МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №1
по дисциплине «Построение и анализ алгоритмов»
Тема: Рекурсивный бэктрекинг

Студент гр. 8382 _____ Ершов М.И.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2020

Цель работы.

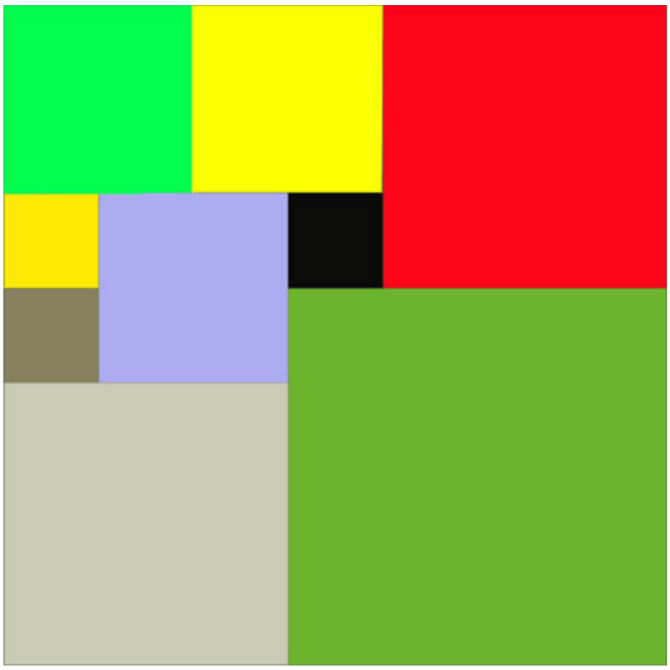
Приводится цель работы в соответствии с методическими указаниями.

Задание.

Вар. 2р. Рекурсивный бэктрекинг. Исследование времени выполнения от размера квадрата.

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

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).

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.

Нахождение минимального количества обрезков для заполнения «столешницы» размера N реализует рекурсивный алгоритм reccFinding(), который принимает на вход указатель на «столешницу», площадь текущего обрезка, оставшуюся полезную площадь «столешницы», количество уже вставленных обрезков и массив этих обрезков.

Рекурсивный алгоритм вызываем итеративно в цикле от K до 1, где K – это размерность очередного обрезка, равная (размер «столешницы»)/2.

После вызова функции пытаемся вставить обрезок размера K в первое свободное место (идем слева направо, сверху вниз). Получилось – вызываем рекурсивно и снова пытаемся вставить обрезок размера K. Не получилось – пробуем для K-1. Заканчиваем, если:

- Столешница полностью заполнена.

В этом случае проверяем – является ли получившееся число обрезков оптимальным (сравниваем с 2*N+1)? Если да – сохраняем получившееся решение как оптимальное и откатываемся до момента, где есть возможность вставить вместо обрезка L обрезок L-1. В случае, если число обрезков не оптимальное – просто делаем откат до аналогичного случая.

- Решение не оптимальное уже на текущем этапе

В этом случае аналогично делаем откат до обрезка, который можно заменить.

Описание функций и структур данных.

Структура Square используется для хранения обрезков в стиле задания – координаты левой верхней точки и размер стороны.

«Столешница» - двумерный массив int, реализованный через vector<vector<int>>. На месте вставленного обрезка каждое число меняется на площадь обрезка.

Функция init инициализирует двумерный массив int нулями.

Функция printSquare печатает «столешницу» с текущим расположением обрезков.

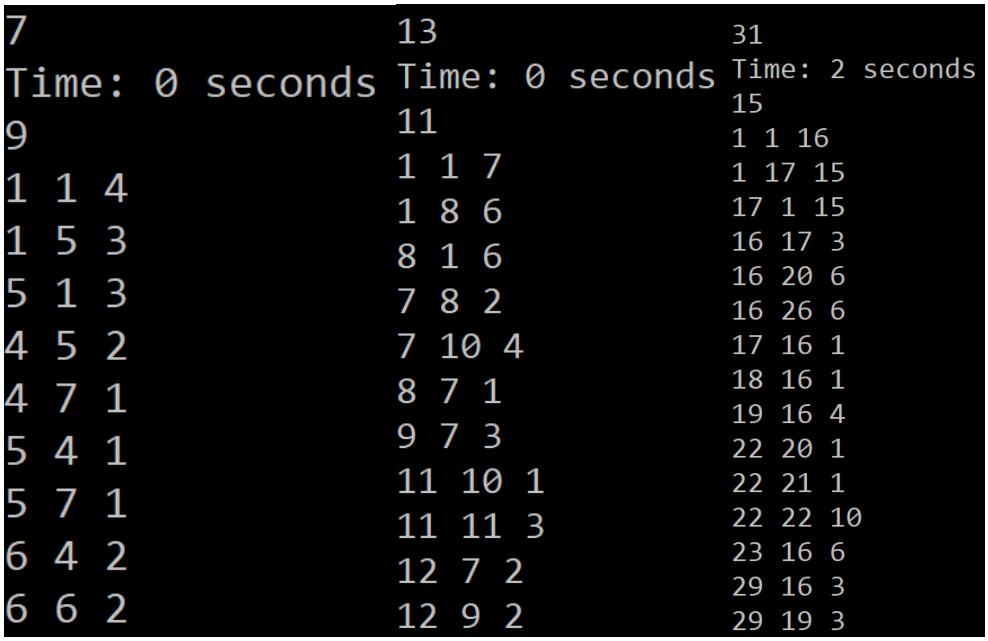
Функция deletePiece удаляет последний вставленный обрезок из «столешницы».

Функция putPiece вставляет заданный обрезок в заданные координаты.

Функция putFirstThree вставляет 3 стартовых обрезка в «столешницу».

Функция reccFinding – см. выше.

Тестирование.



В ходе тестирования ожидаемые результаты полностью совпали с результатами выполнения программы.

Исследование.

В ходе исследования времени выполнения было выявлено, что при вводе в качестве размера «столешницы» простых чисел, время выполнения программы увеличивается экспоненциально с увеличением чисел. Например, при N = 29 время выполнения близко к 1 секунде, при N = 31 – 2 секунды, N=37 – 16 секунд, N=41 – 85 секунд (ограничение N <= 40 было сознательно проигнорировано для проведения исследования).

При росте не простых чисел такого увеличения не наблюдается, так как для подобных чисел реализована оптимизация – сведение больших чисел к более простым случаям.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#include <iostream>
#include <sstream>
#include <windows.h>
#include <clocale>
#include <iostream>
#include <vector>
#include <ctime>

#define UNDERLINE "\033[4m"
#define CLOSEUNDERLINE "\033[0m"

using std::cin;
using std::cout;
using std::endl;
using std::vector;

struct Square {
    int x;
    int y;
    int size;
};

vector<Square> minArrOfSquares;
int n;
int minNumOfSquares;

void init(vector<vector<int>> &square) {
    for(int i = 0; i < n; i++) {
        square.resize(n);
        for(int j = 0; j < n; j++) square[i].push_back(0);
    }
}

void printSquare(vector<vector<int>> &square, int k = 1) {
     int r = 3;
     int flag = 1;
    for(int i = 0; i < n*k; i++) {
        for(int j = 0; j < n*k; j++) {
            cout.width(r);
            if (square[i][j + 1] != square[i][j]) {
                cout << square[i][j] << "|";
                    if (flag) {
                        r--;
                        flag = 0;
                    }
                } else {
                    cout << square[i][j];
                    r = 3;
                    flag = 1;
                }
```

```
            cout.width(r);
        }
        cout << endl;
        r = 3;
    }
    cout << endl;
}

void deletePiece(vector<vector<int>> &square, vector<Square> &arrOfSquares)
{
    Square tmp = *(arrOfSquares.rbegin());
    arrOfSquares.pop_back();
    for(int i = tmp.x; i < tmp.x + tmp.size; i++) {
        for(int j = tmp.y; j < tmp.y + tmp.size; j++) {
            square[i][j] = 0;
        }
    }

}

int putPiece(vector<vector<int>> &square, int x, int y, int size) {
     if(n-x < size || n-y < size) return 0;

    for(int i = x; i < x+size; i++) {
        for(int j = y; j < y+size; j++) {
            if(square[i][j] != 0) return 0;
        }
    }

    for(int i = x; i < x+size; i++) {
        for(int j = y; j < y+size; j++) {
            square[i][j] = size;
        }
    }

    return 1;
}

void putFirstThree(vector<vector<int>> &square, vector<Square>
&arrOfSquares, int& activeArea) {
    arrOfSquares.push_back({0, 0, (n+1)/2});
    putPiece(square, 0, 0, (n+1)/2);
    activeArea -= ((n+1)/2)*((n+1)/2);

    arrOfSquares.push_back({0, (n+1)/2, n/2});
    putPiece(square, 0, (n+1)/2, n/2);
    activeArea -= (n/2)*(n/2);

    arrOfSquares.push_back({(n+1)/2, 0, n/2});
    putPiece(square, (n+1)/2, 0, n/2);
    activeArea -= (n/2)*(n/2);
}
```

```
void reccFinding(vector<vector<int>> &square, int curSquare, int activeArea,
int countOfSquares, vector<Square> &arrOfSquares) {
    if(countOfSquares + 1 == minNumOfSquares && activeArea > curSquare *
curSquare) return;

    int flag = 0;
    for(int i = 0; i < n && !flag; i++) {
        for(int j = 0; j < n && !flag; j++) {
            if(square[i][j] == 0) {
                if(putPiece(square, i, j, curSquare)) {
                    flag = 1;
                    //cout << "putPiece square with width " << curSquare <<
" to x = " << i << " and y = " << j << endl;
                    //printSquare(square);
                    arrOfSquares.push_back({i, j, curSquare});
                } else return;
            } else j += square[i][j] - 1;
        }
    }

    if(countOfSquares + 1 == minNumOfSquares) {
      //cout << countOfSquares + 1 << " is equal to " << minNumOfSquares <<
endl;
        deletePiece(square, arrOfSquares);
        return;
    }

    if(activeArea == curSquare*curSquare && countOfSquares + 1 <
minNumOfSquares) {
        minNumOfSquares = countOfSquares + 1;
        minArrOfSquares.assign(arrOfSquares.begin(), arrOfSquares.end());
        //cout << "New optimal!" << endl;
        //printSquare(square);
        //Square tmp = *(arrOfSquares.rbegin());
        //cout << "Deleting " << tmp.x << " " << tmp.y << " " << tmp.size <<
endl;
        deletePiece(square, arrOfSquares);
        //printSquare(square);
        return;
    }

    for(int i = n/2; i >= 1; i--) {
        if(i*i <= activeArea) {
           //cout << "Trying for " << i << " inside a function" << endl;
            reccFinding(square, i, activeArea - curSquare*curSquare,
countOfSquares + 1, arrOfSquares);
        }
    }

    deletePiece(square, arrOfSquares);
}

int main() {
```

```
      while(true) {

          cin >> n;
          int k = 1;

          if((n < 2) || (n > 50)) {
              cout << "Error!" << endl;
              return 0;
          }

          vector<vector<int>> square;
          vector<Square> arrOfSquares;

          init(square);

          vector<vector<int>> result;
          init(result);

          if (n % 2 == 0) {
              k = n/2;
              n = 2;
            }

            if(n % 3 == 0) {
              k = n/3;
              n = 3;
            }

            if(n % 5 == 0) {
              k = n/5;
              n = 5;
          }

          int activeArea = n*n;
          minNumOfSquares = 2*n+1;

          putFirstThree(square, arrOfSquares, activeArea);

          time_t start = clock();
          for(int i = n/2; i >= 1; i--) {
            //cout << "Trying for " << i << endl;
            //printSquare(square);
              reccFinding(square, i, activeArea, arrOfSquares.size(),
arrOfSquares);
          }
          time_t end = (clock() - start) / CLK_TCK;
          cout << "Time: " << end << " seconds" << endl;

          cout << minNumOfSquares << endl;
          for(auto& i : minArrOfSquares) {
              cout << k*i.x + 1 << " " << k*i.y + 1 << " " << k*i.size <<
endl;
              putPiece(result, k*i.x, k*i.y, k*i.size);
```

```
            }

            printSquare(result, k);
        }
    return 0;
}
```