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: Перебор с возвратом

Студентка гр. 1304 _____ Виноградова М.О.

Преподаватель _____ Шевелева А.М.

Санкт-Петербург
2023

Цель работы.

Написать программу, которая ищет минимальное количество обрезков(квадратов), из которых можно построить столешницу(квадрат) заданного размера N, и координаты левого верхнего угла и длину стороны каждого обрезка(квадрата).

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. Пример представлен на рис. 1.

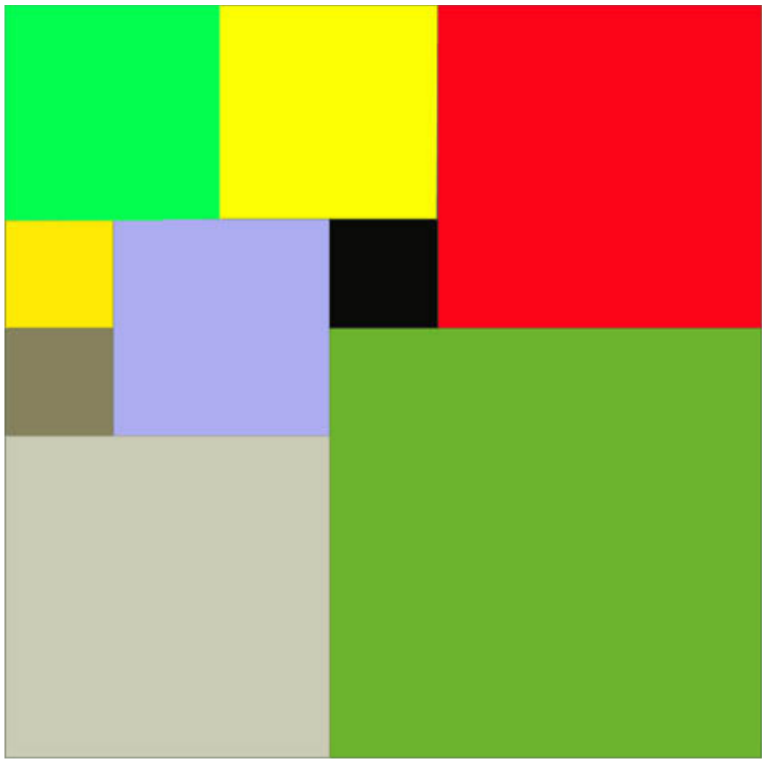

Рисунок 1 – пример квадрата 7×7

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 ($2 \leq N \leq 20$).

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 ($1 \leq x,y \leq N$) и длину стороны соответствующего обрезка(квадрата).

Выполнение работы.

Структура **Square** хранит в себе координаты (x,y) и длину(len) каждого обрезка(квадрата).

Структура **Matrix** хранит в себе двумерный массив (x), количество добавленных квадратов(color), также определяет цвет в который будет покрашен следующий обрезок при вставке (color+1) и свободную площадь столешницы.

Для всех структур реализованы конструкторы.

Переменная n – хранит длину стороны.

Переменная record – хранит минимальное количество квадратов (начальная оценка 2*n+1, случай самого большого квадрата и 2*n-1 единичного квадрата, добавление единицы необходимо так как используется строгое сравнение для получения лучшего решения).

Переменная max_len – максимальная длина стороны для рассмотрения, равна половине длины стороны так как после оптимизации необходимо рассматривать только четверть столешницы.

Существует три возможных случая для размера стороны столешницы

1) Сторона - четное число:

В данном случае оптимальным количеством квадратов будет 4, позиции которых вычисляются в зависимости от длины стороны. Позиция первого квадрата (1,1), позиция второго и третьего квадратов соответственно (1, n/2+1) и (n/2+1,1). Позиция последнего квадрата (n/2+1, n/2+1). Стороны всех квадратов равны n/2.

Для реализации данного случая написана функция **evenNumber**(), которая если сторона четное число выводит ответ и возвращает true, иначе false.

2) Сторона - простое число (нечетное так как все четные случаи рассматриваются в пункте 1):

Для нахождения решения был использован метод ветвей и границ.

Функции **isCorrect(**Matrix matr,int start_x,int start_y,int len**)** проверяется можно ли поставить квадрат с длиной стороны len на позицию start_x и start_y. Для этого проверяется, свободна ли переданная позиция (то есть значение элемента в массиве по данным координатам равно 0). Далее проверяется, что пространство для вставки не занято другими квадратами с помощью обхода по единичному контору вставляемого квадрата.

Функция **addSquare**(Matrix matr,int start_x,int start_y,int len) осуществляет вставку квадрата со стороной len на позицию start_x start_y. Увеличивает количество цветов на единицу и уменьшает площадь на площадь вставленного квадрата.

Для поиска позиции для вставки используется функция **startPosition**(vector<vector<int>> arr), которая возвращает координаты первой не занятой клетки (при обходе переданный двумерный массив рассматривается сверху вниз, слево направо). Для оптимизации рассмотрение начинается с середины столешницы (n/2, n/2) и рассматривается только правая нижняя четверть столешницы.

Для тестирования и отладки использовалась функция **print_sq(**vector<vector<int>> arr**)**, которая выводит содержимое переданного двумерного массива.

Функция **optimisation**(Matrix& matr) добавляет в матрицу три квадрата. Один на позицию (0,0) со стороной n/2+1, и два квадрата справа и снизу от данного, со сторонами n/2. Данная оптимизация необходима для более быстрой работы программы (вместо целой столешницы рассматривается только четверть).

В функции **bandp**(Matrix matr, vector<vector<int>> P) реализован метод ветвей и границ. Данный метод является рекурсивным. В функцию передается текущее решение (Matrix) и лучшее решение (двумерный массив). Если текущее решение является решением задачи (то есть вся площадь столешницы закрашена), а также если оно лучше ранее найденного, то данные обновляются (минимальное количество квадратов record и матрица столешницы P). Если текущее решение не является решением задачи, то рассматриваются все возможные расширения данного решение. То есть в цикле по длинам (от max_len до 1) рассматриваются все возможные вставки квадратов на первую свободную позицию. Если удалось вставить квадрат, то функция вызывается рекурсивно.

3) Сторона - составной число (также нечетное):

В данном случае необходимо найти наименьший простой делитель(min_devider) не равный 1 и найти решение для него. В ответе необходимо помножать полученные координаты и длину квадратов на коэффициент равный n/min_devider.

Для этого реализована функция **compositNumber**(), которая находит коэффициент (если сторона не составное число, то возвращает единицу) и минимальный простой делитель. N становится равным данному делителю.

Для вывода ответа для случаев 2 и 3 реализованы функции:

initSquareList(vector<vector<int>> P) – которая переданный двумерный массив преобразует в массив с элементами типа Square.

printAnswer(vector<Square>, int coefficient) – выводит на экран

количество квадратов и сами координаты с длиной, помножая на коэффициент и прибавляя единицу.

Код приведен в приложении А.

Выводы.

Проанализированы различные варианты развития событий в задаче: сторона столешницы – четное число, нечетное простое число и нечетное составное число. Для решения задачи был применен метод ветвей и границ. Также были найдены различные способы оптимизации решения, такие как: начальная вставка трех квадратов (для нечетного случая), предварительная оценка решения, предварительная оценка максимально возможной длины стороны квадрата. В результате написаны необходимые для реализации программы функции и структуры. Выполнена задача лабораторной работы, то есть найдено оптимальное заполнение столешницы со стороной N, квадратами со стороной от 1 до N-1.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include<vector>
#include "math.h"
#include <algorithm>
using namespace std;

//структура для хранения ответа в формате (позиция x, позиция y, длина
стараны квадрата len)
struct Square{
    int x;
    int y;
    int len;
public:
    Square(int x,int y,int len)
    {
        this->x=x;
        this->y=y;
        this->len=len;
    }
};
//структура для хранения столешницы
// столешница - двумерный массив
// color - количество уже добавленных квадратов
// S - незаполненая площадь столешницы
struct Matrix{
    vector<vector<int>> x;
    int color;
    int S;
public:
    Matrix(vector<vector<int>> x,int n)
    {
        this->x=x;
        this->color=0;
        this->S=n*n;
    }
};

//глобальные переменные
// n - сторона квадрата, которую вводит пользователь
// max_len - максимальная длина стороны квадрата
// record - переменная для фиксация лучшего решения
static int n;
static int max_len;
static int record;

//функция поиска позиции для вставки
pair<int, int> startPosition(vector<vector<int>> x){
    for(int i=n/2;i<n;i++){
        for(int j=n/2;j<n;j++){
            if(x[i][j]==0){
```

```
                return {i,j};
            }
        }
    }
    return{-1,-1};
}
//функция вставки квадрата на переданную позицию
Matrix addSquare(Matrix matr,int start_x,int start_y,int len){
    for(int i=start_x;i<start_x+len;i++){
        for(int j=start_y;j<start_y+len;j++){
            matr.x[i][j]=matr.color+1;
        }
    }
    matr.color++;
    matr.S-=len*len;

    return matr;
}
// функция проверки на корректность
bool isCorrect(Matrix matr,int start_x,int start_y,int len){
    if(matr.x[start_x][start_y]!=0 )
        return false;
    if(start_x+len<=n && start_y+len<=n){
        for(int i=start_x;i<start_x+len;i++){
            if(matr.x[i][start_y]!=0 || matr.x[i][start_y+len-1]!=0)
                return false;
        }
        for(int j=start_y;j<start_y+len;j++){
            if(matr.x[start_x][j]!=0 || matr.x[start_x+len-1][j]!=0)
                return false;
        }
    }else
        return false;

    return true;

}
//дополнительная функция для вывода матрицы
void print_sq(vector<vector<int>> arr){
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            cout<<arr[i][j]<<" ";
        }
        cout<<"\n";
    }
    cout<<"\n";
}

// функция реализующая метод ветвей и границ
//если вся плита запонена и ответ оптимальнее ранее найденого, то фикси-
руется новое решение
//если ответ не найден и текущее решение меньше оптимального, то рассмат-
риваются все дополнения текущего решения
void bandb(Matrix matr,vector<vector<int>>&P){

    if(matr.S == 0 && matr.color<record){//решение
        record=matr.color;
        P=matr.x;
```

```
        return;
    }else{
        if(matr.color<record){//расширение текущего решения
            for(int len=max_len;len>=1;len--){
                pair<int,int>pos = startPosition(matr.x);
                if(pos.first==-1)
                    return;
                if(isCorrect(matr,pos.first,pos.second,len)){
                    bandb(addSquare(matr,pos.first,pos.second,len),P);
                }
            }
        }
    }

}
//функция для случая с четной стороной
bool evenNumber(){
    if(n%2==0){
        cout<<"4\n";
        cout<<1<<" "<<1<<" "<<n/2<<"\n";
        cout<<1<<" "<<n/2+1<<" "<<n/2<<"\n";
        cout<<n/2+1<<" "<<1<<" "<<n/2<<"\n";
        cout<<n/2+1<<" "<<n/2+1<<" "<<n/2;
        return true;
    }
    return false;
}
//формирование вектора ответа
vector<Square> initSquareList(vector<vector<int>> P){
    vector<Square> sq;
    for(int c=1;c<=record;c++){
        pair<int,int> pos(-1,-1);
        int S=0;
        for(int i=0;i<n;i++){
            for (int j=0;j<n;j++){
                if(c==P[i][j]){
                    S++;
                    if(pos.first==-1 && pos.second==-1){
                        pos.first = i;
                        pos.second = j;
                    }
                }
            }
        }
        sq.push_back(Square(pos.first,pos.second,sqrt(S)));
    }
    return sq;
}
//вывод ответа
void printAnswer(vector<Square> sq, int coefficient){
    cout<<record<<"\n";
    for(int i=0;i<record;i++){//тк индексация начинается с нуля, необхо-
димо прибавить единицу
       cout<<sq[i].x*coefficient+1<<"         "<<sq[i].y*coefficient+1<<"
"<<sq[i].len*coefficient<<"\n";
    }
}
//начальная оптимизация матрицы
```

```
void optimisation(Matrix& matr){
    matr = addSquare(matr,0,0,n/2+1);
    matr = addSquare(matr,n/2+1,0,n/2);
    matr = addSquare(matr,0,n/2+1,n/2);
    max_len = n/2;
}
//функция возвращает коэффициент домножения
int compositNumber(){
    vector<int> prime_list = {2,3,5,7,11,13,17,19,23,29,31,37};
    int coefficient=1;
    if(!count(prime_list.begin(),prime_list.end(),n)) {


        int min_devider;
        for (int i = 0; i < prime_list.size(); i++) {
            if (n % prime_list[i] == 0) {
                min_devider = prime_list[i];
                break;
            }
        }
        coefficient = n / min_devider;
        n = min_devider;//минимальный простой делитель
    }
    return coefficient;
}
int main(){
    //ввод пользователем длины стороны
    cin>>n;
    //Случай для четной стороны
    if(evenNumber())
        return 0;

    vector<vector<int>> x(n,vector<int>(n,0));//текущее решение
    vector<vector<int>> P(n,vector<int>(n,0));//минимальное решение

    record = 2*n+1;//начальная оценка количества квадратов

    //случай, когда сторона составное число
    int coefficient=compositNumber();

    //инициализация структуры
    Matrix matr(x,n);

    //начальная оптимизация
    optimisation(matr);

    //метод ветвей и границ
    bandb(matr,P);

    //вывод ответа
    printAnswer(initSquareList(P),coefficient);

    return 0;
}
```