**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: Бэктрекинг. Магический квадрат.**

|  |  |  |
| --- | --- | --- |
| Студент гр. 5382 |  | Никитин В.А. |
| Преподаватель |  | Шолохова О.М. |

Санкт-Петербург

2017 г.

**Цель работы.**

Ознакомиться с основными сортировками, получить навыки программирования метода бэктрекинга на языке программирования С++.

**Постановка задачи.**

Необходимо реализовать построение и подсчет всех вариантов магических квадратов nxn.

**Основные теоретические положения.**

**Магический квадрат** порядка n представляет собой таблицу nxn, заполненную числами от 1 до n2, которые расположены так, что суммы в каждой строке, в каждом столбце и двух диагоналях одинаковы. Сумма равна n(n2+1)/2.

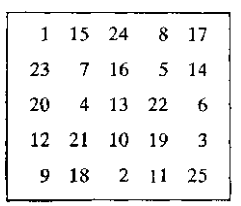


Рисунок 1.

**Бэктрекинг** — 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.

**Спецификация программы.**

*Назначение программы*.

Программа предназначена для построения всех возможных магических квадратов с стороной равной n.

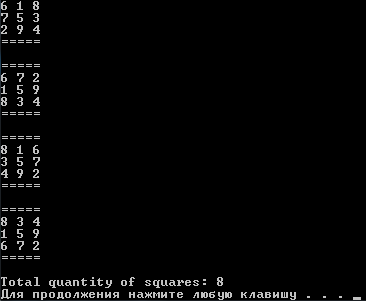
*Описание программы*.

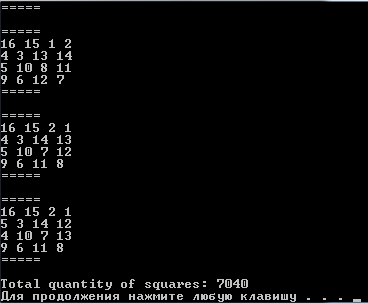
Программа написана на языке C++. На вход программа принимает только размер стороны квадрата, причем желательно, чтобы размер не превышал 4, т.к. уже при n = 4, программа генерирует >7000 квадратов за продолжительный период времени. На выход программы выводит поочередно все квадраты и суммарное их число.

*Реализация*.

Функции:

* void magicsquare(int square[size][size],const int size) //создает квадрат n\*n, в котором все значения равные 0, и массив, который отвечает за “доступность“ ячейки в квадрате.
* bool checkrows(int square[size][size], const int size) // проверяет все строки на удовлетворение условиям магического квадрата.
* bool checkcols(int square[size][size], const int size) // проверяет все столбцы на удовлетворение условиям магического квадрата.
* bool checkdiags(int square[size][size], const int size) // проверяет главные диагонали на удовлетворение условиям магического квадрата.
* void fill(int square[size][size], const int size, int row, int col) // основная функция генерации квадрата. Проверяет квадрат на наполненность и на удовлетворение условию магического квадрата. Рекурсивно заполняет все ячейки в квадрате.

**Тестирование.**

****

**Выводы**

В ходе выполнения лабораторной работы был исследован алгоритм бэктрекинга, была написана программа, генерирующая все варианты магического квадрата n\*n.

**Приложение А. Исходный код.**

#include "stdafx.h"

#include <iostream>

using namespace std;

const int size = 3;

const int ttlsqs = size\*size;

int sum = size\*(size\*size+1)/2;

int square[size][size];

bool possible[ttlsqs];

int squarecounter = 0;

void magicsquare(int square[size][size],const int size){

//зануляем

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

for (int j = 0; j < size; ++j)square[i][j] = 0;

}

//начинаем со всеми возможными номерами

for (int i = 0; i < ttlsqs; ++i) possible[i] = true;

}

bool checkrows(int square[size][size], const int size){

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

int test = 0;

bool unfilled = false;

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

test += square[i][j];

if (square[i][j] == 0) unfilled = true;

}

if ((!unfilled) && (test != sum))return false;

}

return true;

}

bool checkcols(int square[size][size], const int size){

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

int test = 0;

bool unfilled = false;

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

test += square[j][i];

if (square[j][i] == 0) unfilled = true;

}

if ((!unfilled) && (test != sum))return false;

}

return true;

}

bool checkdiags(int square[size][size], const int size){

int test1 = 0;

int test2 = 0;

bool unfilled = false;

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

test1 += square[i][i];

test2 += square[i][size - i - 1];

if (square[i][i] == 0) unfilled = true;

}

if ((!unfilled) && ((test1 != sum) || (test2 != sum)))return false;

return true;

}

void fill(int square[size][size], const int size, int row, int col){

if (!checkrows(square, size) || !checkcols(square, size) || !checkdiags(square, size)) return;

if (row == size){

cout << endl << "=====" << endl;

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

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

cout << square[i][j]<<" ";

}

cout << endl;

}

cout << "=====" << endl;

squarecounter++;

return;

}

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

if (possible[i]){

square[row][col] = i + 1;

possible[i] = false;

int newcol = col + 1;

int newrow = row;

if (newcol == size){

newrow++;

newcol = 0;

}

fill(square, size, newrow, newcol);

square[row][col] = 0;

possible[i] = true;

}

}

}

int main() {

magicsquare(square, size);

fill(square, size, 0, 0);

cout << endl <<"Total quantity of squares: "<< squarecounter<<endl;

system("pause");

}