# Week 6

软73 沈冠霖 2017013569

April 2, 2019

## 1 T1

**代码**

```
OPTIMAL_BST(A,p,r)
for i=1 to n+1
    e[i, i-1] = q_{i-1}
    w[i, i-1] = q_{i-1}
for l = 1 to n
    for i = 1 to n-l+1
        j=i+l-1
        e[i,j]=∞
        w[i,j]=w[i,j-1]+p_j + q_j
         for r = root[i,j-1] to root[i+1,j]
            t = e[i,r-1]+e[r+1,j]+w[i,j]
            if t¡e[i,j]
                e[i,j]=t
                root[i,j]=r
 return e and root
```

只是把第三层循环改成了从root[i,j-1]到root[i+1,j]

**证明** 首先，正确性是显然的，根据题目中的结论，$root[i, j-1] \leq root[i, j] \leq root[i+1, j]$，而在求解e[i,j]之前，已经求解过所有长度为l-1以及更短的子树的最优情况和其根了，因此必然正确

其次，对于时间复杂度：考虑长度为l的情况，共有e[1,l]到e[n-l+1,n]这n-l+1个情况需要求解，求解情况e[i,i+l-1]需要遍历root[i+1,i+l-1] - root[i,i+l-2] + 1次。对所有n-l+1种情况的求解遍历次数求和,可以得到，处理所有长度为l的情况，需要遍历l + root[l+1,n]-root[1,l-1]次，因为$\forall 1 \leq i \leq j \leq n, 1 \leq root[i, j] \leq n$,因此处理长度为l的情况，需要处理时间为O(n)，共有n种长度，因此时间复杂度为O($n^2$)。又因为这样做有$\theta(n^2)$个子问题，因此时间复杂度也是$\Omega(n^2)$，因此时间复杂度是$\theta(n^2)$

# 2 T3

## 2.1 3.1

证明：假设从第一行到第i-1行（$1 < i \leq n$）第j列的缝隙至少有$x_{i-1}$种取值，那么以第i行任意一个元素为结尾的缝隙可能来自其左上方，正上方，右上方，即使元素在最左或者最右，也能有$2x_{i-1}$种取值。因为$x_1 = 1$,则有$2^{m-1} \leq x_m$，从第一行到第m行的所求缝隙至少有$n2^{m-1}$条，因此与m成指数关系

## 2.2 3.2

**设计思路** 子问题：假设这张图片只有前i行，以像素(i,j)为结尾的缝隙的最小破坏度V[i,j]
状态转移方程：V[1,j] = d[1,j],V[i,j]=min(V[i-1,j],V[i-1,j-1],V[i-1,j+1])+d[i,j]（所有不满足$1 \leq i \leq m, 1 \leq j \leq n$的V[i,j]都定义为正无穷）
结果：min(V[m][j])就是最小的破坏度，可以定义一个数组previous[i][j]存储以(i,j)为最下方点的一条最优子缝隙中,(i,j)点的上一个坐标，然后回溯就能得到具体的缝隙

**证明** 首先，每个子问题之间互相独立。因为要以(i,j)为最下方点的最优子缝隙，只需要找到以(i,j-1),(i,j),(i,j+1)为最下方点的子缝隙的最优解就行了，和具体前面缝隙如何无关。
其次，子问题最优解是整体最优解。假设存储的V[i,j]不是所求的最优解，那么必定可以找到另一条到(i,j)的缝隙，其破坏值更小，那么从这条缝隙按照相同的道路向下延伸，得到的最终结果必定更小，与原结果是整体最优解矛盾。
最终，算法需要更新mn个子问题，每个子问题最多需要3次计算，因此时间复杂度是O(mn)

**代码**
```
OPTIMAL_Seam(d,V,Seam)
for i=1 to n
    V[1,i]=d[1,i]
    previous[1,i]=0
for i = 2 to m
    for j = 1 to n
        a = V[i-1,j-1],b=V[i-1,j],c=V[i-1,j+1]
        if(j==1)
            a = +∞
        if(j==n)
            c = +∞
        if(a < b&&a < c)
            V[i,j]=a+d[i,j],previous[i,j]=j-1
        else if(c < b&&c < a)
            V[i,j]=c+d[i,j],previous[i,j]=j+1
        else
            V[i,j]=b+d[i,j],previous[i,j]=j
```

```
    min = ∞,minplace = 0
    for i = 1 to n
        if(V[m,i]¡min)
            min = V[m,i],minplace=i
    currentplace=minplace
    for i = m to 1
        Seam[i]=currentplace
        currentplace = previous[i,currentplace]
    return min and Seam
```

此时min就是最小的损耗值，Seam[i]代表第i行位于这个缝隙里的是(i,Seam[i])像素

# 3 T2

## 3.1 设计思路

**1.划分子问题** 设原数组为A,min[i]表示长度为i的单调递增子序列长度最小末尾元素

**2.初始值** 设置min[i]全部为正无穷，当前已经更新0个元素

**3.状态转移方程** 设当前更新第i个元素，$\min[j] \leq A[i] < min[j{+}1]$,则min[j+1] = A[i],更新到$i > n$结束，结果就是最大的使得min[j]不为正无穷的j
想要求出最长的一个序列，可以先用一个数组sequence[i]记录min[i]对应的最小末尾的下标。在更新每个元素的时候用一个数组previous[i]记录A[i]的前一个数下标，也就是previous[i] = sequence[j]，这样递推就可以求出序列

**4.正确性证明** 首先，min数组单调递增。因为假设已经更新了m个元素，如果有$min[i] > min[j], i < j$,则在min[j]对应的单调递增子序列里截取前i个数，他们一定是单调递增的,且第i个数$p \leq min[j] < min[i]$，与$min[i] \leq p$矛盾。
其次，如果更新的第i个新元素有$\min[j] \leq A[i] < min[j+1]$，则将其放到此时min[j]对应的那个单调递增子序列后面，就可以产生一个最小末尾元素小于此时min[j+1]的长为j+1的单调递增子序列，而对于$k > j+1, min[k-1] > A[i]$,不可能找到一个在前i-1个元素的序列中长度为k-1的单调递增子序列，使得可以添加A[i]让其仍然满足条件，所以此时，min为更新i个元素后的局部最优解。因此，根据数学归纳法，所有元素全部更新完后，min为整体最优解

**5.复杂度分析** 每次更新只需要一个时间复杂度O(n)的二分查找，一共更新n次，所以总共时间复杂度是O(nlgn)

### 3.2 测试结果

**测试环境** CPU:Inter Core i5-6300HQ,2.3GHZ
内存：12G
环境：VS2017,release模式

**正确性分析** **先**测试3组特殊数据（完全正序，完全倒序，完全相等，再测试5组长度为5的随机数据，详细结果见表1。可以看出，测试的结果完全正确，可以初步说明结果基本正确

**时间分析** 算法的理论复杂度是O(nlgn)，而实际我测试了从$10^1$到$10^8$的8组数据，得到运行时间如表2。
可以看出，在数据规模很小($\leq 10000$)的时候，运行时间是不定波动的，因为我的代码在测试时间的时候还有I/O，这会影响时间，但是，可以看出，这个时候运算非常快。
在数据规模大大的时候($\geq 1000000$)，可以看出运行时间明显和数据规模正相关，而且数据规模每扩大10倍，时间增加10倍多一些，大致可以看出nlgn的关系

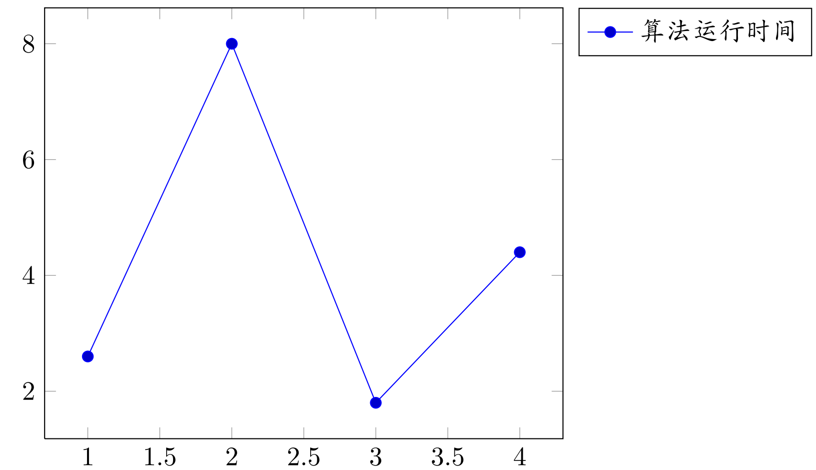
Table 1: 几组数据的运行结果和预期结果对比

| 测量序号 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 数据 | 1 2 3 4 5 | 5 4 3 2 1 | 5 5 5 5 5 | 41 67 3 0 69 | 2 7 5 6 6 | 5 45 81 7 61 | 91 95 42 27 36 | 9 4 2 5 9 |
| 预期结果 | 5 | 1 | 5 | 3 | 4 | 3 | 2 | 3 |
| 结果 | 5 | 1 | 5 | 3 | 4 | 3 | 2 | 3 |
| 是否正确 | 是 | 是 | 是 | 是 | 是 | 是 | 是 | 是 |

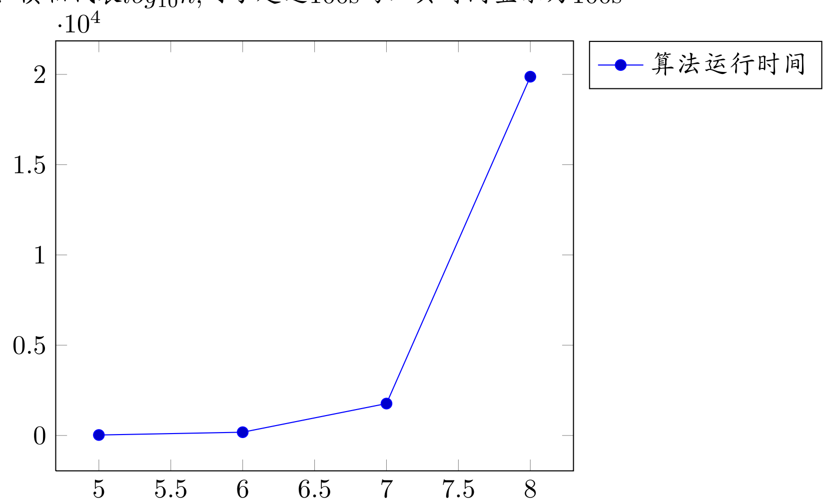
Table 2: 不同数据规模下运行的时间

| 测量序号 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 数据范围 | 10 | 100 | 1000 | 10000 | $10^5$ | $10^6$ | $10^7$ | $10^8$ |
| 算法运行时间 (ms) | 2.6 | 8 | 1.8 | 4.4 | 25.4 | 179.8 | 1766.6 | 19872.2 |

注：每组数据都是运行5次后取的平均值




注：横轴代表$log_{10}n$,对于超过100s的，其时间显示为100s




注：横轴代表$log_{10}n$,对于超过100s的，其时间显示为100s