**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №1.**

**по дисциплине «Алгоритмы и структуры данных»**

**Тема: «Поиск с возвратом»**

Студентка гр. 7382 Лящевская А. П.

Преподаватель Фирсов М. А.

Санкт-Петербург

2019

**Цель работы.**

Ознакомиться с алгоритмом перебора с возвратом и научиться применять его на практике.

**Задание.**

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).

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.

**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

**Индивидуальное задание.**

**Вариант 1p.**

Рекурсивный бэктрекинг. Выполнение на Stepik всех трёх заданий в разделе 2.

**Описание алгоритма.**

На вход программа ожидает число от 2 до 40, оно будет соответствовать длине квадрата для разбиения, иначе выводится сообщение об  
ошибке и программа экстренно завершается.

Если рассмотреть разбиение всех квадратов длины от 2 до 40, то можно вывести следующее утверждение: минимальному разбиению (разбиению с наименьшим количеством квадратов) для непростых чисел *N* (Напр.: 6, 21, 33) будет соответствовать разбиение квадрата с длиной стороны равной наименьшему целочисленному делителю числа *N*не равному единице. Так, с помощью функции *multiplicity* находится этот самый делитель *mul*, если он есть и далее идет работа уже либо с квадратом длины *mul*, либо с прежним квадратом, если наименьший делитель *N* равен единице.

Сам квадрат в памяти программы представлен в виде двумерного массива *arr*. Кроме того, существует стандартный стек *sqrs* с типом элементов *Square*, где *Square* это структура, которая будет содержать информацию о квадрате, местоположение его левого верхнего угла относительно начала массива *arr* и длину самого квадрата. Для успешного выполнения задания была задействована функция *6.37*\**N*^(*1*/*4*) которая определяет переменную *limit* и дает приблизительное представление о минимальном количестве квадратов для соответствующего *N* (*mul*). Она нужна для того чтобы ограничить перебор, так как его сложность по памяти от числа к числу возрастает экспоненциально*.*

Далее элементы *limit*, *&sqrs*, *arr*, *N* (*mul*), 0 из main подаются в рекурсивную функцию поиска минимального количества квадратов, где 0 из этого списка означает 0-й уровень рекурсии.

В самой рекурсии происходит следующее:

1. Проверка на достижение предельной глубины рекурсии. Если это предел, то возвращается -1.
2. Поиск пустой клетки в массиве *arr*. Если таковой не оказалось, то возвращается 0.
3. Создание дополнительных стеков *srqs\_tmp* и *sqrs\_min*. В первый будут записываться текущие квадраты, а во второй их минимальная последовательность.
4. Нахождение максимальной длины квадрата, который можно поместить, начиная с данной точки левого верхнего угла, с помощью функции *find\_max\_length*.
5. Далее пробуются разные длины квадрата. Для этого происходит заполнение квадрата функцией *fill\_square*.
6. Рекурсивное обращение из функции в эту же функцию, только с счетчиком рекурсии +1.
7. Возращенное значение количества квадратов записывается в переменную *k.*
8. Происходит проверка значения *k* на минимальное и, если это так, запоминаем длину текущего квадрата, *k\_min* и переписываем *sqrs\_min* значениями из *sqrs\_tmp*. Если же нет, то опустошаем *sqrs\_tmp*.
9. Далее очищаем квадрат. И переходим к следующей длине квадрата.
10. Когда достигнем максимальной длины квадрата *max\_length* копируем в *sqrs* значения из *sqrs\_min*, записываем в *sqrs* текущий квадрат и возвращаем *k\_min.*

После выхода из рекурсии выводится *k\_min*, значения квадратов в нужном масштабе и освобождается память.

**Описание структур**

|  |  |  |
| --- | --- | --- |
| **Название структуры** | **Объект** | **Описание** |
| struct Square {} | int *x*; | Координата квадрата x относительно начала массива. |
| int *y*; | Координата квадрата y относительно начала массива. |
| int *length*; | Длина квадрата. |

**Описание функций**

|  |  |  |
| --- | --- | --- |
| **Сигнатура** | **Параметры** | **Описание** |
| *int multiplicity (int N)* | *N* – длина квадрата | Возвращает наименьший делитель не равный нулю, если это непростое число. В случае с простым числом возвращает 1. |
| *void demo (int \*\*arr, int N)* | *arr* – массив клеток квадрата  *N* – длина массива | Демонстрация массива. |
| *void fill\_square (int \*\*arr, int x, int y, int length\_square)* | *arr* – массив клеток квадрата  *x* – координата х квадрата  *y* – координата y квадрата  *length\_square* – длина квадрата | Инициализирует квадрат в массиве в соответствии с параметрами х, у и length\_square |
| *int find\_empty\_cell (int \*\*arr, int &x, int &y, int N)* | *arr* – массив клеток квадрата  *x* – координата х квадрата  *y* – координата y квадрата  *N* – размер массива | Находит пустую клетку в массиве arr и переводит значения x и y в соответствии с ее координатами. Возвращает 1 в случае, когда нет пустых клеток, и 0 в обратном. |
| *void stack\_copy*  *(stack < Square> \*sqrs,*  *stack < Square>\*sqrs\_copy)* | *sqrs –* стек для копирования  *sqrs\_copy* – копируеммый стек | Копирует стек. |
| *void empty\_stack*  *(stack < Square> \*sqrs)* | *sqrs –* опустошаемый стек | Опустошает стек. |
| *int find\_min\_sqrs(int limit* *, stack <Square> \*sqrs, int \*\*arr, int N, int count\_rec)* | *limit –* предельное значение счетчика рекурсии  *sqrs –* стек для квадратов  *arr* – массив клеток квадрата  *N –* размер массива  *Count\_rec –* счетчик | Рекурсивная функция для перебора возможных значений расстановки квадратов.  Возвращает найденное количество квадратов. |
| *void clear\_square(int \*\*arr, int x, int y, int length\_square)* | *arr –* массив клеток квадрата  *x* – координата х квадрата  *y* – координата y квадрата  *length\_square –* длина квадрата | Очищает квадрат длиной *length\_square* с координатами *х* и *у.* |
| *int more\_square(int \*\*arr, int x, int y, int length\_square)* | *arr –* массив клеток квадрата  *x* – координата х квадрата  *y* – координата y квадрата  *length\_square –* длина квадрата | Проверяет можно ли вместить квадрат длиной *length\_square* с координатами *х* и *у* в массив *arr.* |

**Тестирование.**

|  |  |  |
| --- | --- | --- |
| Номер  теста | Тест: | Результат: |
| 1 | 2 | 4  1 1 1  1 2 1  2 1 1  2 2 1 |
| 2 | 6 | 4  1 1 3  1 4 3  4 1 3  4 4 3 |
| 3 | 13 | 11  1 1 7  1 8 6  8 1 6  7 8 1  7 9 3  7 12 2  8 7 2  9 12 2  10 7 4  10 11 1  11 11 3 |
| 4 | 51 | Error |
| 5 | 37 | 37  15  1 1 19  1 20 18  20 1 18  19 20 11  9 21 3  19 24 7  19 31 7  20 19 2  22 19 5  26 24 2  26 26 12  27 19 4  27 23 1  28 23 3  31 19 7 |
| 6 | 25 | 8  1 1 5  1 6 5  1 11 5  1 16 10  6 1 10  6 11 5  11 11 15  16 1 10 |
| 7 | 33 | 6  1 1 11  1 12 11  1 23 11  12 1 11  12 12 22  23 1 11 |

**Вывод.**

Я ознакомилась с алгоритмом перебора с возвратом и научилась применять его на практике.

**Приложение 1. Код программы.**

**Kvadrat**[**.c**](https://github.com/makometr/AiSD/pull/37/files#diff-240c2792fd92b595432e18629f6e16b6)**pp**

#include <iostream>

#include <cmath>

#include <stack>

#include <ctime>

using namespace std;

struct Square{ //Структура квадрата

int x;

int y;

int length;

};

int multiplicity(int N){//Проверка кратнности

if(N%2 == 0 && N!=2)

return 2;

if(N%3 == 0 && N!=3)

return 3;

if(N%5 == 0 && N!=5)

return 5;

return 1;

}

void demo(int \*\*arr, int N){//Распечатка квадрата

cout << "Текущее состояние:" << endl;

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

for(int l = 0; l < N; l++)

cout << arr[m][l]<< ' ';

cout << endl;

}

cout << endl;

}

void fill\_square(int \*\*arr, int x , int y, int length\_square){//Раскрашивает квадрат

for(int i = 0; i < length\_square; i++)

for(int j = 0; j < length\_square; j++)

arr[x+i][y+j] = length\_square;

}

void clear\_square(int \*\*arr, int x, int y, int length\_square){//Очищает квадрат

for(int i = 0; i < length\_square; i++)

for(int j = 0; j < length\_square; j++)

arr[x+i][y+j] = 0;

}

int more\_square(int \*\*arr, int x, int y, int length\_square){//Проверяет возможность вместить квадрат определенной длины

for(int i = 0; i < length\_square; i++)

if(arr[x + length\_square - 1][y + i] || arr[x + i][y + length\_square - 1])

return 1;

return 0;

}

void stack\_copy(stack <Square> \* sqrs, stack <Square> \* sqrs\_copy){//Копирвание стека

while(!sqrs->empty()){

sqrs\_copy->push(sqrs->top());

sqrs->pop();

}

}

int find\_empty\_cell(int \*\*arr, int &x, int &y, int N){ // Нахождение пустой клетки

for(x = 0; x < N; x++){

for(y = 0; y < N - 1; y++)

if(arr[x][y] == 0)

break;

if(arr[x][y] == 0)

break;

}

if(x == N) x = N-1;

if(arr[x][y] != 0)

return 1;

return 0;

}

void empty\_stack(stack <Square> \*sqrs){

while(!sqrs->empty())

sqrs->pop();

}

int find\_min\_sqrs(int limit, stack <Square> \*sqrs, int \*\*arr, int N, int count\_rec, int scale){ //Находит рекурсивно минимальное распледеление квадрата

if(limit < count\_rec) //Условие выхода за пределы

{

cout << endl << endl << "Выход за пределы." << endl;

return -1;

}

else

cout << "Поставим следующий квадрат." << endl;

demo(arr, N); //Вывод заполненности квадрата

int x, y;

if(find\_empty\_cell(arr, x, y, N)) // Нахождение пустой клетки

{

cout << "Пустых клеток не осталось." << endl;

return 0;

}

else

cout << "Найдена пустая клетка: х = " << x << ", y = "<< y << endl;

stack <Square> sqrs\_tmp; //Временный стек квадратов

stack <Square> sqrs\_max; //Итоговый стек квадратов

int distance\_to\_the\_border = (N-x > N-y) ? (N-y-!arr[0][0]) : (N-x-!arr[0][0]); //Хранит расстояние до границы

cout << "Расстояние до границ = " << distance\_to\_the\_border << endl;

int length\_square, k\_min = limit + 1, k, need\_length = 1;

cout << "Перебер длин квадрата:" << endl;

for(length\_square = 1; length\_square <= distance\_to\_the\_border; length\_square++){ //Пробегает по длине квадрата

cout << "Текущая длина: " << length\_square << endl;

cout << "Заполним квадрат. ";

fill\_square(arr, x, y, length\_square); //Заполняет квадрат

k = find\_min\_sqrs(limit, &sqrs\_tmp, arr, N, count\_rec+1, scale) + 1;//Хранит количество размещенных квадратов

if(k < k\_min && k != 0){ //Если надено распределение с меньшим количеством квадратов сохраним стек квадратов

cout << "Нашли распределение лучше, чем " << limit + 1 << ". Сохраним его." << endl;

k\_min = k; //Сохраняем минимальное распеределение

need\_length = length\_square;

while(!sqrs\_max.empty()){

sqrs\_max.pop();

}

stack\_copy(&sqrs\_tmp, &sqrs\_max);

}

else //Иначе очищаем временный стек

{

cout << "Было найдено распределение по-лучше, очистим стек квадратов." << endl;

empty\_stack(&sqrs\_tmp);

}

cout << "Сотрем квадрат длинной = " << length\_square << " в точке х = " << x << ", y = " << y << "." << endl;

clear\_square(arr, x, y, length\_square); //Стираем нарисованный квадрат

demo(arr, N);

if(distance\_to\_the\_border - length\_square - 1 < 0) break;

if(more\_square(arr, x, y, length\_square + 1)) // Если нельзя вместить квадрат побольше, выходим

{

cout << "Квадрат побольше вместить не получилось." << endl;

break;

}

}

while(!sqrs->empty())

sqrs->pop();

stack\_copy(&sqrs\_max, sqrs); //Копируем стек в основнй стек

sqrs->push({x\*scale+1, y\*scale+1, need\_length\*scale}); //И добавляем текущий квадрат

return k\_min;

}

int main(){

setlocale(LC\_ALL, "rus");

int N, min\_k;

cout << "Введите длину квадрата, для которого вы хотите получить минимальное распределение" << endl;

cin >> N;

unsigned int time1 = clock();

if(!(N>=2 && N<=40)){

cout << "Ошибка, введено неверное число." << endl; return 0;

}

stack <Square> sqrs;

int mul = multiplicity(N);

int limit = 6.37 \* sqrt(sqrt((mul == 1) ? N : mul)); //Задаем предельную величину для количества квадратов

int \*\*arr ;

if(mul!=1){

arr = new int \*[mul];

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

arr[i] = new int[mul];

for(int m = 0; m < mul; m++)

for(int l = 0; l < N; l++)

arr[m][l] = 0;

min\_k = find\_min\_sqrs(limit, &sqrs, arr, mul, 0, N/mul);

}

else {

arr = new int \*[N];

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

arr[i] = new int[N];

for(int m = 0; m < N; m++)

for(int l = 0; l < N; l++)

arr[m][l] = 0;

// int half = N-N/2;

// fill\_square(arr, 0, half);

// fill\_square(arr, half, half - 1);

// fill\_square(arr, 0, half - 1);

min\_k = find\_min\_sqrs(limit/\* - 3\*/, &sqrs, arr, N, 0, 1)/\* + 3\*/;

// sqrs.push({half + 1, 1, half - 1});

// sqrs.push({1, half + 1, half - 1});

// sqrs.push({1, 1, half});

}

cout << "Количество = " << min\_k << endl;

Square tmp;

cout << "Минимальное распределение квадратов:" << endl;

for(int j = 0; j < min\_k; j++){

tmp = sqrs.top();

cout << j + 1 << ") x = " << tmp.x << ", y = " << tmp.y << ", длина = " << tmp.length << endl;

sqrs.pop();

}

unsigned int time2 = clock();

cout << "TIME : " << time2-time1 << endl;

delete [] arr;

return 0;

}