

# Algorithm-homework2

孔静—2014K8009929022

October 30, 2016

## Contents

1 Largest Divisible Subset	1
2 Money Robbing	3
3 Partition	4
4 Decoding	5
5 Minimum path sum	6

## 1 Largest Divisible Subset

a

考虑从小到大排列的数，不妨设  $a < b < c$ ，若  $b$  能整除  $c$ ， $a$  能整除  $b$ ，那么由于传递性， $a$  也能整除  $c$ 。在整除关系传递性的基础上，考虑最长整除序列，就可以类比算法研讨课上最长递增序列一样。所以我们第一步排序；第二步考虑  $S_i$ ，在前面已经有了  $i-1$  个数，对每一个满足  $j < i$  的  $Opt(j)$ ，若  $s_i \% s_j = 0$  满足，则可以将  $s_i$  增加在  $s_j$  所在整除序列上，因此：

$$Opt(i) = \max\{f(1), f(2), f(3), \dots, f(i-1)\}$$

其中

$$f(i) = \begin{cases} 1, & 0 \leq j < i \quad s_i \% s_j \neq 0 \\ Opt(j) + 1, & 0 \leq j < i \quad s_i \% s_j = 0 \end{cases}$$

b

即利用上述公式对数据先排序后遍历，由于需要输出该序列，我设了两个参量， $Opt[i][1]$  来记录当前序列长度。 $Opt[i][2]$  来记录他的当前最大序列的上一个数。

---

**Algorithm 1** Find The Largest Divisible Subset

---

```
procedure SEARCH(n)
  quicksort(S) * 快速排序 *
  i = 1
  for i <= n do
    j = 1
    Opt[i][1] = 1
    for j < i do
      if S[i] % S[j] == 0 then
        if Opt[i][0] < Opt[j][0] + 1 then
          Opt[i][1] = Opt[j][0] + 1
          Opt[i][2] = S[j]
        end if
      end if
    end for
    m = max(Opt) * 找到令 Opt[i][1] 最大的 i *
    j = j + 1
  end for
  i = i + 1
end for
printsubset(m) * 利用 [2] 存的数据及传递性输出该最长序列 *
end procedure
```

---

c

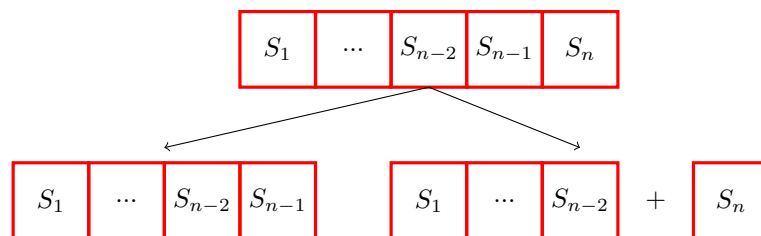
显然正确，首先已经从小到大排序好，对于  $i > j$ ，若  $s_i$  能被  $s_j$  整除，那么也必然能被其他  $s_j$  序列中的数整除，因为其他数比  $s_j$  更小，利用整除的传递性，能寻找出最长序列。

d

快排  $O(n \log n)$ ，遍历  $O(n^2)$ ，找到最大的数  $O(n)$ 。  
综上算法复杂度是  $O(n^2)$ 。

## 2 Money Robbing

a 不妨将房子编号，并将金额记为  $S[i]$ 。



$$\text{Opt}(i) = \max \{ \text{Opt}(i-1), \text{Opt}(i-2) + S_i \}$$

b

---

### Algorithm 2 Money Robbing

---

```

procedure MOREMONEY(n)
    Opt[1] = S[1]
    Opt[2] = max(S[1], S[2])
    i = 3
    for i <= n do
        Opt[i] = max(Opt[i-1], Opt[i-2] + S[i])
        i = i + 1
    end for
    return <Opt[n]>
end procedure

```

---

c

对于每个  $S[i]$  来说，只有被抢和没被抢两种情况，没被抢那么  $\text{Opt}[i] = \text{Opt}[i-1]$ ， $S[i]$  被抢，那么  $S[i-1]$  必然没被抢，所以  $\text{Opt}[i] = \text{Opt}[i-2] + S[i]$ 。因此是这两种情况的最大值。

d

该程序只是进行一次大小比较，遍历即可，子问题  $O(1)$ ，遍历  $O(n)$ 。  
 综上算法复杂度是  $O(n)$ 。

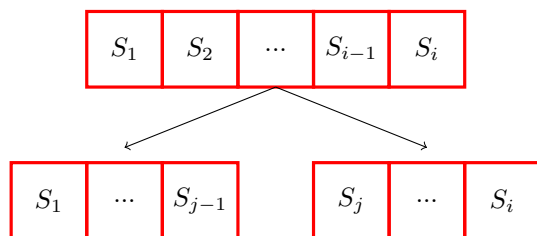
circle

删去任意一点，剩下部分仍可看作一条直线，区别在于  $n$  个点删去该点后，就只需要返回  $\text{Opt}[n-1]$  即可。但不能确认哪个点可以被删去。不过任意 2 个连续的点必然有 1 个点不被选中可删去，所以可以在直线基础上，以任意 2 个连续的 2 个点为起点  $S[1]$ ，返回  $\text{Opt}[n-1]$ ，比较 2 个返回值，选择最大的即可。计算了 2 次直线情况，所以算法复杂度仍为  $O(n)$ 。

### 3 Partition

a

对于末尾字符  $S[i]$ ，它与它前面字符可能形成多种情况的回文序列，形成回文后，即可切断扔掉，然后计算剩下的字符仍需要切几刀即可。



$$\text{Opt}(i) = \max_{j \in P} \{ \text{Opt}(j-1) + 1 \}$$

其中， $j$  满足条件：第  $j$  个字符到第  $i$  个字符为回文子序列

b

---

#### Algorithm 3 Partition

---

```

procedure PARTITION( $n$ )
     $\text{Opt}[0] = -1$ 
     $i = 2$ 
    for  $i \leq n$  do
         $j = 1$ 
        for  $j \leq i$  do
            if  $\text{judge}(j, i) == 1$  then * 判断 S 序列第  $j$  到  $i$  个字符是否是回文 *
                 $\text{Opt}[i] = \min(\text{Opt}[i], \text{Opt}[j-1] + 1)$ 
            end if
        end for
    end for
    return  $\langle \text{Opt}[n] \rangle$ 
end procedure

```

---

c

考虑字符  $S[i]$ ，判断其与之前字符是否能成为回文，有至多  $i$  种情况，对每种情况进行遍历，选最小值，即是这  $i$  个字符的最小切分，遍历到  $n$ ，即可得到最终结果。

d

子问题有  $O(n)$ ，判断回文  $O(n)$ ，遍历是  $O(n)$ 。  
 综上算法复杂度是  $O(n^3)$ 。

## 4 Decoding

a

对于末尾数字  $S[i]$ ，它与  $S[i-1]$  如果可以组成一个编码是一种情况；它单独成一个编码也是一种情况。

$$Opt(i) = \begin{cases} Opt(i-1) + Opt(i-2), S_{i-1}S_i \notin P \\ Opt(i-1), S_{i-1}S_i \in P \end{cases}$$
$$P = \{1, 2, 3 \dots 26\}$$

b

---

**Algorithm 4** Decoding

---

```
procedure PARTITION(n)
  Opt[0] = 0
  Opt[1] = 1
  i = 2
  for i <= n do
    if 27 > S[i-1] * 10 + S[i] > 0 then
      Opt[i] = Opt[i-1] + Opt[i-2]
    else
      Opt[i] = Opt[i-1]
    end if
    i = i + 1
  end for
  return <Opt[n]>
end procedure
```

---

c

类比斐波那契数列，但与之不同的是， $S_{i-1}S_i$  不一定有对应的字母，所以有两种情况，遍历一遍即可得到结果。

d

子问题有  $O(2)$ ，遍历是  $O(n)$ 。  
综上算法复杂度是  $O(n)$ 。

## 5 Minimum path sum

a

在第  $i$  层,  $i$  个数据, 每个数据向上至多 2 条路, 即对每个数据来说有 1 2 种情况可以走。

$$\text{Opt}[i][j] = \begin{cases} \text{num}[i][j] + \min(\text{Opt}[i-1][j-1], \text{Opt}[i-1][j]), & 1 < j < i \\ \text{num}[i][j] + \text{Opt}[i-1][1], & j = 0 \\ \text{num}[i][j] + \text{Opt}[i-1][i-1], & j = i \end{cases}$$

b

---

**Algorithm 5** Minimum path sum

---

```
procedure PARTITION(S,n)
    Opt[1] = 1
    i = 2
    for i <= n do
        Opt[i][1] = num[i][1] + Opt[i-1][1]
        Opt[i][i] = num[i][i] + Opt[i-1][i-1]
        j = 2
        for j <= i do
            Opt[i][j] = num[i][j] + min(Opt[i-1][j], Opt[i-1][j+1])
            j = j + 1
        end for
        i = i + 1
    end for
    answer = minOpt , n * 对第 n 层 n 个结果找出最小值 return <answer>
end procedure
```

---

c

除了首末两个数据只有一条路可走, 其他数据具有选择权利, 然后对最后结果找出最小, 即最终到达该层的最小值。

d

子问题有  $O(2)$ , 遍历是  $O(n^2)$ 。  
综上算法复杂度是  $O(n^2)$ 。