**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №1**

**по дисциплине «ПиАА»**

Тема: Бэктрекинг

|  |  |  |
| --- | --- | --- |
| Студент(ка) гр. 0000 |  | Ивченко А.А. |
| Преподаватель |  | Размочаева Н.В. |

Санкт-Петербург

2020

**Цель работы.**

Ознакомиться с работой алгоритма поиска с возвратом, научится применять полученные знания в решении задач на перебор всех возможных вариантов.

1

**Формулировка 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).

N

Например, столешница размера 7×7 может быть построена из 9 обрезков.

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.

**Описание алгоритма.**

В ходе работы был реализован класс Matrix размера N\*N.

Для решения поставленной задачи был разработан алгоритм, осуществляющий поиск с возвратом итеративным методом. Для упрощения и сокращения количества итераций примерно ¾ матрицы заполняется 3-мя квадратами. Для больших значений N в свободный угол ставится 4-ый квадрат.

Работа алгоритма заключается в последовательном заполнении свободных областей по возможности максимально большими квадратами и уменьшении сторон наименьших квадратов на 1 (в случае, если квадрат единичный, он удаляется) до тех пор, пока матрица не станет заполненной полностью. Наименьшее количество обрезков, то есть лучшее решение в процессе перебора вариантов расстановок устанавливается как максимальное значение для следующих проверок. Алгоритм прекращает свою работу, когда текущее число квадратов в стеке превосходит лучшее решение.

Алгоритм оптимизирован для не простых чисел. Оценочная сложность алгоритма O(n3) в лучшем случае.

**Вывод.**

В ходе лабораторной работы был разобран алгоритм поиска с возвратом, в частности итеративный его метод реализации. Была составлена программа, выполняющая поиск наилучшей конфигурации квадратов в заданных границах, а также считающая время работы алгоритма.

**ИСХОДНЫЙ КОД**

**#include** <iostream>

#include <vector>

#include <ctime>

#include <algorithm>

using namespace std;

struct Square{

int x,y,len;

};

class Matrix{

private:

int n;

int\*\* matrix;

int count,new\_count;

vector <Square> sq\_arr;

vector <vector<Square>> variants;

public:

Matrix(int length, int count) :n(length), new\_count(length\* length), count(0){

matrix = new int\* [length];

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

matrix[i] = new int[length];

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

matrix[i][j] = 0;

}

}

}

void set\_square(Square sq){

count++;

for (int i = sq.y; i < sq.y + sq.len; i++){

for (int j = sq.x; j < sq.x + sq.len; j++){

matrix[i][j] = count;

}

}

}

void rem\_square(Square sq){

for (int i = sq.y; i < sq.y + sq.len; i++){

for (int j = sq.x; j < sq.x + sq.len; j++){

matrix[i][j] = 0;

}

}

count--;

}

bool check(int x, int y, int m) {

if (x >= n || y >= n)

return false;

if (x + m > n || y + m > n)

return false;

for (int i = y; i < y + m; i++) {

for (int j = x; j < x + m; j++) {

if (matrix[i][j] != 0) {

return false;

}

}

}

return true;

}

bool is\_filled() {

for (int i = n - 1; i >= 0; --i)

for (int j = n - 1; j >= 0; --j)

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

return false;

return true;

}

void backtracking()

{

Square sq;

sq.len = ceil(n / 2);

sq.x = 0;

sq.y = 0;

sq\_arr.push\_back(sq);

set\_square(sq);

sq.x = n / 2;

sq.y = 0;

sq.len = n / 2 + 1;

sq\_arr.push\_back(sq);

set\_square(sq);

sq.x = n/2+1;

sq.y = n / 2 + 1;

sq.len = n / 2;

sq\_arr.push\_back(sq);

set\_square(sq);

if (n > 15) {

sq.len = n / 4+1;

sq.x = 0;

sq.y = 3\*n / 4;

sq\_arr.push\_back(sq);

set\_square(sq);

count = 4;

}

while (count < new\_count && !is\_filled()) {

for (int y = 0; y < n; y++) {

for (int x = 0; x < n; x++) {

if (matrix[y][x] == 0) {

for (int size\_square = n; size\_square > 0; size\_square--) {

if (check(x, y, size\_square)) {

Square sq{ x,y, size\_square };

set\_square(sq);

sq\_arr.push\_back(sq);

break;

}

}

}

}

}

/\*

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

{

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

{

std::cout << table[i][j] << " ";

}

std::cout << "\n";

}

std::cout << "\n";

\*/

if (count < new\_count)

{

new\_count = count;

variants.push\_back(sq\_arr);

}

while (!sq\_arr.empty() && sq\_arr[sq\_arr.size() - 1].len < n/5 + 1) {

rem\_square(sq\_arr[sq\_arr.size() - 1]);

sq\_arr.pop\_back();

}

if (!(sq\_arr.empty() && (sq\_arr[sq\_arr.size() - 1]).len < n/2) ){//уменьшение стороны верхнего квадрата в стеке на 1

sq = sq\_arr[sq\_arr.size() - 1];

sq\_arr.pop\_back();

rem\_square(sq);

sq.len -= 1;

set\_square(sq);

sq\_arr.push\_back(sq);

}

}

}

void print(int k){

Square sq;

vector<int> minEls;

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

int minElement = variants[i].size();

minEls.push\_back(minElement);

}

int minElementIndex = std::min\_element(minEls.begin(), minEls.end()) - minEls.begin();

int minElement = \*std::min\_element(minEls.begin(), minEls.end());

std::cout << minElement << '\n';

for (int i = variants.size()-1; i > 0; i--) {

if (minElement == variants[i].size())

{

for (int j = variants[i].size()-1; j >= 0; j--) {

sq = variants[i][j];

variants[i].pop\_back();

std::cout << sq.x \* k + 1 << " " << sq.y \* k + 1 << " " << sq.len \* k;

if (!(variants[i].empty()))

std::cout << std::endl;

}

}

else

variants.pop\_back();

}

}

};

int simplify(int size) {

int start = size;

for (int i = 2; i <= size; i++){

if (size % i == 0)

return i;

}

return start;

}

int main(){

int n;

std::cin >> n;

int start\_size = simplify(n);

Matrix matrix(start\_size, 0);

int kf = n / start\_size;

if (n % 2 == 0) {

std::cout << 4 << "\n";

std::cout << 1 << " " << 1 << " " << n / 2 << "\n";

std::cout << 1 + n / 2 << " " << 1 << " " << n / 2 << "\n";

std::cout << 1 << " " << 1 + n / 2 << " " << n / 2 << "\n";

std::cout << 1 + n / 2 << " " << 1 + n / 2 << " " << n / 2 << "\n";

return 0;

}

else if (n % 3 == 0) {

std::cout << 6 << "\n";

std::cout << 1 << " " << 1 << " " << 2 \* n / 3 << "\n";

std::cout << 1 + 2 \* n / 3 << " " << 1 << " " << n / 3 << "\n";

std::cout << 1 << " " << 1 + 2 \* n / 3 << " " << n / 3 << "\n";

std::cout << 1 + 2 \* n / 3 << " " << 1 + n / 3 << " " << n / 3 << "\n";

std::cout << 1 + n / 3 << " " << 1 + 2 \* n / 3 << " " << n / 3 << "\n";

std::cout << 1 + 2 \* n / 3 << " " << 1 + 2 \* n / 3 << " " << n / 3 << "\n";

return 0;

}

else if (n % 5 == 0) {

std::cout << 8 << "\n";

std::cout << 1 << " " << 1 << " " << 3 \* n / 5 << "\n";

std::cout << 1 + 3 \* n / 5 << " " << 1 << " " << 2 \* n / 5 << "\n";

std::cout << 1 << " " << 1 + 3 \* n / 5 << " " << 2 \* n / 5 << "\n";

std::cout << 1 + 3 \* n / 5 << " " << 1 + 3 \* n / 5 << " " << 2 \* n / 5 << "\n";

std::cout << 1 + 2 \* n / 5 << " " << 1 + 3 \* n / 5 << " " << n / 5 << "\n";

std::cout << 1 + 2 \* n / 5 << " " << 1 + 4 \* n / 5 << " " << n / 5 << "\n";

std::cout << 1 + 3 \* n / 5 << " " << 1 + 2 \* n / 5 << " " << n / 5 << "\n";

std::cout << 1 + 4 \* n / 5 << " " << 1 + 2 \* n / 5 << " " << n / 5 << "\n";

return 0;

}

srand(time(0));

matrix.backtracking();

matrix.print(kf);

cout << std::endl << "runtime = " << clock() / 1000.0 << endl;

return 0;

}