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. 7383 _____ Корякин М.П.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2019

Цель работы.

Изучить и реализовать на языке программирования C++ поиск с возвратом для задачи с минимальным количеством отрезков, помещающихся в столешницу.

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).

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 ($2 \leq N \leq 40$).

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, w, задающие координаты левого верхнего угла ($1 \leq x,y \leq N$) и длину стороны соответствующего обрезка (квадрата).

Описание работы алгоритма.

Для быстроты работы алгоритма предусмотрена градация сторон квадрата: для четных сторон квадрата, для сторон квадрата кратных трем, кратных пяти предусмотрены отдельные функции, которые более быстро осуществляют расстановку отрезков в столешнице.

Для более сложных случаев, а именно для стороны квадрата, которая определена простым числом предусмотрен алгоритм основной алгоритм «бэктрэкинга». Изначально мы заполняем наш квадрат тремя большими квадратами, и в минимальное количество квадратов записываем максимально возможное количество. Далее программа следует одному алгоритму: мы находим в квадрате точку, от которой мы

будем заполнять нашу столешницу отрезками (три больших квадрата, установленные на прошлом шаге, будут определены по дефолту в любом случае прогонки алгоритма «бэктрэкинга»). После чего мы заполняем квадратную область столешницы максимально возможным квадратом, после чего снова ищем в столешнице свободную область для заполнения и снова заполняем ее максимально возможным квадратом. По итогу заполнения нашей столешнице квадратами мы сравниваем количество квадратов в ней с минимальным значением (которое изначально нарочно было сделано максимальным) квадратов в столешнице, и меняем его, если новое значение оказалось меньше. После этого программа сохраняет текущие координаты отрезков в столешнице, и алгоритм начинает рекурсивно откатываться назад для полного перебора отрезков: т.е. там где он вставил максимальный отрезок – вставит меньший отрезок. Таким образом будут перебраны все вариации расстановки отрезков и выявлена наилучшая для решения данной задачи.

Результат работы программы.

Входные данные	Результат
7	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
5	8 0 0 2 0 2 3 2 0 1 2 1 1 3 0 1 3 1 2 3 3 2 4 0 1
37	15 0 0 19 0 19 18 18 19 2 18 21 5 18 26 11

	19 0 18 19 18 1 20 18 3 23 18 8 29 26 3 29 29 8 31 18 6 31 24 1 31 25 1 32 24 5
21	6 0 0 14 0 14 7 7 14 7 14 0 7 14 7 7 14 14 7

Выводы.

В ходе написание программы на языке C++ был изучен и реализован алгоритм «Поиск с возвратом». Данный алгоритм был применен для решении задачи с нахождением минимального количества квадратных отрезков, которые полностью заполнят квадратную столешницу.

ПРИЛОЖЕНИЕ А.
ИСХОДНЫЙ КОД.

```
#include <iostream>
using namespace std;
struct Point{
   unsigned i;
   unsigned j;
};

class Foursquare{
private:
   unsigned num;
   unsigned** arr;
   unsigned** framing_arr;

public:
   void framing_odd(){
      this->set_main_sq();
      unsigned min = (num*num - 2*( (num/2) * (num/2)) - ( ((num+1)/2) *
((num+1)/2 )))+3;
      this->b_track(3, min);
      cout << min << endl;
      print_coordinates();
   }

   void framing_even(){
      for(unsigned i=0; i<num/2; i++){
         for(unsigned j=0; j<num/2; j++)
            framing_arr[i][j] = num/2;
      }

      for(unsigned i=num/2; i<num; i++){
         for(unsigned j=0; j<num/2; j++)
            framing_arr[i][j] = num/2;
      }

      for(unsigned i=0; i<num/2; i++){
         for(unsigned j=num/2; j<num; j++)
            framing_arr[i][j] = num/2;
      }

      for(unsigned i=num/2; i<num; i++){
         for(unsigned j=num/2; j<num; j++)
            framing_arr[i][j] = num/2;
      }
```

```
    cout << 4 << endl;
    print_coordinates();
}

void framing3(unsigned k){
    for(unsigned i=0; i<(k/3)*2; i++){
        for(unsigned j=0; j<(k/3)*2; j++)
            framing_arr[i][j] = (k/3)*2;
    }
    for(unsigned i=(k/3)*2; i<k; i++){
        for(unsigned j=0; j<k; j++)
            framing_arr[i][j] = k/3;
    }
    for(unsigned i=0; i<(k/3)*2; i++){
        for(unsigned j=(k/3)*2; j<k; j++)
            framing_arr[i][j] = k/3;
    }
    cout << 6 << endl;
    print_coordinates();
}

void framing5(unsigned k){
    unsigned one = (k/5)*2;
    unsigned two = (k/5)*3;
    for(unsigned i=0; i<one; i++){
        for(unsigned j=0; j<one; j++)
            framing_arr[i][j] = one;
    }

    for(unsigned i=0; i<two; i++){
        for(unsigned j=one; j<k; j++)
            framing_arr[i][j] = two;
    }

    for(unsigned i=two; i<k; i++){
        for(unsigned j=one/2; j<k; j++)
            framing_arr[i][j] = one;
    }

    for(unsigned i=one; i<two; i++){
        for(unsigned j=0; j<one; j++)
            framing_arr[i][j] = one/2;
    }

    for(unsigned i=two; i<k; i++){
        for(unsigned j=0; j< (k-(one*2)); j++)
```

```
            framing_arr[i][j] = one/2;
        }

        cout << 8 << endl;
        print_coordinates();
    }

    void b_track(unsigned count, unsigned& min){
        if(min <= count)
            return;

        Point tmp = this->find_NoClrLeft();
        if(tmp.i == num && tmp.j == num){
            if(min > count){
                min = count;
                for(unsigned j=0; j<num; j++){
                    for(unsigned k=0; k<num; k++)
                        framing_arr[j][k] = arr[j][k];
                }
            }

            return;
        }

        for(unsigned i=num/2; i>0; i--){
            if(is_fit(tmp, i)){
                count++;
                for(unsigned j=tmp.i; j<tmp.i+i; j++){
                    for(unsigned k=tmp.j; k<tmp.j+i; k++)
                        arr[j][k] = i;
                }

                b_track(count, min);
                count--;
                for(unsigned j=tmp.i; j<tmp.i+i; j++){
                    for(unsigned k=tmp.j; k<tmp.j+i; k++)
                        arr[j][k] = 0;
                }
            }
        }
    }

    bool is_fit(Point p, unsigned sz){
        for(unsigned i=p.i; i<p.i+sz; i++){
            for(unsigned j=p.j; j<p.j+sz; j++){
                if(i==num || j==num || arr[i][j])
                    return false;
```

```
            }
        }

        return true;
    }

    void set_main_sq(){
        for(unsigned i=0; i<(num+1)/2; i++){
            for(unsigned j=0; j<(num+1)/2; j++)
                arr[i][j] = (num+1)/2;
        }

        for(unsigned i=num-1; i>num/2; i--){
            for(unsigned j=0; j<num/2; j++)
                arr[i][j] = num/2;
        }

        for(unsigned i=0; i<num/2; i++){
            for(unsigned j=num-1; j>num/2; --j)
                arr[i][j] = num/2;
        }
    }

    void print_coordinates(unsigned k=1, Point cur_pos={0,0}){
        for(unsigned i=0; i<num; i++){
            for(unsigned j=0; j<num; j++){
                if(framing_arr[i][j]){
                    cout << i*k << ' ' << j*k << ' ' << k*framing_arr[i][j] << endl;
                    delete_fr_sq({i, j}, framing_arr[i][j]);
                }
            }
        }
    }

    void delete_fr_sq(Point st, unsigned sz){
        for(unsigned j=st.i; j<st.i+sz; j++){
            for(unsigned k=st.j; k<st.j+sz; k++)
                framing_arr[j][k] = 0;
        }
    }

public:
    Foursquare(unsigned sz){
        num = sz;
        arr = new unsigned*[num];
        for(unsigned i=0; i<num; i++){
            arr[i] = new unsigned[num];
```

```
            for(unsigned j=0; j<num; j++)
                arr[i][j] = 0;
        }
        framing_arr = new unsigned*[num];
        for(unsigned i=0; i<num; i++){
            framing_arr[i] = new unsigned[num];
            for(unsigned j=0; j<num; j++)
                framing_arr[i][j] = 0;
        }
    }

    void framing(){
        if(!(num%2))
            framing_even();
        else if(!(num%3))
            framing3(num);
        else if(!(num%5))
            framing5(num);
        else
            framing_odd();
    }

    Point find_NoClrLeft(){
        for(unsigned i=num/2; i<num; i++){
            for(unsigned j=num/2; j<num; j++)
            {
                if(!arr[i][j])
                    return {i, j};
            }
        }

        return {num, num};
    }

    ~Foursquare(){
        num = 0;
        for(unsigned i=0; i<num; i++){
            delete arr[i];
            delete framing_arr[i];
        }
    }
};


int main(){

    unsigned N;
```

```
    cin >> N;
    Foursquare a(N);
    a.framing();
    return 0;
}
```