Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Южно-Уральский государственный университет (НИУ)»

Высшая школа электроники и компьютерных наук

Кафедра системного программирования

ОТЧЕТ

о выполнении дополнительного практического задания №3

по дисциплине

«Структуры и алгоритмы обработки данных»

Вариант 5

Проверил:

ст. преподаватель кафедры СП

Петрова Л.Н.

Выполнил:

Студент группы КЭз-391

Галиулин Р.Р.

Челябинск
2025

Содержание

1. Описание задачи

Задача: Рекурсия с возвратом / Ход конём

Найти маршрут обхода конем шахматной доски, заданных размеров, из заданного начального положения коня.

Входные данные (файл input.txt):

M, N - размеры шахматной доски. X, Y - начальные координаты расположения коня.

Выходные данные (файл output.txt): Напечатать номер хода в каждой ячейки поля, если маршрут существует и "Маршрут не существует"в противном случае.

Пример:

Входные данные: M=10, N=10, X=1, Y=1.

Выходные данные:

28	63	12	45	26	61	10	43	24	59
13	46	27	62	11	44	25	60	9	42
64	29	94	77	86	69	92	79	58	23
47	14	85	68	93	78	87	70	41	8
30	65	76	95	84	91	80	97	22	57
15	48	67	100	75	96	71	88	7	40
66	31	50	83	90	73	98	81	56	21
49	16	33	74	99	82	89	72	39	6
32	51	2	35	18	53	4	37	20	55
1	34	17	52	3	36	19	54	5	38

Для решения задачи был применён эвристический метод Варнсдорфа - поиска пути с минимальным количеством дальнейших шагов и избегания потенциально тупиковых ситуации, вроде угловых клеток.

2. Листинг программы

Язык программирования: C++ 14. Среда разработки: Ubuntu 24.10, gcc 14.2.0, nvim

Листинг 1: Задача: Ход конем

```cpp
// Галиулин РР.. КЭз-391
// Структуры и алгоритмы обработки данных
// Дополнительное практическое занятие №3

// Найти маршрут обхода конем шахматной доски, заданных размеров, из заданного
// начального положения коня.

#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <string>
#include <algorithm>
#include <iomanip>

// представляет все возможные ходы коня на шахматной доске.
const int moves[8][2] = {
    {2, 1}, {1, 2}, {-1, 2}, {-2, 1},
    {-2, -1}, {-1, -2}, {1, -2}, {2, -1}
};

// Проверяет, является ли ход допустимым.
bool IsValidMove(int x, int y, int M, int N,
                 const std::vector<std::vector<int>>& board)
{
    return (x >= 0 && x < M && y >= 0 && y < N && board[x][y] == 0);
}

// Количество допустимых ходов из текущей позиции.
int CountValidMoves(int x, int y, int M, int N,
                    const std::vector<std::vector<int>>& board) {
    int count = 0;
    for (const auto& move : moves) {
        int nx = x + move[0];
        int ny = y + move[1];
        if (IsValidMove(nx, ny, M, N, board)) {
            count++;
        }
    }
    return count;
}

// обход конём шахматной доски с использованием метода Варнсдорфа.
bool SolveKnightTour(int x, int y, int move_count, int M, int N,
                     std::vector<std::vector<int>>& board) {
    if (move_count == M * N + 1) return true; // Проверяет завершение обхода
    всей доски.

    std::vector<std::pair<int, std::pair<int, int>>> next_moves;

    for (const auto& move : moves) {
        int nx = x + move[0];
        int ny = y + move[1];
        if (IsValidMove(nx, ny, M, N, board)) {
            int valid_moves = CountValidMoves(nx, ny, M, N, board);
```

```
55              next_moves.push_back({valid_moves, {nx, ny}});
56          }
57      }
58      // Сортирует ходы
59      std::sort(next_moves.begin(), next_moves.end(),
60            [](auto &a, auto &b) { return a.first < b.first; });
61
62      for (const auto& next_move : next_moves) {
63          int nx = next_move.second.first;
64          int ny = next_move.second.second;
65          board[nx][ny] = move_count;
66          if (SolveKnightTour(nx, ny, move_count + 1, M, N, board)) return
    true;
67          board[nx][ny] = 0; // backtracking
68      }
69
70      return false;
71  }
72
73  // Записывает текущее состояние доски в выходной файл и на консоль.
74  void PrintBoard(std::ofstream& output_file,
75                  const std::vector<std::vector<int>>& board) {
76      for (const auto& row : board) {
77          for (int cell : row) {
78              output_file << std::setw(3) << cell << " ";
79              std::cout << std::setw(3) << cell << " ";
80          }
81          output_file << std::endl;
82          std::cout << std::endl;
83      }
84  }
85
86  // Извлекает входные данные
87  int ParseValue(const std::string& str) {
88      size_t pos = str.find('=');
89      if (pos != std::string::npos) {
90          return stoi(str.substr(pos + 1));
91      }
92      return -1;
93  }
94
95  int main() {
96      std::ifstream input_file("input.txt");
97      std::ofstream output_file("output.txt");
98
99      if (!input_file || !output_file) {
100         std::cerr << "Не удалось открыть файл." << std::    endl;
101         return 1;
102     }
103
104     std::string line;
105     getline(input_file, line);
106     std::stringstream ss(line);
107
108     std::string m_str, n_str, x_str, y_str;
109     ss >> m_str >> n_str >> x_str >> y_str;
110
111     int M = ParseValue(m_str);
112     int N = ParseValue(n_str);
113     int X = ParseValue(x_str);
```

```
114     int Y = ParseValue(y_str);
115 
116     std::cout << "Входные данные:" << std::endl;
117     std::cout << "M = " << M << ", N = " << N << ", X = " << X << ", Y = "
118               << Y << std::endl;
119 
120     // Проверяет корректность входных данных.
121     if (input_file.fail() || M <= 0 || N <= 0 || X <= 0 || Y <= 0 || X > M
    || Y > N) {
122         std::cerr << "Некорректные входные данные" << std::endl;
123         return 1;
124     }
125 
126     // Предупреждение для больших размеров доски.
127     if (M * N > 1000) {
128         std::cerr << "Предупреждение: большие размеры доски могут замедлить "
129                      "выполнение программы." << std::endl;
130     }
131 
132     std::vector<std::vector<int>> board(M, std::vector<int>(N, 0));
133     board[X - 1][Y - 1] = 1; // Начальная позиция.
134 
135     // Запускает решение
136     if (SolveKnightTour(X - 1, Y - 1, 2, M, N, board)) {
137         std::cout << "Решение найдено:" << std::endl;
138         PrintBoard(output_file, board);
139     } else {
140         std::cout << "Маршрут не существует" << std::endl;
141         output_file << "Маршрут не существует" << std::endl;
142     }
143 
144     input_file.close();
145     output_file.close();
146 
147     return 0;
148 }
```

3. Контрольные тесты

Тест 1

Входные данные: M=6, N=6, X=3, Y=3.

Выходные данные:

31	2	21	16	29	8
22	15	30	9	20	17
3	32	1	18	7	28
14	23	36	27	10	19
33	4	25	12	35	6
24	13	34	5	26	11

Тест 2

Входные данные: M=4, N=4, X=1, Y=1.

Выходные данные:

Маршрут не существует

Тест 3

Входные данные: M=5, N=5, X=1, Y=1.

Выходные данные:

1	22	11	16	7
12	17	8	21	10
25	2	23	6	15
18	13	4	9	20
3	24	19	14	5

Тест 4

Входные данные: M=3, N=3, X=1, Y=1.

Выходные данные:

Маршрут не существует

Тест 3