**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №1**

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

Тема: «Поиск с возвратом»

|  |  |  |
| --- | --- | --- |
| Студент гр. 6381 |  | Вероха В. Н. |
| Преподаватель |  | Филатов А.Ю. |

Санкт-Петербург

2018

**Цель работы:**

Изучение алгоритма поиска с возвратом (backtracking).

**Условия задания:**

У Вовы много квадратных обрезков доски. Их стороны (размер) изменяются от 1 до *N*−1, и у него есть неограниченное число обрезков любого размера. Но ему очень хочется получить большую столешницу - квадрат размера *N*. Он может получить ее, собрав из уже имеющихся обрезков (квадратов).



Рисунок 1. Пример разбиения квадрата со стороной 7.

Внутри столешницы не должно быть пустот, обрезки не должны выходить за пределы столешницы и не должны перекрываться. Кроме того, Вова хочет использовать минимально возможное число обрезков.

**Входные данные:**

Размер столешницы - одно целое число ***N***(2≤***N***≤20).

**Выходные данные:**

Одно число *K*, задающее минимальное количество обрезков (квадратов), из которых можно построить столешницу (квадрат) заданного размера *N*. Далее должны идти *K* строк, каждая из которых должна содержать три целых числа *x*,*y* и *w*, задающие координаты левого верхнего угла (1≤*x*, *y*≤*N*) и длину стороны соответствующего обрезка (квадрата).

**Пример входных данных**7 **Соответствующие выходные данные**9  
1 1 2  
1 3 2  
3 1 1  
4 1 1  
3 2 2  
5 1 3  
4 4 4  
1 5 3  
3 4 1

**Описание алгоритма:**

Сторона квадрата N задается пользователем. Проверка делимости N на 2, 3 и 5. Если N чётное, то минимально число квадратов, на которое можно разбить квадрат с чётной стороной – 4. Если делимость была обнаружена, то результат – это квадрированный квадрат в масштабе N/2, N/3 и N/5 соответственно. Если же число простое (начиная с 7), то устанавливаем 3 квадрата следующим образом: 1 – nxn, 2 – (n+1) x(n+1), 3 – nxn (сторона квадрата – 2n+1). После чего вызываем функцию *SearchEmptyArea* для поиска ближайшей свободной области, если нашли такую точку, то происходит вызов функции *Recurcion* для рассмотрения всех возможных вариантов заполнения оставшейся области и выбора лучшего из них.

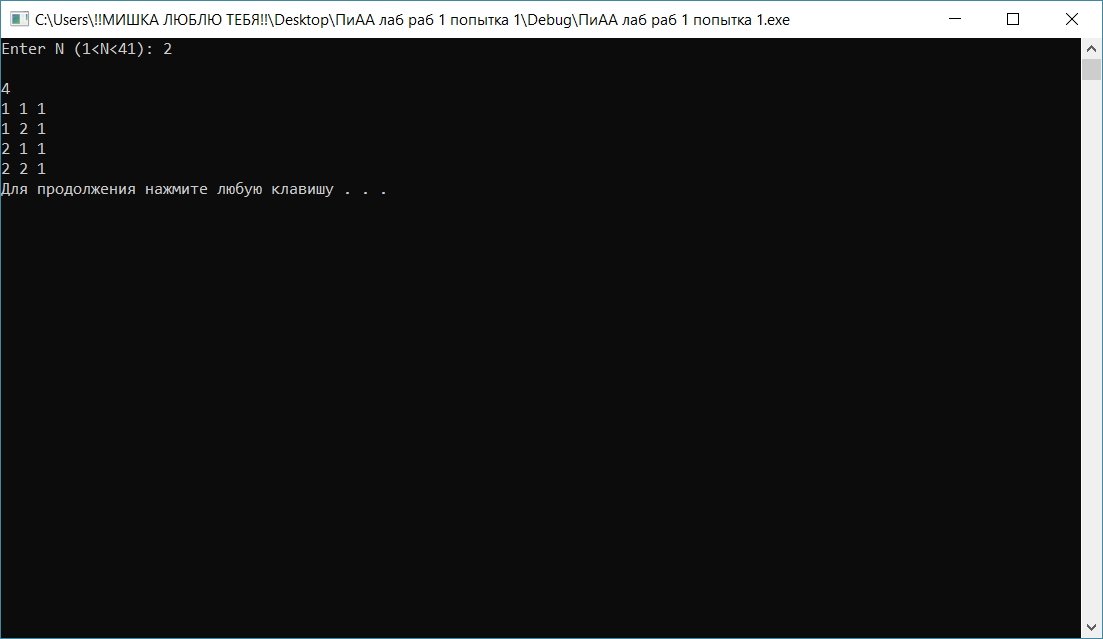
**Описание функций:**

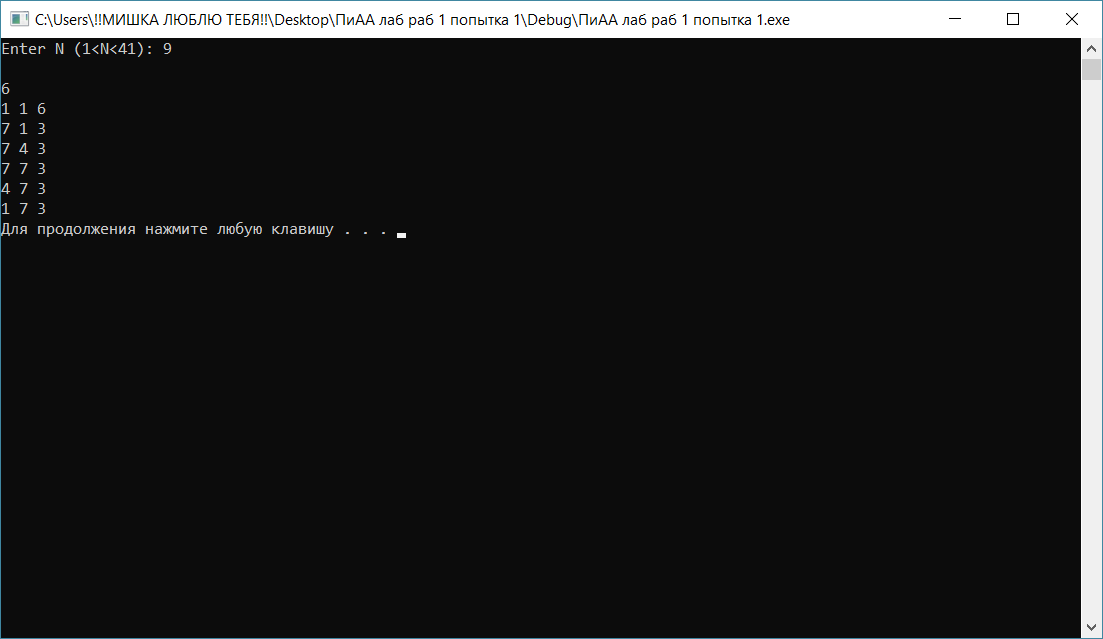
*bool SetLength(int N, int x, int y, int size, int length, vector<vector<int>>& Table)* - функция, которая устанавливает область двумерного массива Table с ячейки c индексом [x][y] (левый верхний угол) в виде квадрата со стороной size.

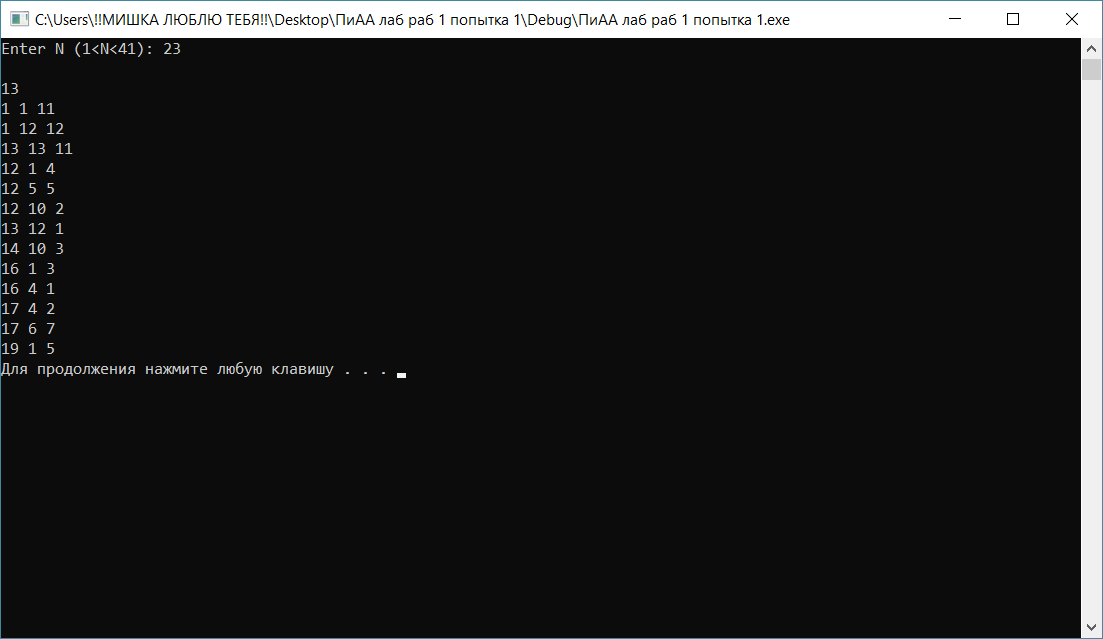
*int SearchEmptyArea (int last\_x, int last\_y, vector<vector<int> > & Table, int N)* - функция осуществляет поиск ближайшей свободной клетки в двумерном массиве Table.

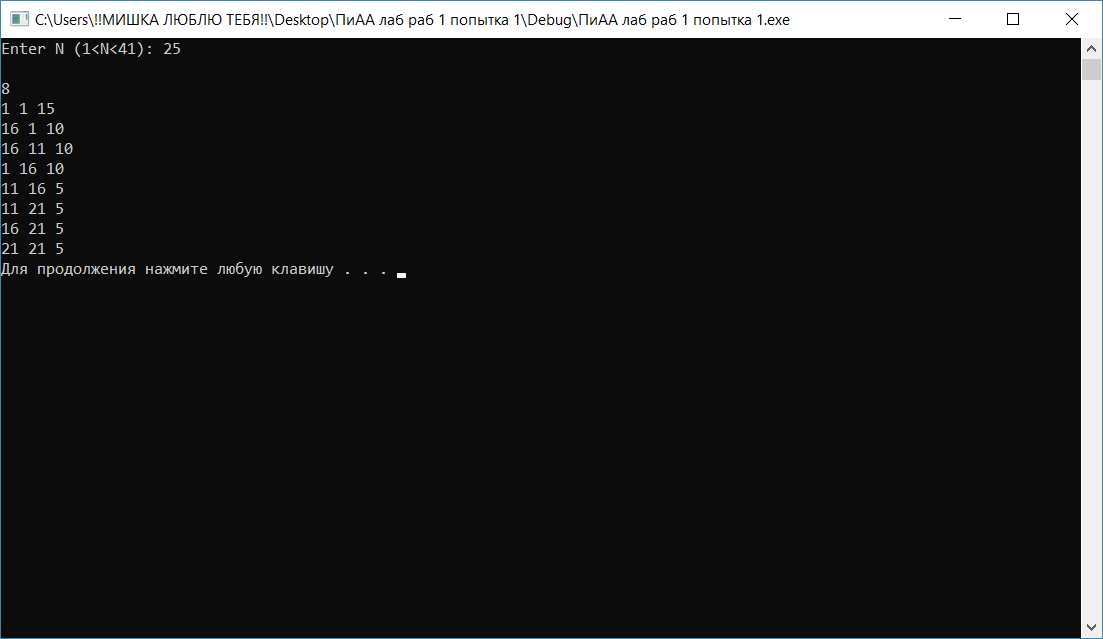
int Recurcion(vector<vector<int> > & Table, vector<vector<int> > & Result, int N, int x, int y, int MinNumberSquare, int CurrentNumberSquare) - рекурсивная функция, осуществляющая перебор подстановок всех возможных размеров квадратов с левым верхним углом в ячейке с индексами x и y, а также поиск минимального квадрирования заданного квадрата и сохранение полученного результата.

**Тестирование:**









**Вывод:**

В ходе выполнения лабораторной работы был реализован алгоритм поиска с возвратом и на его основе была решена задача минимального заполнения квадрата размером NxN.

**Приложение А. Код программы:**

#include "stdafx.h"

#include <iostream>

#include <vector>

using namespace std;

bool SetLength(int N, int x, int y, int size, int length, vector<vector<int> >& Table) {

if (y + size > N || x + size > N) {

return false;

}

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

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

if (Table[i][j] = 0)

return false;

}

}

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

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

Table[i][j] = length;

}

}

return true;

}

int SearchEmptyArea(int last\_x, int last\_y, vector<vector<int> > & Table, int N) {

for (int j = last\_y; j < N; j++) {

if (Table[last\_x][j] == 0)

return last\_x \* 100 + j;

}

for (int i = last\_x + 1; i < N; i++) {

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

if (Table[i][j] == 0)

return i \* 100 + j;

}

}

return -1;

}

int Recurcion(vector<vector<int> > & Table, vector<vector<int> > & Result, int N, int x, int y, int MinNumberSquare, int CurrentNumberSquare) {

int Point = 0;

int MaxSquare = N - 1;

int ResultNumber = 0;

int Y = y + 1;

int C = CurrentNumberSquare - 1;

if (CurrentNumberSquare > MinNumberSquare)

return MinNumberSquare;

while ((Y < N) && (Y >= 0)) {

if (Table[x][Y] != 0) {

MaxSquare = Y - y;

Y = -2;

}

Y++;

}

if (Y == N)

MaxSquare = N - y;

if ((N - x) < MaxSquare)

MaxSquare = N - x;

for (int i = 1; i <= MaxSquare; i++)

{

SetLength(N, x, y, i, CurrentNumberSquare, Table);

Point = SearchEmptyArea(x, y + i - 1, Table, N);

if (Point > -1) {

ResultNumber = Recurcion(Table, Result, N, Point / 100, Point % 100, MinNumberSquare, CurrentNumberSquare + 1);

if (ResultNumber < MinNumberSquare) {

MinNumberSquare = ResultNumber;

Result[C][0] = x + 1;

Result[C][1] = y + 1;

Result[C][2] = i;

}

}

else {

if (CurrentNumberSquare < MinNumberSquare) {

Result[C][0] = x + 1;

Result[C][1] = y + 1;

Result[C][2] = i;

MinNumberSquare = CurrentNumberSquare;

}

SetLength(N, x, y, i, 0, Table);

return MinNumberSquare;

}

}

SetLength(N, x, y, MaxSquare, 0, Table);

return MinNumberSquare;

}

int main() {

int N = 0;

cout << "Enter N (1<N<41): ";

cin >> N;

cout << endl;

vector<vector<int> > Table(N, vector<int>(N, 0));

vector<vector<int> > Result(2 \* N, vector<int>(3, 0));

if (N % 2 == 0) {

int M = N / 2 + 1;

cout << 4 << endl;

cout << 1 << " " << 1 << " " << N / 2 << endl;

cout << 1 << " " << M << " " << N / 2 << endl;

cout << M << " " << 1 << " " << N / 2 << endl;

cout << M << " " << M << " " << N / 2 << endl;

}

else if (N % 3 == 0) {

int M = N / 3;

int K = 2 \* M + 1;

cout << 6 << endl;

cout << 1 << " " << 1 << " " << 2 \* M << endl;

cout << K << " " << 1 << " " << M << endl;

cout << K << " " << M + 1 << " " << M << endl;

cout << K << " " << K << " " << M << endl;

cout << M + 1 << " " << K << " " << M << endl;

cout << 1 << " " << K << " " << M << endl;

}

else if (N % 5 == 0) {

int M = N / 5;

int L = 1 + 2 \* M;

int K = 1 + 3 \* M;

int O = 1 + 4 \* M;

cout << 8 << endl;

cout << 1 << " " << 1 << " " << 3 \* M << endl;

cout << K << " " << 1 << " " << 2 \* M << endl;

cout << K << " " << L << " " << 2 \* M << endl;

cout << 1 << " " << K << " " << 2 \* M << endl;

cout << L << " " << K << " " << M << endl;

cout << L << " " << O << " " << M << endl;

cout << K << " " << O << " " << M << endl;

cout << O << " " << O << " " << M << endl;

}

else {

int Point = 0;

int MinNumberSquare = 0;

int M = (N - 1) / 2;

int K = (N + 1) / 2;

SetLength(N, 0, 0, M, 1, Table);

Result[0][0] = 1;

Result[0][1] = 1;

Result[0][2] = M;

SetLength(N, 0, M, K, 2, Table);

Result[1][0] = 1;

Result[1][1] = K;

Result[1][2] = K;

SetLength(N, K, K, M, 3, Table);

Result[2][0] = K + 1;

Result[2][1] = K + 1;

Result[2][2] = M;

Point = SearchEmptyArea(0, N - 1, Table, N);

MinNumberSquare = Recurcion(Table, Result, N, Point / 100, Point % 100, 2 \* N + 1, 4);

cout << MinNumberSquare << endl;

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

for (int j = 0; j < 3; j++)

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

cout << endl;

}

}

system("pause");

return 0;

}