

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

Чирков С.А.

Преподаватель

Фирсов М.А.

Санкт-Петербург

2020

Цель работы.

Построить рекурсивный 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.

5р. Рекурсивный бэктрекинг. Возможность задать список квадратов (от 0 до N^2 квадратов в списке), которые обязательно должны быть использованы в покрытии квадрата со стороной N .

Описание алгоритма

Программа работает с квадратом N на N . Программа заполняет по очереди ячейки фигуры (представлен в программе как двумерный массив) начиная от квадратов с длиной стороны $k=\text{ceil}(n/2)$ и до $k=0$ (но идет проверка, если в поданном массиве обязательных квадратов имеется квадрат со стороной больше чем $\text{ceil}(n/2)$ – алгоритм запускается с этого значения), при заполнении квадратом место последнего квадрата записывается в список `res`. После заполнения идет подсчет количества квадратов и запоминание числа разбиения. Дальше программа удаляет квадраты с длиной равной единице и последний добавленный квадрат, продолжает работать с предыдущими данными, но на один шаг дальше.

Использованные оптимизации.

Так как пользователь может ввести обязательные квадраты для замощения было принято решение отказаться от нескольких оптимизаций (установка первых трёх квадратов в углы фиксировано, приведение квадрата к квадрату-аналогу с длиной равной простому числу и прочее). Алгоритм не рассматривает возможное решение, если оно превышает уже найденное по количеству квадратов. Также учитывается сколько минимум квадратов можно поставить на определенную оставшуюся свободную площадь, если это значение с учетом уже поставленных квадратов превышает найденное решение происходит шаг назад. Квадраты с длиной равной единице устанавливаются и удаляются сразу, а не в основном цикле для экономии шагов в цикле. Массив обязательных квадратов тоже анализируется программой, если не был установлен наибольший квадрат из него или была запрошена площадь больше возможной программа останавливается.

Описание функций и структур данных.

- `bt(m, res, n, x, y)` – рекурсивная функция для обхода квадрата, `m` – двумерный список текущего замощения квадрата, `res` – список координат и длин установленных квадратов, `n` – сторона квадрата, который программа пытается установить, `x` и `y` – текущие координаты квадрата. Возвращаемого значения нет.
- `sqnew(m, res, n, x, y)` – функция для установки квадрата. Аргументы соответствуют `bt()`. Возвращаемого значения нет.
- `sqdel(m, res)` – функция удаления последнего поставленного квадрата. Аргументы соответствуют `bt()`. Возвращаемого значения нет.
- `allsqdel(m, res)` – функция удаления всех квадратов с единичной длиной. Аргументы соответствуют `bt()`. Возвращаемого значения нет.
- `printsq(m)` – функция вывода текущего замощения квадрата, `m` – двумерный список текущего замощения квадрата. Возвращаемого значения нет.
- `check(m,x,y,n)` – функция проверки возможно ли поставить квадрат. Аргументы соответствуют `bt()`. Возвращает 1 если установить возможно, в противном случае 0.
- `IsPlace(n)` – функция получения наименьшего количества квадратов, возможных для оставшейся площади, `n` – сторона квадрата, который программа пытается установить. Возвращает наименьшее количество оставшихся квадратов для замощения.
- `GetSimple(p)` – функция, возвращающая первый простой делитель числа `p`.

Способ хранения частичных решений.

Частичные решения хранятся в переменной `res` в виде списка координат и длин соответствующих квадратов для замощения. После каждого полного заполнения квадрата переменная `res_min` принимает значение `len(res)` - количества квадратов (если оно меньше уже имеющегося значения `res_min`) и в таком случае массив `t` (ответов) приравнивается `res`.

Тестирование с промежуточными выводами

```
введите длину квадрата
16
введите длины квадрата, которые обязательны для ответа (через пробел), 0 - если неважно
11
16 квадратов - вероятный ответ: ['1 1 11', '1 12 5', '6 12 5', '11 12 5', '12 1 5', '12 6 5',
'12 11 1', '13 11 1', '14 11 1', '15 11 1', '16 11 1', '16 12 1', '16 13 1', '16 14 1', '16 15
1', '16 16 1']
15 квадратов - вероятный ответ: ['1 1 11', '1 12 5', '6 12 5', '11 12 5', '12 1 5', '12 6 3',
'12 9 3', '15 6 2', '15 8 2', '15 10 2', '16 12 1', '16 13 1', '16 14 1', '16 15 1', '16 16 1'
]
13 квадратов - вероятный ответ: ['1 1 11', '1 12 5', '6 12 5', '12 1 5', '12 6 5', '11 13 4',
'12 11 2', '14 11 2', '15 13 2', '15 15 2', '11 12 1', '16 11 1', '16 12 1']
13 квадратов - минимальное замощение
1 1 11
1 12 5
6 12 5
12 1 5
12 6 5
11 13 4
12 11 2
14 11 2
15 13 2
15 15 2
11 12 1
16 11 1
16 12 1
```

Тестирование.

8	8
5 2	3
10 квадратов - минимальное замощение	11 квадратов - минимальное замощение
1 1 5	1 1 4
1 6 3	1 5 4
4 6 3	5 1 4
6 1 3	5 5 3
6 4 2	5 8 1
7 6 2	6 8 1
7 8 1	7 8 1
8 4 1	8 5 1
8 5 1	8 6 1
8 8 1	8 7 1
	8 8 1

```

16
11 3
13 квадратов - минимальное замощение
1 1 11
1 12 5
6 12 5
12 1 5
12 6 5
11 12 3
14 11 3
14 14 3
4
0
4 квадратов - минимальное замощение
1 1 2
1 3 2
3 1 2
3 3 2
11 15 2
12 11 1
13 11 1
13 15 1
13 16 1

16
5
6
Нет решения
16
1 квадратов - минимальное замощение
1 1 16
8
6 5
Нет решения

```

Сложность алгоритма

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

Сложность по памяти

$O(N^2)$, 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 рекурсивный алгоритм поиска с возвратом. Полученный алгоритм обладает экспоненциальной сложностью по операциям и квадратичной сложностью по памяти.

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД.

```
def bt(m, res, n, x, y):
    """
    рекурсивная функция обхода квадрата
    аргументы:
        двумерный массив m (текущее состояние замощения)
        список res (текущие координаты и длины квадратов в замощении)
        число n (сторона обрабатываемой длины квадрата)
        числа x,y (текущие координаты для обработки в цикле далее)
    """
    global t
    global res_min
    global free
    global use
    if use[0]>(-len(m) // 2): #пропуск лишних вариантов, если есть обязательный квадрат больше ceil(n//2)
        if res:
            a = res[0].split(' ')
            if int(a[0])!=1:
                return
            if int(a[1])!=1:
                return
            if int(a[2])<use[0]:
                return
        if len(res)>1 and len(use)>1:
            a = res[1].split(' ')
            if int(a[2])<use[1]:
                return
        if res_min <= len(res): #пропуск лишних вариантов, относительно промежуточного найденного решения
            return
        if lsPlace(n)+len(res)>=res_min: #пропуск лишних вариантов, относительно возможной оставшейся
свободной площади
            return
        if n==1: #единичные квадраты выставляются не в основном цикле, чтобы не было дополнительных
итераций
            for i in range(len(m)):
                for j in range(len(m)):
                    if(m[i][j]==0):
                        sqnew(m, i, j, n, res)
                        free-=1
        else:
            if free-n*n>=0: #если есть свободное место
                for i in range(x,x+1): #проход текущей строки до её конца
                    for j in range(y,len(m)-n+1):
                        if check(m, i, j, n):
                            sqnew(m, i, j, n, res)
                            free-=n*n
                            if j==len(m)-(n-1):
                                bt(m, res, n, i+1,0)
                            else:
                                bt(m, res, n, i, j+n-1)
                        sqdel(m, res) #шаг назад для прохода всех вариантов
                        free+=n*n

                for i in range(x+1,len(m)-n+1): #проход остальных строк с их начала
                    for j in range(len(m)-n+1):
                        if check(m, i, j, n):
                            sqnew(m, i, j, n, res)
                            free-=n*n
```



```

        if j==len(m)-(n-1):
            bt(m, res, n, i+1,0)
        else:
            bt(m, res, n, i, j+n-1)
            sqdel(m, res)
            free+=n*n
    bt(m, res, n-1, 0, 0)
if free==0:    #проверка на возможный ответ
    if res_min > len(res):
        for s in use: #проверка на обязательные квадраты
            flag=0
            for item in res:
                a= item.split(' ')
                if str(s)==a[2]:
                    flag=1
                    break
            if flag==0:
                break
        if flag==1: #сохранение ответа
            res_min = len(res)
            t=list(res)
            #print(res_min,'квадратов - вероятный ответ:',t)

    allsqdel(m, res)

def sqnew(m, x, y, n, res): #установка квадрата
    for i in range(n):
        for j in range(n):
            m[x+i][y+j]=1
    res.append(str(x+1)+' '+str(y+1)+' '+str(n))

def sqdel(m, res): #удаление квадрата
    if res:
        x = res[len(res)-1].split(' ')
        for i in range(int(x[2])):
            for j in range(int(x[2])):
                m[int(x[0])+i-1][int(x[1])+j-1]=0
    res.pop()

def allsqdel(m, res): #удаление всех единичных квадратов
    global free
    while len(res)>0:
        x = res[len(res)-1].split(' ')
        if int(x[2])==1:
            sqdel(m,res)
            free+=1
        else:
            break

def printsq(m): #промежуточный вывод двумерного массива
    for i in range(len(m)):
        for j in range(len(m)):
            print(m[i][j],end=' ')
        print("\n")
    print('-----')

def check(m,x,y,n): #проверка можно ли поставить квадрат
    for i in range(n):
        for j in range(n):

```

```

        if m[x+i][y+j]!=0:
            return 0
    return 1

def IsPlace(n): #получение наименьшего количества квадратов, возможных для оставшейся площади
    global free
    answer = 0
    copyfree = free
    while n > 0:
        answer+=copyfree//(n*n)
        copyfree=copyfree%(n*n)
        n=n-1
    return answer

def GetSimple(p): #приведение к простому числу
    for i in range(2, p):
        if p%i==0:
            return i
    return p

#print('введите длину квадрата')
c = int(input())
#print("введите длины квадрата, которые обязательны для ответа (через пробел)")
use = list(map(int,input().split(' ')))
if use == [0]:
    use = [-(-c // 2)]
free=c*c
b = [0] * c
for i in range(c):
    b[i] = [0] * c
n = -(-c // 2)
res = []
t = []
res_min = c*c
use.sort(reverse = True)
summ=0
for s in use:
    summ+=s*s
if summ > c*c:
    print("Нет решения")
else:
    if use[0]<=n:
        bt(b, res, n, 0, 0)
    else:
        bt(b, res, use[0], 0, 0)
    if res_min!=c*c and t!=[]:
        print(res_min,"квадратов - минимальное замощение")
        #print("координаты и длина квадратов:")
        for x in t:
            q=x.split(' ')
            q[0]=(int(q[0])-1)+1
            q[1]=(int(q[1])-1)+1
            q[2]=int(q[2])
            print(q[0], q[1], q[2])
    else:
        print("Нет решения")

```