Санкт-Петербургский государственный политехнический университет

Кафедра “Прикладная математика”

**Отчет по лабораторной работе 3**

**“Алгоритмы и структуры данных”**

Студент группы № 5030102/20001

ФИО: Тишковец Сергей Евгеньевич

Выполнил (дата) 13.11.2023

Оглавление

[**Постановка задачи 3**](#_Toc149322136)

[**Описание алгоритма 3**](#_Toc149322137)

[**Текст программы 4**](#_Toc149322138)

[**Описание тестирования 6**](#_Toc149322139)

# **Постановка задачи**

*Вариант 3:**Кратчайший путь*

На шахматной доске размером N\*N найти кратчайший путь ходами коня из поля А в поле В.

# **Описание алгоритма**

Структура Square:

Состоит из координат абсциссы, ординаты и количества шагов.

Функция для проверки, находится ли точка внутри доски:

1. Возвращает 1, если точка находится внутри доски.
2. В противном случае возвращает 0.

Функция для нахождения наименьшего количества шагов, необходимых коню для достижения точки В:

1. Инициализируется двумерный массив, хранящий в себе информацию о том, был ли в этой точке конь или нет, и заполняется нулями.
2. В виде двумерного массива определяются все возможные ходы для коня.
3. С помощью структуры инициализируется начальная позиция коня, очередь и указатели на начало и конец очереди.
4. Запускается цикл, критерием остановки которого является совпадение начала и конца очереди, то есть момент, когда все возможные варианты будут рассмотрены, но решение не найдется.
5. Внутри цикла производится проверка на совпадение координат текущей точки и точки В. В случае совпадения очередь очищается, и функция возвращает количество шагов.
6. В случае, если точка В не была достигнута, начинается перебор всех возможных вариантов и вычисление кратчайшего пути до точки В.

Ввод исходных данных:

Начальная позиция коня и координаты точки В вводятся пользователем и считываются программой с консоли.

# **Текст программы**

#pragma warning(disable:4996)

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#define N 20 // Board size

// Define a struct for a square

typedef struct {

int x;

int y;

int number;

} Square;

// Function to check if a position is inside the board

int Inside(int x, int y) {

if (x >= 0 && x < N && y >= 0 && y < N) return 1;

return 0;

}

// Function to find the minimum number of steps for a knight to reach the target position

int Solution(int Knight[2], int Target[2]) {

// Initialize the visited matrix

int visited[N][N];

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

visited[i][j] = 0;

}

}

// Define the possible moves for the knight

int Moves[8][2] = { {1, 2}, {1, -2}, {-1, 2}, {-1, -2}, {2, 1}, {2, -1}, {-2, 1}, {-2, -1} };

// Initialize the start position and the queue

Square Start = { Knight[0], Knight[1], 0 };

Square\* queue = (Square\*)malloc(N \* N \* sizeof(Square));

int front = 0, rear = 0;

queue[rear++] = Start;

while (front != rear) {

Square curr = queue[front++];

if (curr.x == Target[0] && curr.y == Target[1]) { // Target was reached

free(queue);

return curr.number;

}

for (int i = 0; i < 8; i++) {

int x\_new = curr.x + Moves[i][0];

int y\_new = curr.y + Moves[i][1];

// Finding the minimum number of steps to reach the target

if (Inside(x\_new, y\_new) && !visited[x\_new][y\_new]) {

Square new\_square = { x\_new, y\_new, curr.number + 1 };

queue[rear++] = new\_square;

visited[x\_new][y\_new] = 1;

}

}

}

free(queue);

return -1; // Target position is not reachable

}

int main() {

int Knight[2]; // Knight position

printf("Enter knight position: ");

scanf("%d %d", &Knight[0], &Knight[1]);

int Target[2]; // Target position

printf("Enter target position: ");

scanf("%d %d", &Target[0], &Target[1]);

int number = Solution(Knight, Target);

if (number != -1) printf("Number of steps: %d", number);

else printf("Impossible to reach the position");

return 0;

}

# **Описание тестирования**

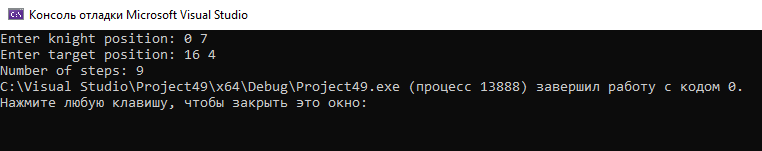
Для тестирования данной программы производился ее неоднократный запуск с вводом конкретных данных, при котором проверялось:

1. Стабильность работы программы при одинаковых входных данных;
2. Корректное выполнение всех заявленных процедур;
3. Корректное завершение программы;

Приведем пример тестирования на скриншотах, представленных ниже:

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание



При проведении тестирования такого рода никаких проблем обнаружено не было, что позволяет судить о корректности работы программы в целом.