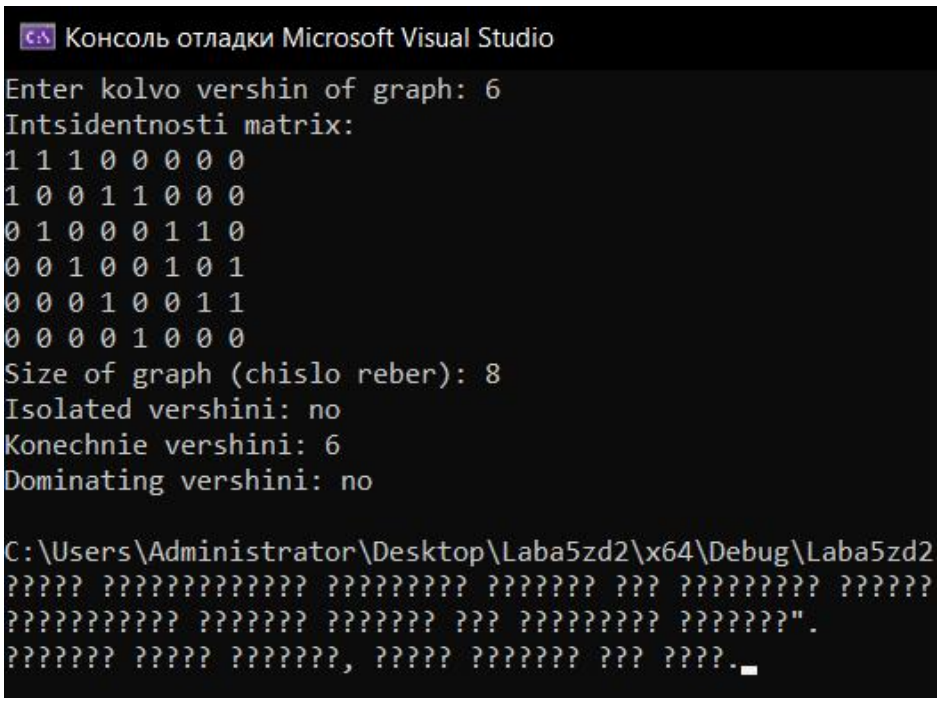


Рисунок 2 - Результат работы программы

Вывод: В результате выполнения работы были успешно реализованы алгоритмы преобразования матрицы смежности в матрицу инцидентности и разработаны методы анализа вершин графа.