第六章　动态规划

6.3

分析：

可以不记录$f_i[j]$，而用两个变量 f1 和 f2 来保存源程序中的$f_1[j]$和$f_2[j]$，并在一轮循环中用 f1 f2 来计算$f_1[j]$和$f_2[j]$即新的 f1,f2。因而可以节省 2n-2 个空间，在最后一轮循环后即 j=n 后，使之仍然能够计算出 f*，并且仍然能够构造出最快装配路线。F1,f2 保存的为$f_1[n]$和$f_2[n]$。根据这两个值，就能计算出 f*。

6.5

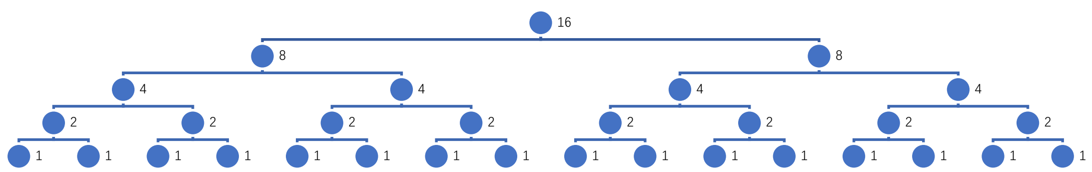
类似于归并算法

分治法,其中 s 数组存放记录的是划分位置 k

MatrixChainMultiply($A_{i\ldots j}$,s,i,j)

```
if i =j then
        return   A_i
else
        q←s[i,j]
        A1←MatrixChainMultiply(A_{i...q},s,i,q)
        A2←MatrixChainMultiply(A_{q+1...j},s,q+1,j)
        return MatrixMultiply(A1,A2)
```

6.7



备忘录方法是一种自顶向下的高效动态规划方法，它可能包含递归过程，并在第一次解决一个子问题时将结果记录到一个表中，在下一次遇见孩子子问题时，只需查表而无需再解决一次，因而当某个算法有重叠子问题时，能很高效地解决问题。而 MergeSort 算法的每个子问题都是相互独立的，不存在重叠子问题，因而使用备忘录方法并不能提高效率。

6.9

个人理解这道题的原算法本质就是通过观测 b 数组中记录的路径来找到 LCS 的来源。而 B 数组的得到方式是通过 X[]和 Y[]的比较而来，当不存在 b 数组时，只需要重新比较 X[]和 Y[]更换成条件即可。

```
PrintLCS(c,X,Y,i,j)
if i=0 or j=0 then
    return 0
if X[i]=Y[j] then
    PrintLCS(c,X,Y,i-1,j-1)
    print x[i]
else jf c[i-1,j]>=c[I,j-1] then
        PrintLCS(c,X,Y,i-1,j)
    else
        PrintLCS(c,X,Y,i,j-1)
```

6.12

动态规划法（时间复杂度 O(N^2))

设长度为 N 的数组为{a0，a1, a2, ...an-1)，则假定以 aj 结尾的数组序列的最长递增子序列长度为 L(j)，则 L(j)={ max(L(i))+1, i<j 且 a[i]<a[j] }。也就是说，我们需要遍历在 j 之前的所有位置 i(从 0 到 j-1)，找出满足条件 a[i]<a[j]的 L(i)，求出 max(L(i))+1 即为 L(j)的值。最后，我们遍历所有的 L(j)（从 0 到 N-1），找出最大值即为最大递增子序列。时间复杂度为 O(N^2)。例如给定的数组为{5，6，7，1，2，8}，则 L(0)=1, L(1)=2, L(2)=3, L(3)=1, L(4)=2, L(5)=4。所以该数组最长递增子序列长度为 4，序列为{5，6，7，8}。算法代码如下：

```
LIS(A,L)
for i⟵0 to n do
    L[i]⟵1
for j⟵0 to n do
    for i⟵0 to j do
        if(A[j]>A[i]Z and L[j] < L[i]+1) then
            L[j] =L[i]+1
for i⟵0 to n do
    max(L[i])
return max
```

解法 3：O(NlgN）算法

假设存在一个序列 d[1..9] ={ 2，1 ，5 ，3 ，6，4， 8 ，9， 7}，可以看出来它的 LIS 长度为 5。

下面一步一步试着找出它。

我们定义一个序列 B，然后令 i = 1 to 9 逐个考察这个序列。

此外，我们用一个变量 Len 来记录现在最长算到多少了

首先，把 d[1]有序地放到 B 里，令 B[1] = 2，就是说当只有 1 一个数字 2 的时候，长度为 1 的 LIS 的最小末尾是 2。这时 Len=1

然后，把 d[2]有序地放到 B 里，令 B[1] = 1，就是说长度为 1 的 LIS 的最小末尾是 1，d[1]=2 已经没用了，很容易理解吧。这时 Len=1

接着，d[3] = 5，d[3]>B[1]，所以令 B[1+1]=B[2]=d[3]=5，就是说长度为 2 的 LIS 的最小末尾是 5，很容易理解吧。这时候 B[1..2] = 1, 5，Len = 2

再来，d[4] = 3，它正好加在 1,5 之间，放在 1 的位置显然不合适，因为 1 小于 3，长度为 1 的 LIS 最小末尾应该是 1，这样很容易推知，长度为 2 的 LIS 最小末尾是 3，于是可以把 5 淘汰掉，这时候 B[1..2] = 1, 3，Len = 2

继续，d[5] = 6，它在 3 后面，因为 B[2] = 3，而 6 在 3 后面，于是很容易可以推知 B[3] = 6，这时 B[1..3] = 1, 3, 6，还是很容易理解吧？ Len = 3 了噢。

第 6 个, d[6] = 4，你看它在 3 和 6 之间，于是我们就可以把 6 替换掉，得到 B[3] = 4。B[1..3] = 1, 3, 4， Len 继续等于 3

第 7 个, d[7] = 8，它很大，比 4 大，嗯。于是 B[4] = 8。Len 变成 4 了

第 8 个, d[8] = 9，得到 B[5] = 9，嗯。Len 继续增大，到 5 了。

最后一个, d[9] = 7，它在 B[3] = 4 和 B[4] = 8 之间，所以我们知道，最新的 B[4] =7，B[1..5] = 1, 3, 4, 7, 9，Len = 5。

于是我们知道了 LIS 的长度为 5。

注意，这个 1,3,4,7,9 不是 LIS，它只是存储的对应长度 LIS 的最小末尾。有了这个末尾，我们就可以一个一个地插入数据。虽然最后一个 d[9] = 7 更新进去对于这组数据没有什么意义，但是如果后面再出现两个数字 8 和 9，那么就可以把 8 更新到 d[5], 9 更新到 d[6]，得出 LIS 的长度为 6。

然后应该发现一件事情了：在 B 中插入数据是有序的，而且是进行替换而不需要挪动——也就是说，我们可以使用二分查找，将每一个数字的插入时间优化到 O(logN)~~~~~于是算法的时间复杂度就降低到了 O(NlogN)～！

代码如下（代码中的数组 B 从位置 0 开始存数据）：

数组 B 记录 LIS 序列

```
LIS2(A)
len=1
B[0]=A[0]
for i←0 to n do
    if(A[i] >B[len-1])
        B[len] = A[i]
        len++
    else
        pos = BiSearch(B,len,A[i]) //二分法查找插入的位置
        B[pos]=A[i]
return len
```

智能科学与技术系
周雨
31520181154417