Министерство науки и высшего образования
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 22ВВП2:
Гавин В.Н.
Дулатов Д.А.
Приняли
Акифьев И.В.
Юрова О.В.

Пенза 2023

Задание 1

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

Задание 2*

- Постройте для графа G матрицу инцидентности.
- Определите размер графа G, используя матрицу инцидентности графа.
- Найдите изолированные, концевые и доминирующие вершины.

Решение заданий

Задание 1:

Программа выводит размер графа, определенный на основе этого размера матрицы.

```
int graphSize = 0;

for (int i = 0; i < n; i++) {
  for (int j = 0; j < n; j++) {
    if (i > j && adjacencyMatrix[i][j] == 1) {
      graphSize++;
    }
  }
}
```

Добавили блоки для нахождения изолированных, концевых и доминирующих вершин в графе после вывода матрицы смежности. Они анализирует каждую вершину в графе и определяют, является ли она изолированной, концевой или доминирующей, и затем выводят соответствующие вершины на экран.

```
            // Найдем изолированные, концевые и доминирующие
вершины
```

```
        printf("Изолированные вершины:");
        for (int i = 0; i < n; i++) {
              int isolated = 1; // Предполагаем, что вершина
изолирована
            for (int j = 0; j < n; j++) {
                if (adjacencyMatrix[i][j] == 1) {
                    isolated = 0; // Вершина не изолирована
                    break;
                }
            }
            if (isolated) {
                printf(" %d", i + 1);
            }
        }
        printf("\n");

        printf("Концевые вершины:");
        for (int i = 0; i < n; i++) {
            int degree = 0;
            for (int j = 0; j < n; j++) {
                degree += adjacencyMatrix[i][j];
            }
            if (degree == 1) {
                printf(" %d", i + 1);
            }
        }
        printf("\n");

        printf("Доминирующие вершины:");
        for (int i = 0; i < n; i++) {
             int dominating = 1; // Предполагаем, что вершина
доминирующая
            for (int j = 0; j < n; j++) {
                if (i != j && adjacencyMatrix[i][j] != 1) {
                              dominating = 0; // Вершина не
доминирующая
                    break;
                }
            }
            if (dominating) {
                printf(" %d", i + 1);
            }
        }
        printf("\n");
```

Задание 2:

Для построения матрицы инцидентности для графа G, мы внесли следующие изменения в код программы:

```
    // Создаем матрицу инцидентности
```

```
        int m = 0; // Количество рёбер
        for (int i = 0; i < n; i++) {
            for (int j = i + 1; j < n; j++) {
                if (adjacencyMatrix[i][j] == 1) {
                    m++;
                }
            }
        }

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

         // Инициализация матрицы инцидентности
         for (int i = 0; i < n; i++) {
              for (int j = 0; j < m; j++) {
                   incidenceMatrix[i][j] = 0;
              }
         }

        // Заполняем матрицу инцидентности
        int edgeIndex = 0;
        for (int i = 0; i < n; i++) {
            for (int j = i + 1; j < n; j++) {
                if (adjacencyMatrix[i][j] == 1) {
                    incidenceMatrix[i][edgeIndex] = 1;
                    incidenceMatrix[j][edgeIndex] = 1;
                    edgeIndex++;
                }
            }
        }

        // Выводим матрицу инцидентности на экран
        printf("Матрица инцидентности для графа G:\n");
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                printf("%d ", incidenceMatrix[i][j]);
            }
            printf("\n");
        }
```

Для определения размера графа G, используя матрицу инцидентности, мы внесли следующие изменения в код программы:

В матрице инцидентности графа столбцы соответствуют ребрам графа. Поэтому по количеству столбцов в матрице инцидентности можно определить количество ребер в графе.

```
printf("Размер графа G =  %d\n", m);
```

Для нахождения изолированных, концевых и доминирующих вершин по матрице инцидентности, мы внесли следующие изменения в код программы:

```
                // Найдем изолированные вершины по матрице инцидентности
        printf("Изолированные вершины:");
        for (int i = 0; i < n; i++) {
              int isolated = 1; // Предполагаем, что вершина изолирована
            for (int j = 0; j < m; j++) {
                if (incidenceMatrix[i][j] == 1) {
                    isolated = 0; // Вершина не изолирована
                    break;
                }
            }
            if (isolated) {
                printf(" %d", i + 1);
            }
        }
        printf("\n");

        // Найдем концевые вершины по матрице инцидентности
        printf("Концевые вершины:");
        for (int i = 0; i < n; i++) {
            int degree = 0;
            for (int j = 0; j < m; j++) {
                if (incidenceMatrix[i][j] == 1) {
                    degree++;
                }
            }
            if (degree == 1) {
                printf(" %d", i + 1);
            }
        }
        printf("\n");

              // Найдем доминирующие вершины по матрице инцидентности
    printf("Доминирующие вершины:");
```

```
    for (int i = 0; i < n; i++) {
        int degree = 0; // Степень вершины
        for (int j = 0; j < m; j++) {
            if (incidenceMatrix[i][j] == 1) {
                degree++;
            }
        }

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

Листинг

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

// Функция для генерации случайных чисел от 0 до 1 (ребро или нет)
int randomEdge() {
return rand() % 2;
}

int main() {
setlocale(LC_ALL, "Rus");
int n; // Размер матрицы
srand(time(NULL)); // Инициализация генератора случайных чисел

printf("Введите размер матрицы (количество вершин): ");
scanf("%d", &n);

// Выделение памяти для матрицы смежности
int **adjacencyMatrix = (int **)malloc(n * sizeof(int *));
for (int i = 0; i < n; i++) {
     adjacencyMatrix[i] = (int *)malloc(n * sizeof(int));
}

// Заполняем матрицу смежности случайными значениями
for (int i = 0; i < n; i++) {
     for (int j = 0; j < n; j++) {
```

```
                if (i == j) {
                    adjacencyMatrix[i][j] = 0; // Нет петель
                }
                else {
                    adjacencyMatrix[i][j] = randomEdge();
                    adjacencyMatrix[j][i] =
adjacencyMatrix[i][j]; // Граф неориентированный, поэтому
зеркально заполняем
                }
            }
        }

        // Выводим матрицу смежности на экран
        printf("Матрица смежности для графа G:\n");
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                printf("%d ", adjacencyMatrix[i][j]);
            }
            printf("\n");
        }

        // Определение размера графа на основе матрицы смежности
        int m = 0; // Количество рёбер
        for (int i = 0; i < n; i++) {
            for (int j = i + 1; j < n; j++) {
                if (adjacencyMatrix[i][j] == 1) {
                    m++;
                }
            }
        }

        printf("Размер графа G =  %d\n", m);

        // Найдем изолированные, концевые и доминирующие вершины
        printf("Изолированные вершины:");
        for (int i = 0; i < n; i++) {
            int isolated = 1; // Предполагаем, что вершина
изолирована
            for (int j = 0; j < n; j++) {
                if (adjacencyMatrix[i][j] == 1) {
                    isolated = 0; // Вершина не изолирована
                    break;
                }
            }
```

```
            if (isolated) {
                printf(" %d", i + 1);
            }
        }
        printf("\n");

        printf("Концевые вершины:");
        for (int i = 0; i < n; i++) {
            int degree = 0;
            for (int j = 0; j < n; j++) {
                degree += adjacencyMatrix[i][j];
            }
            if (degree == 1) {
                printf(" %d", i + 1);
            }
        }
        printf("\n");

        printf("Доминирующие вершины:");
        for (int i = 0; i < n; i++) {
            int dominating = 1; // Предполагаем, что вершина
доминирующая
            for (int j = 0; j < n; j++) {
                if (i != j && adjacencyMatrix[i][j] != 1) {
                    dominating = 0; // Вершина не доминирующая
                    break;
                }
            }
            if (dominating) {
                printf(" %d", i + 1);
            }
        }
        printf("\n");

        // Создаем матрицу инцидентности
        int **incidenceMatrix = (int **)malloc(n * sizeof(int
*));
        for (int i = 0; i < n; i++) {
            incidenceMatrix[i] = (int *)malloc(m * sizeof(int));
        }

        // Инициализация матрицы инцидентности
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
```

```
                incidenceMatrix[i][j] = 0;
            }
        }

        // Заполняем матрицу инцидентности
        int edgeIndex = 0;
        for (int i = 0; i < n; i++) {
            for (int j = i + 1; j < n; j++) {
                if (adjacencyMatrix[i][j] == 1) {
                    incidenceMatrix[i][edgeIndex] = 1;
                    incidenceMatrix[j][edgeIndex] = 1;
                    edgeIndex++;
                }
            }
        }

        // Выводим матрицу инцидентности на экран
        printf("Матрица инцидентности для графа G:\n");
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                printf("%d ", incidenceMatrix[i][j]);
            }
            printf("\n");
        }

        printf("Размер графа G =  %d\n", m);

        // Найдем изолированные вершины по матрице инцидентности
        printf("Изолированные вершины:");
        for (int i = 0; i < n; i++) {
            int isolated = 1; // Предполагаем, что вершина
изолирована
            for (int j = 0; j < m; j++) {
                if (incidenceMatrix[i][j] == 1) {
                    isolated = 0; // Вершина не изолирована
                    break;
                }
            }
            if (isolated) {
                printf(" %d", i + 1);
            }
        }
        printf("\n");
```

```
// Найдем концевые вершины по матрице инцидентности
printf("Концевые вершины:");
for (int i = 0; i < n; i++) {
     int degree = 0;
     for (int j = 0; j < m; j++) {
          if (incidenceMatrix[i][j] == 1) {
               degree++;
          }
     }
     if (degree == 1) {
          printf(" %d", i + 1);
     }
}
printf("\n");

printf("Доминирующие вершины:");
for (int i = 0; i < n; i++) {
     int degree = 0; // Степень вершины
     for (int j = 0; j < m; j++) {
          if (incidenceMatrix[i][j] == 1) {
               degree++;
          }
     }

     if (degree == n - 1) {
          printf(" %d", i + 1);
     }
}
printf("\n");


// Освобождаем выделенную память
for (int i = 0; i < n; i++) {
     free(adjacencyMatrix[i]);
}
free(adjacencyMatrix);

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

return 0;
}
```

Результаты работы программы

```
Введите размер матрицы (количество вершин): 3
Матрица смежности для графа G:
0 1 0
1 0 1
0 1 0
Размер графа G =  2
Изолированные вершины:
Концевые вершины: 1 3
Доминирующие вершины: 2
Матрица инцидентности для графа G:
1 0
1 1
0 1
Размер графа G =  2
Изолированные вершины:
Концевые вершины: 1 3
Доминирующие вершины: 2
```

Вывод

В ходе выполнения заданий были рассмотрены два способа представления графов: матрицей смежности и матрицей инцидентности.

С помощью этих матриц были выполнены следующие задачи:

- Определение размера графа
- Нахождение изолированных вершин
- Нахождение концевых вершин
- Нахождение доминирующих вершин

Выполнение заданий позволило исследовать различные характеристики графов с использованием двух разных матриц.