

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ
по лабораторной работе №1
по дисциплине «Построение и анализ алгоритмов»
Тема: Поиск с возвратом

Студент гр. 8382

Щеглов А.С.

Преподаватель

Фирсов М.А.

Санкт-Петербург

2020

Цель работы.

Построить итеративный алгоритм поиска с возвратом для решения поставленной задачи. Определить сложность полученного алгоритма по операциям и по памяти.

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

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).

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

Вариант дополнительного задания.

1и. Итеративный бэктрекинг. Поиск решения за разумное время (меньше 1 минуты) для $2 \leq N \leq 30$.

Описание алгоритма

После применения оптимизаций, описанных в следующем пункте, программа работает не с квадратом N на N , а с квадратом $p/2$ без левого верхнего

квадрата, где p – наименьший простой множитель числа N . Программа заполняет по очереди ячейки фигуры (представлен в программе как двумерный массив) начиная от квадратов шириной $n=p/2-1$ и до $n=0$, при заполнении квадратом место последнего квадрата записывается в массив z . После заполнения идет подсчет количества квадратов и запоминание числа разбиения. Далее программа удаляет единицы и последний добавленный квадрат и продолжает работать с предыдущими данными, но на один шаг дальше.

Использованные оптимизации.

Путем математических вычислений было выяснено, что имеет смысл работать не с числом N , а с его наименьшим простым множителем (дает то же число разбиений). Так же выявлено определенное расположение первых трех квадратов, из чего и возникает фигура квадрата без маленького квадратика 1×1 . Так же постоянно идет учет оставшихся клеток в квадрате и каждый раз при полном заполнении квадрата алгоритм смотрит, в теории возможно ли достичь минимума меньше уже имеющегося минимума из этой позиции, если нет – то удаляет предыдущий элемент (через массив z).

Для четных N , таким образом, ответ заранее известен (из за простого множителя 2).

Описание функций и методов.

- `int MiniS(int n, int z)` – n – это ширина квадрата, z – количество незаполненных клеток. Возвращаемое значение – минимальное количество квадратов на z клетках начиная с квадрата шириной n
- `int IsPlaceForP(int i, int j, int n, int p, int flag, int *s)` – i – координата квадрата по оси x , j – координата квадрата по оси y , n – ширина квадрата, p – ширина неоквадрата, $flag$ – флаг, $*s$ – массив (неоквадрат). Возвращает 1 если есть место для установки квадрата с шириной n в точке i, j , при $flag=0$

смотрит есть ли вообще на фигуре место для квадрата шириной n , 0 в другом случае

- `int PlaceP(int i, int j, int n, int p, int *s)` – i - координата квадрата по оси x , j - координата квадрата по оси y , n - ширина квадрата, p -ширина неоквадрата, `flag` – флаг, `*s` – массив(неоквадрат). Помещает квадрат шириной n в точку $i\ j$ (возвращает индекс верхней левой точки квадрата). Возвращает $i*p+j$ (координата левого верхнего угла квадрата).
- `int DelP(int h, int p, int *s)` – h – координаты верхней левой точки квадрата, p - ширина неоквадрата, `s` - массив(неоквадрат). Функция удаляет квадрат, где h – его верхняя левая точка. Возвращает 1.
- `int GetSimple(int N)` – N - число. Возвращает наименьший простой множитель N .

Способ хранения частичных решений.

Частичные решения хранятся в переменной `sz` и двумерном массиве `square`. После каждого полного заполнения квадрата переменная `min` принимает значение `sz` - количества квадратов(если оно меньше уже имеющегося значения `min`) и в таком случае массив `answer` (ответов) приравнивается `square`.

Тестирование с промежуточными выводами

Промежуточными выводами являются *число `sz`* и *таблица неоквадрата `square`*, где

-1 это как раз отсутствующий кусочек квадрата.

`z[cz]` – массив из координат уже поставленных квадратов, `sz` – количество квадратов на столе.

`square` – это двумерный массив представляющий из себя квадрат $p/2+1 * p/2+1$ с элементом `[0][0]='-1'`, там находится текущее разбиение

Эти число и таблица на данный момент алгоритма являются минимальными, что смог вычислить алгоритм

```

25
4
-1 2 2
1 2 2
1 1 1
8
1 1 15
16 1 10
1 16 10
11 16 10
16 11 5
21 11 5
21 16 5
21 21 5

```

```

21
2
-1 1
1 1
6
1 1 14
15 1 7
1 15 7
8 15 7
15 8 7
15 15 7

```

```

19
18
-1 9 9 9 9 9 9 9 9 9 9
1 9 9 9 9 9 9 9 9 9 9
1 9 9 9 9 9 9 9 9 9 9
1 9 9 9 9 9 9 9 9 9 9
1 9 9 9 9 9 9 9 9 9 9
1 9 9 9 9 9 9 9 9 9 9
1 9 9 9 9 9 9 9 9 9 9
1 9 9 9 9 9 9 9 9 9 9
1 9 9 9 9 9 9 9 9 9 9
1 1 1 1 1 1 1 1 1 1 1
11
-1 1 8 8 8 8 8 8 8 8 8
2 2 8 8 8 8 8 8 8 8 8
2 2 8 8 8 8 8 8 8 8 8
2 2 8 8 8 8 8 8 8 8 8
2 2 8 8 8 8 8 8 8 8 8
2 2 8 8 8 8 8 8 8 8 8
2 2 8 8 8 8 8 8 8 8 8
2 2 8 8 8 8 8 8 8 8 8
2 2 2 2 2 2 2 2 2 2 2
1 1 2 2 2 2 2 2 2 2 2
10
-1 1 1 7 7 7 7 7 7 7 7
3 3 3 7 7 7 7 7 7 7 7
3 3 3 7 7 7 7 7 7 7 7
3 3 3 7 7 7 7 7 7 7 7
3 3 3 7 7 7 7 7 7 7 7
3 3 3 7 7 7 7 7 7 7 7
3 3 3 7 7 7 7 7 7 7 7
3 3 3 3 3 3 3 3 3 3 1
3 3 3 3 3 3 3 3 3 3 1
3 3 3 3 3 3 3 3 3 3 1
9
-1 2 2 7 7 7 7 7 7 7 7
1 2 2 7 7 7 7 7 7 7 7
3 3 3 7 7 7 7 7 7 7 7
3 3 3 7 7 7 7 7 7 7 7
3 3 3 7 7 7 7 7 7 7 7
3 3 3 7 7 7 7 7 7 7 7
3 3 3 7 7 7 7 7 7 7 7
3 3 3 1 3 3 3 3 3 3 3
2 2 2 2 3 3 3 3 3 3 3
2 2 2 2 3 3 3 3 3 3 3
13
1 1 10
11 1 9
1 11 9
10 11 2
10 13 7
11 10 1
12 10 3
15 10 3
17 13 1
17 14 3
17 17 3

```

```

11
10
-1 5 5 5 5 5
1 5 5 5 5 5
1 5 5 5 5 5
1 5 5 5 5 5
1 5 5 5 5 5
1 1 1 1 1 1
7
-1 1 4 4 4 4
2 2 4 4 4 4
2 2 4 4 4 4
2 2 4 4 4 4
2 2 2 2 2 2
1 1 2 2 2 2
11
1 1 6
7 1 5
1 7 5
6 7 1
6 8 4
7 6 2
9 6 2
10 8 2
10 10 2
11 6 1
11 7 1

```

Тестирование.

```

29
14
1 1 15
16 1 14
1 16 14
15 16 2
15 18 5
15 23 7
16 15 1
17 15 3
20 15 3
20 18 3
20 21 2
22 21 1
22 22 8
23 15 7

```

Для 29 время работы 45.87

```

17
12
1 1 9
10 1 8
1 10 8
9 10 2
9 12 4
9 16 2
10 9 1
11 9 3
11 16 2
13 12 1
13 13 5
14 9 4

```


Для 17 время работы 1.13

```
19
13
1 1 10
11 1 9
1 11 9
10 11 2
10 13 7
11 10 1
12 10 3
15 10 3
17 13 1
17 14 3
17 17 3
18 10 2
18 12 2
```

Для 19 время работы 2.14

```
21
6
1 1 14
15 1 7
1 15 7
8 15 7
15 8 7
15 15 7
```

Для 21 время работы 0.83

```
23
13
1 1 12
13 1 11
1 13 11
12 13 1
12 14 3
12 17 7
13 12 2
15 12 5
19 17 2
19 19 5
20 12 4
20 16 1
21 16 3
```

Для 23 время работы 4.68

Сложность алгоритма

Сложность алгоритма по операциям можно оценить величиной $O(e^N)$.

Сложность по памяти

В лучшем случае (N – четное) – $O(1)$, т.к. мы уже знаем как метить (см. раздел оптимизация).

В худшем случае (N – нечетное) – $O(N^2)$, т.к. используется двумерный массив.

Выводы.

Была реализована программа, находящая разбиение квадратного поля минимально возможным количеством квадратов. Для решения поставленной задачи был построен и проанализирован итеративный алгоритм поиска с возвратом. Полученный алгоритм обладает экспоненциальной сложностью по операциям и квадратичной сложностью по памяти.

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД.

```
#include <stdio.h>
#include <string>
#include <iostream>

using namespace std;

int MiniS(int n, int z)//Функция для получения минимального возможного числа разбиений из позиции z
{
    int sum=0, i;
    for(i=n;i>0;i--)
    {
        sum+=z/(i*i);
        z=z%(i*i);
    }
    return sum;
}

int IsPlaceForP(int i,int j,int n,int p,int flag, int *s)//Есть ли место для размещения квадрата
{
    int a,b,v=0;
    if(flag==0)
    {
        for(a=0;a<p;a++)
        {
            for(b=0;b<p;b++)
            {
                v+=IsPlaceForP(a,b,n,p,1,s);
            }
        }
    }
    else
    {
        if(i+n>p || j+n>p)
            return 0;
        for(a=i;a<i+n;a++)
        {
            for(b=j;b<j+n;b++)
            {
                if(s[a*p+b]!=0)
                {
                    return 0;
                }
            }
        }
    }
    if(flag==1)
        return 1;
    if(v==0)
        return 0;
    else
        return 1;
}

int PlaceP(int i, int j, int n,int p, int *s)//Разместить квадрат
{

```

```

int a,b;
for(a=i;a<i+n;a++)
{
    for(b=j;b<j+n;b++)
    {
        s[a*p+b]=n;
    }
}
return i*p+j;
}
int DelP(int h,int p, int *s) //Удалить квадрат
{
    int i=h/p,j=h%p, n=s[h],a,b;
    for(a=i;a<i+n;a++)
    {
        for(b=j;b<j+n;b++)
        {
            s[a*p+b]=0;
        }
    }
    return 1;
}
int GetSimple(int N) //Получить наименьшее простое число
{
    int i=2;
    while(1)
    {
        if(N%i==0)
            return i;
        i++;
    }
}

int main()
{
    int N,i,j,a=0,b=0,place=0, maxplace,pz,min=1000;
    cin>>N;
    int p=GetSimple(N);
    int c=p,n=p/2,z[p*p],cz=-1;
    if(p==2) //Обработка четных
    {
        cout<<4<<endl;
        cout<<1<<" "<<1<<" "<<N/2<<endl;
        cout<<N/2+1<<" "<<1<<" "<<N/2<<endl;
        cout<<1<<" "<<N/2+1<<" "<<N/2<<endl;
        cout<<N/2+1<<" "<<N/2+1<<" "<<N/2;
        return 0;
    }
    int s[p][p]={0};
    int square[p/2+1][p/2+1]={0};
    int answer[p/2+1][p/2+1]={0};
    z[0]=0;
    for(i=0;i<p/2+1;i++)//Создание и заполнение неоквадрата
    {
        for(j=0;j<p/2+1;j++)
        {
            square[i][j]=0;
        }
    }
}

```

```

square[0][0]=-1;
if(p%2==0)
{
    PlaceP(0,0,p/2,p,&s[0][0]);
    PlaceP(0,p/2,p/2,p,&s[0][0]);
    PlaceP(p/2,0,p/2,p,&s[0][0]);
}
else
{
    PlaceP(0,0,p/2+1,p,&s[0][0]);
    PlaceP(p/2+1,0,p/2,p,&s[0][0]);
    PlaceP(0,p/2+1,p/2,p,&s[0][0]);
}
p=p/2+1;
maxplace=p*p-1;

i=0;
j=0;
while(square[z[0]/p][z[0]%p]!=1)
{
    while(n>=1)
    {
        if(IsPlaceForP(i,j,n,p,0,&square[0][0]))
        {
            while(i<p)
            {
                while(j<p)
                {
                    if(IsPlaceForP(i,j,n,p,1,&square[0][0]))
                    {
                        cz++;//количество уже поставленных
                        pz=PlaceP(i,j,n,p,&square[0][0]);//ставим
                        place+=n*n;
                        z[cz]=pz;//вносим квадрат в массив
                    }
                    j++;
                }
                j=0;
                i++;
            }
        }
        i=0;
        j=0;
        n--;
    }
    if(cz<min)//Вывод и хранение частичных решений
    {
        min=cz;
        cout<<min<<endl;
        for(i=0;i<p;i++)
        {
            for(j=0;j<p;j++)
            {
                answer[i][j]=square[i][j];
                cout<<answer[i][j]<<" ";
            }
            cout<<endl;
        }
    }
}

```

```

        for(i=0;i<p;i++)
        {
            for(j=0;j<p;j++)
            {
                if(square[i][j]==1)
                {
                    cz--;
                    DelP(i*p+j,p,&square[0][0]);
                    place--;
                }
            }
            n=square[z[cz]/p][z[cz]%p];
            DelP(z[cz],p,&square[0][0]);
            place-=n*n;
            i=(z[cz]+1)/p;
            j=(z[cz]+1)%p;
            cz--;
            while(cz+MiniS(n,maxplace-place)>=min)
        {
            n=square[z[cz]/p][z[cz]%p];
            DelP(z[cz],p,&square[0][0]);
            place-=n*n;
            i=(z[cz]+1)/p;
            j=(z[cz]+1)%p;
            cz--;
        }
        if(cz==-2)
            break;
    }
    p=GetSimple(N); //Преобразование ответа в положенный вид из записи матрицы
    cout<<min+4<<endl;
    cout<<1<<" "<<1<<" "<<(p/2+1)*(N/p)<<endl;
    cout<<(p/2+1)*(N/p)+1<<" "<<1<<" "<<(p/2)*(N/p)<<endl;
    cout<<1<<" "<<(p/2+1)*(N/p)+1<<" "<<(p/2)*(N/p)<<endl;
    answer[0][0]=0;

    j=1;
    for(i=0;i<p/2+1;i++)
    {
        while(j<p/2+1)
        {
            if(answer[i][j]!=0)
            {
                cout<<(p/2)*(N/p)+1+i*(N/p)<<" "<<(p/2)*(N/p)+1+j*(N/p)<<" "<<answer[i][j]*(N/p)<<endl;
                DelP(i*(p/2+1)+j,p/2+1,&answer[0][0]);
            }
            j++;
        }
        j=0;
    }
    return 0;
}

```