МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
Институт математики и информационных систем
Факультет автоматики и вычислительной техники
Кафедра электронных вычислительных машин

Дата сдачи на проверку:

«__»_____ 2025 г.

Проверено:

«__»_____ 2025 г.

ГРАФЫ. ПОИСК ПУТЕЙ В ГРАФАХ.

Отчёт по лабораторной работе №6

по дисциплине

«Дискретная математика»

Разработал студент гр. ИВТб-1301-05-00 _____/Черкасов А. А./

(подпись)

Проверила преподаватель _____/Пахарева И. В./

(подпись)

Работа защищена «__»_____ 2025 г.

Киров

2025

Цель

Цель работы: изучить методы представления ориентированного графа в виде матрицы инцидентности и освоить приёмы полного перебора путей с помощью поиска в глубину. На их основе разработать программу, способную находить все ориентированные цепи, ведущие в заданную вершину, без повторения дуг.

Задание

– Ориентированный граф G задаётся матрицей инцидентности, записанной в файле `input.txt`. Ограничить размерность: число вершин и число дуг (ребер) ≥ 5.

– Найти возможные ориентированные цепи (дуги в пути не должны повторяться) в вершину, номер которой вводится с клавиатуры.

Решение

Схема алгоритма решения представлена на рисунках 1.1 и 1.2. Примеры работы программы представлены на рисунках 2.1 и 2.2. Исходный код представлен в приложении А1.

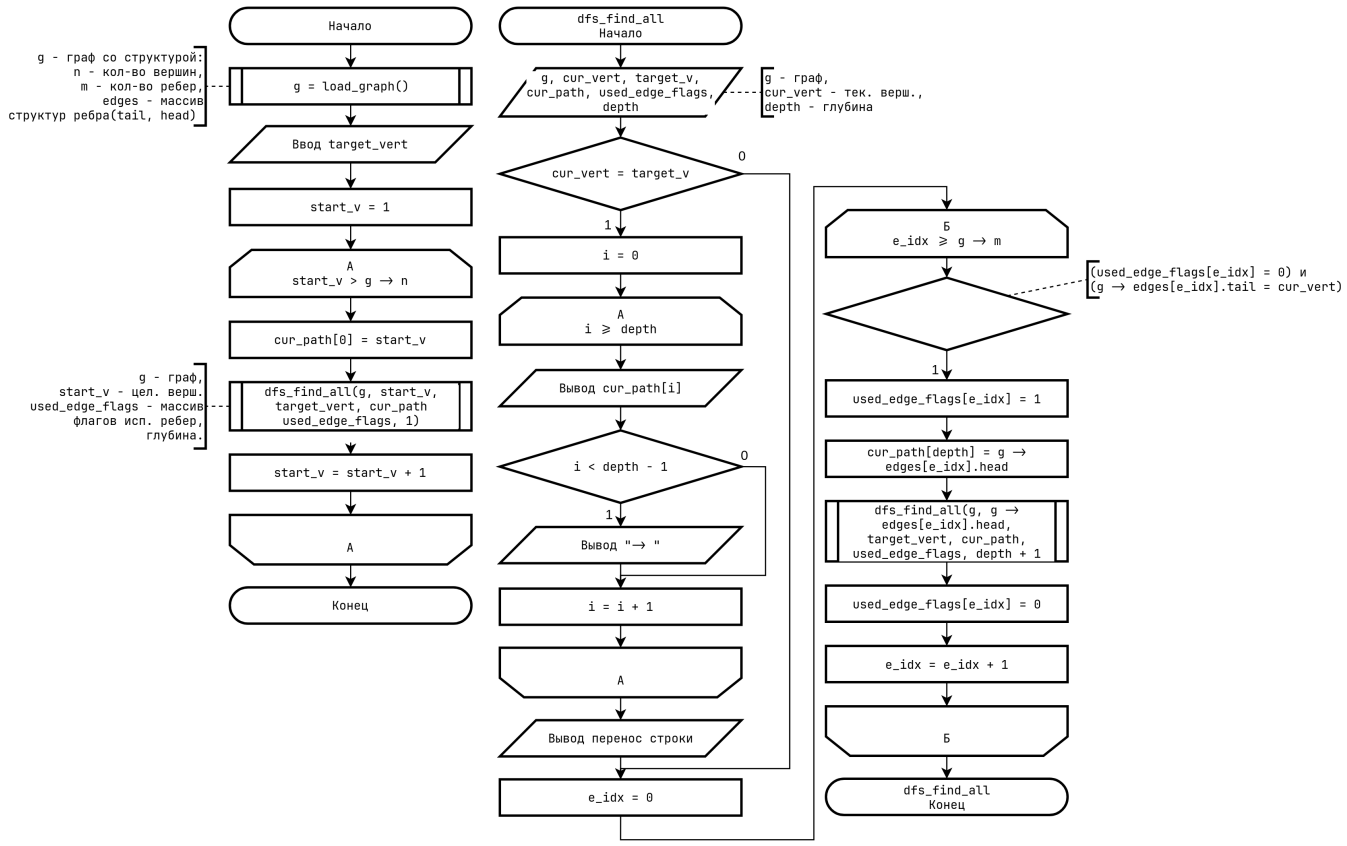

Рисунок 1.1 - Схема алгоритма основной программы и подпрограммы поиска в глубину.

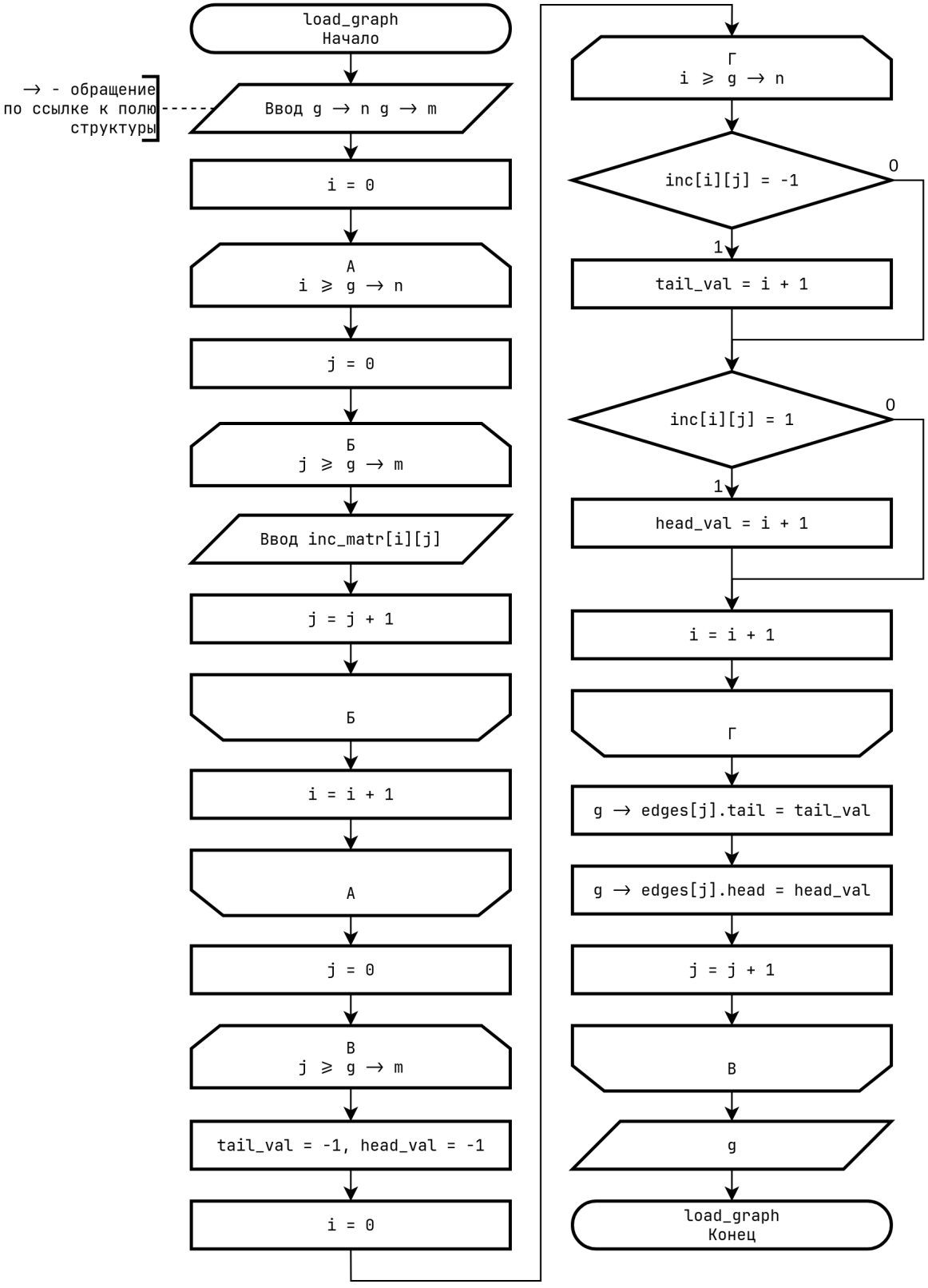

Рисунок 1.2 - Схема алгоритма подпрограммы загрузки графа из матрицы в файле.

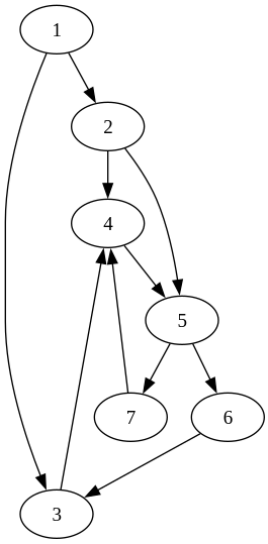

Рисунок 2.1 - Граф.

```
❯ make run
[DEBUG] Linux сборка завершена: target/debug/graph_paths
[RUN] Запуск target/debug/graph_paths
Матрица инцидентности:
 -1  -1   0   0   0   0   0   0   0   0
  1   0  -1   0   0  -1   0   0   0   0
  0   1   0  -1   0   0   0   1   0   0
  0   0   1   1  -1   0   0   0   0   1
  0   0   0   0   1   1  -1   0  -1   0
  0   0   0   0   0   0   1  -1   0   0
  0   0   0   0   0   0   0   0   1  -1
DOT-файл успешно создан: graph.dot
Введите номер целевой вершины (1..7): 6

Пути до целевой вершины 6:
1 → 2 → 4 → 5 → 6
1 → 2 → 5 → 6
1 → 2 → 5 → 7 → 4 → 5 → 6
1 → 3 → 4 → 5 → 6
2 → 4 → 5 → 6
2 → 5 → 6
2 → 5 → 7 → 4 → 5 → 6
3 → 4 → 5 → 6
4 → 5 → 6
5 → 6
5 → 7 → 4 → 5 → 6
6
6 → 3 → 4 → 5 → 6
7 → 4 → 5 → 6
```

Рисунок 2.2 - Пример работы программы.

Вывод

В результате выполнения лабораторной работы была создана и протестирована программа на языке C, которая:

- считывает из файла `input.txt` матрицу инцидентности ориентированного графа размером не менее 5 вершин и дуг;
- запрашивает у пользователя номер целевой вершины и проверяет корректность ввода;
- выполняет поиск в глубину (DFS) с учётом запрета на повторное использование дуг для полного перебора всех ориентированных цепей, заканчивающихся в заданной вершине;
- формирует и выводит на экран список найденных путей в удобочитаемом виде.

Приложение A1. Исходный код

```
#include <stdio.h>
#include <stdlib.h>

typedef struct {
  int tail;
  int head;
} Edge;

typedef struct {
  int n;
  int m;
  int **inc;
  Edge *edges;
} Graph;

int target;
int *used_edge;
int *path;

void dfs(Graph *g, int v, int depth) {
  if (v == target) {
    for (int i = 0; i < depth; i++) {
      printf("%d", path[i]);
      if (i < depth - 1)
        printf(" -> ");
    }
    printf("\n");
  }
  for (int e = 0; e < g->m; e++) {
    if (!used_edge[e] && g->edges[e].tail == v) {
      used_edge[e] = 1;
      path[depth] = g->edges[e].head;
```

```
      dfs(g, g->edges[e].head, depth + 1);
      used_edge[e] = 0;
    }
  }
}

void generate_dot_file(Graph *g, const char *filename) {
  FILE *f = fopen(filename, "w");
  if (!f) {
    perror("Не удалось создать dot-файл");
    return;
  }

  fprintf(f, "digraph G {\n");

  for (int i = 1; i <= g->n; i++) {
    fprintf(f, "  %d;\n", i);
  }

  for (int i = 0; i < g->m; i++) {
    fprintf(f, "  %d -> %d;\n", g->edges[i].tail, g->edges[i].head);
  }

  fprintf(f, "}\n");
  fclose(f);

  printf("DOT-файл успешно создан: %s\n", filename);
}

Graph *load_graph(const char *filename) {
  FILE *f = fopen(filename, "r");
  if (!f) {
    perror("Не удалось открыть input.txt");
    exit(EXIT_FAILURE);
```

```
}

Graph *g = malloc(sizeof(Graph));
if (fscanf(f, "%d %d", &g->n, &g->m) != 2) {
  fprintf(stderr, "Некорректный формат: ожидаются числа n и m\n");
  exit(EXIT_FAILURE);
}

printf("Матрица инцидентности:\n");

g->inc = malloc(g->n * sizeof(int *));
for (int i = 0; i < g->n; i++) {
  g->inc[i] = malloc(g->m * sizeof(int));
  for (int j = 0; j < g->m; j++) {
    if (fscanf(f, "%d", &g->inc[i][j]) != 1) {
      fprintf(stderr, "Некорректный формат: ожидается число в матрице\n");
      exit(EXIT_FAILURE);
    }
    printf("%3d ", g->inc[i][j]);
  }
  printf("\n");
}
fclose(f);

g->edges = malloc(g->m * sizeof(Edge));
for (int j = 0; j < g->m; j++) {
  int tail = -1, head = -1;
  for (int i = 0; i < g->n; i++) {
    if (g->inc[i][j] == -1)
      tail = i + 1;
    if (g->inc[i][j] == 1)
      head = i + 1;
  }
  if (tail < 1 || head < 1) {
```

```
      fprintf(stderr, "Ошибка: некорректное определение дуги в столбце %d\n",
              j + 1);
      exit(EXIT_FAILURE);
    }
    g->edges[j].tail = tail;
    g->edges[j].head = head;
  }

  return g;
}

void free_graph(Graph *g) {
  for (int i = 0; i < g->n; i++)
    free(g->inc[i]);
  free(g->inc);
  free(g->edges);
  free(g);
}

int main() {
  Graph *g = load_graph("input.txt");

  generate_dot_file(g, "graph.dot");

  printf("Введите номер целевой вершины (1..%d): ", g->n);
  if (scanf("%d", &target) != 1 || target < 1 || target > g->n) {
    fprintf(stderr, "Неверный номер вершины\n");
    free_graph(g);
    return EXIT_FAILURE;
  }

  used_edge = calloc(g->m, sizeof(int));
  path = malloc((g->m + 1) * sizeof(int));
```

```
  printf("\nПути до целевой вершины %d:\n", target);
  for (int v = 1; v <= g->n; v++) {
    path[0] = v;
    dfs(g, v, 1);
  }

  free(used_edge);
  free(path);
  free_graph(g);
  return EXIT_SUCCESS;
}
```