**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

**отчет**

**по лабораторной работе №2**

**по дисциплине «Построение и анализ алгоритмов»**

**Тема: Поиск с возвратом.**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 5382 | Лисс Н.И. |  |
| Преподаватель | Шолохова О.М. |  |

Санкт-Петербург

2017

**Цель работы.**

Ознакомиться с основными понятиями и приёмами использования поиска с возвратом, получить навыки решения задач с его помощью.

**Формулировка задания.**

*Симметричная задача о небьющих друг друга ферзях.* Найдите все решения задачи о небьющих друг друга ферзях, инвариантные относительно поворотов на 90º, 180 º и 270 º. В такой постановке каждый ферзь, помещенный не в центре, определяет положение двух других ферзей.

**Теоретические положения.**

Поиск с возвратом (от англ. Backtracking) - это общий метод нахождения решений задачи, в которой требуется полный перебор всех возможных вариантов в некотором множестве М, как правило, позволяет решать задачи, в которых ставятся вопросы типа: "Перечислите все возможные варианты ...", "Сколько существует способов ...", "Есть ли способ ...", "Существует ли объект..." и т. п.

Решение задачи методом поиска с возвратом сводится к последовательному расширению частичного решения. Если на очередном шаге такое расширение провести не удается, то возвращаются к более короткому частичному решению и продолжают поиск дальше. Данный алгоритм позволяет найти все решения поставленной задачи, если они существуют. Для ускорения метода стараются вычисления организовать таким образом, чтобы как можно раньше выявлять заведомо неподходящие варианты. Зачастую это позволяет значительно уменьшить время нахождения решения

Метод поиска с возвратом является универсальным. Достаточно легко проектировать и программировать алгоритмы решения задач с использованием этого метода. Однако время нахождения решения может быть очень велико даже при небольших размерностях задачи (количестве исходных данных), причём настолько велико (может составлять месяцы или даже годы), что о практическом применении не может быть и речи. Поэтому при проектировании таких алгоритмов, обязательно нужно теоретически оценивать время их работы на конкретных данных. Существуют также задачи выбора, для решения которых можно построить уникальные, "быстрые" алгоритмы, позволяющие быстро получить решение даже при больших размерностях задачи. Метод поиска с возвратом в таких задачах применять неэффективно.

**Формальная постановка задачи.**

Исходные данные:Длина стороны доски (матрицы).

Результирующие данные: Информация о том, на каких местах нужно расположить ферзей, чтобы они не били друг друга.

**Спецификация программы.**

Место и форма представления входных данных: входные данные записаны как глобальная переменная и представляют из себя число.

Выходные данные: программа выводит на экран матрицы, где знак «!» означает, что на этом месте стоит ферзь, а знак «.» означает, что на этом месте ничего не стоит.

**Описание алгоритма программы.**

1. Начинаем с пустой доски.
2. Ставим ферзя на свободную строку. Бьём все ячейки, которые находятся по отношению к нему по вертикали, горизонтали и вертикалям.
3. Проверяем, осталось ли не битое место для ферзя в следующем столбце, если да, ставим на свободную строку, если нет, то удаляем нашего ферзя и ставим на следующую строку.(такой приём называется backtracking)
4. Программа завершает свою работу, когда найдены все существующие решения.

**Сложность алгоритма.**

Верхняя оценка:

Сложность постановки первого ферзя =

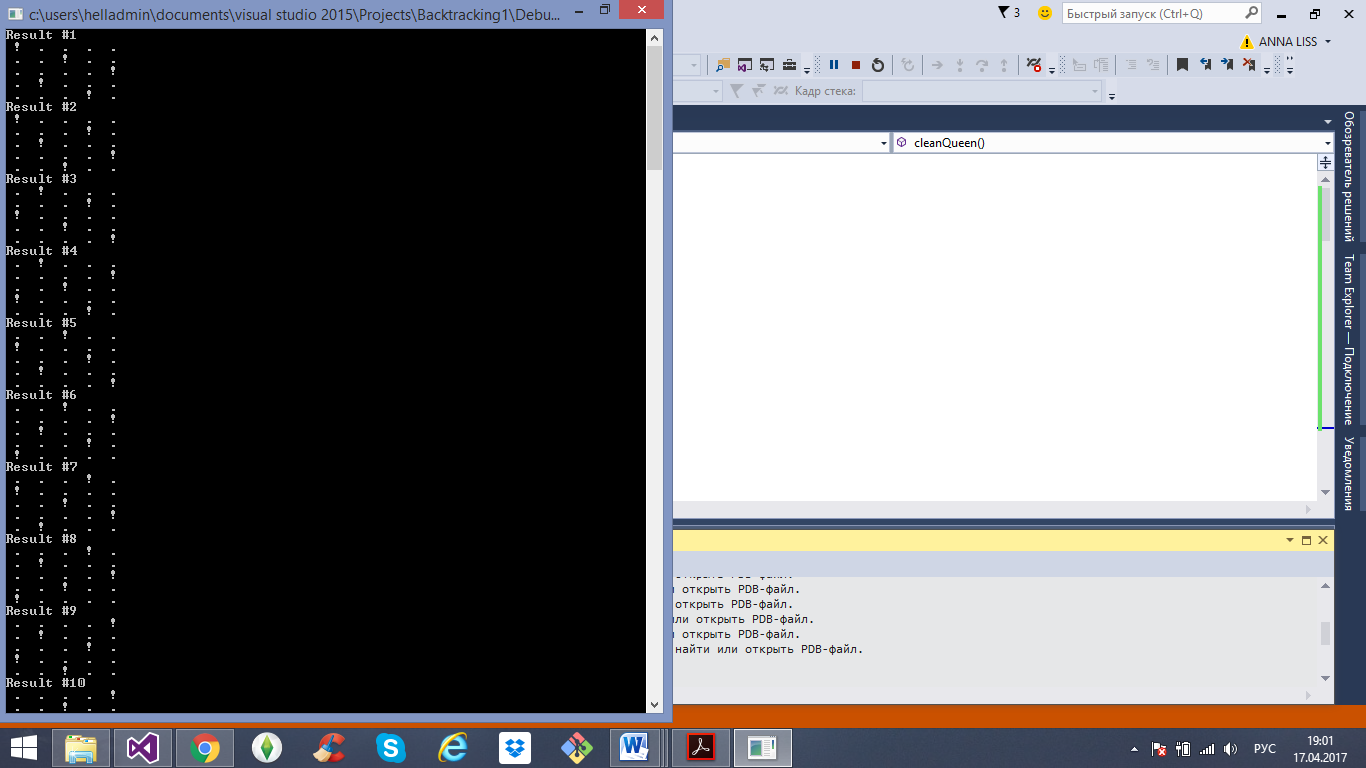
Сложность постановки второго ферзя =

Сложность постановки третьего ферзя =

Следовательно, сложность алгоритма = ()…

**Пример работы программы.**

Размер стороны матрицы = 5.



**Вывод.**

В результате выполнения данной лабораторной работы я изучила теоретические сведения по методу поиска с возвратом (backtracking), а также эти навыки реализовала на языке программирования C++.

**Текст программы.**

**backtracking.cpp**

// Backtracking.cpp: определяет точку входа для консольного приложения.

//

#include "stdafx.h"

#include "Queen.h"

int main()

{

cleanQueen();

tryQueen(0);

//printQueen();

system("pause");

return 1;

}

**Queen.h**

#pragma once

#include <iostream>

using namespace std;

void setQueen(int i, int j);

void resetQueen(int i, int j);

void tryQueen(int i);

void printQueen();

void cleanQueen();

**Queen.cpp**

#include "stdafx.h"

#include "Queen.h"

const int SIZE = 5;

int board[SIZE][SIZE];//матрица ячеек

int resultsCount = 0;

int quantity = 0;

void setQueen(int i, int j)//ставим ферзя и заполняем ячейки, которые ферзь будет бить

{

for (int x = 0; x < SIZE; ++x)

{

++board[x][j];//прибавляем по горизонтали

++board[i][x];//прибавляем по вертикали

int foo;

foo = j - i + x;

if (foo >= 0 && foo < SIZE)//прибавляем по диагонали

++board[x][foo];

foo = j + i - x;

if (foo >= 0 && foo < SIZE)

++board[x][foo];

}

board[i][j] = -1;//-1 означает, что там стоит ферзь

}

void resetQueen(int i, int j)//удаляем ферзя

{

for (int x = 0; x < SIZE; ++x)

{

--board[x][j];//отнимаем по горизонтали

--board[i][x];//отнимаем по вертикали

int foo;

foo = j - i + x;//по горизонтали

if (foo >= 0 && foo < SIZE)

--board[x][foo];

foo = j + i - x;

if (foo >= 0 && foo < SIZE)

--board[x][foo];

}

board[i][j] = 0;

}

void resultQueen()

{

cout << "Result #" << ++resultsCount << endl;

printQueen();

}

void tryQueen(int i)//пытается поставить ферзя на i-ый столбец, если это удасться , то возвращает true

{

int j = 0;

if (quantity == SIZE)//значит всех ферзей поставили

{

resultQueen();

return;

}

for (j = 0; j < SIZE; ++j)

{

if (board[i][j] == 0)//никто не бьет эту ячейку

{

setQueen(i, j);

quantity++;

tryQueen(i + 1);//если не удалось поставить ферзя на следующий столбец

resetQueen(i, j);

quantity--;

}

/\*if (result)// удалось поставить ферзя

break;\*/

}

}

void printQueen()

{

for (int i = 0; i < SIZE; ++i)

{

for (int j = 0; j < SIZE; ++j)

{

if (board[i][j] == -1)

cout << " ! ";//ферзь

else

cout << " . "; //отсутствие ферзя

}

cout << endl;

}

}

void cleanQueen()

{

for (int i = 0; i < SIZE; ++i)

for (int j = 0; j < SIZE; ++j)

board[i][j] = 0;

}