3-1：动态规划

2018 年 9 月 10 日

请独立完成作业，不得抄袭。
若参考了其它资料，请给出引用。
鼓励讨论，但需独立书写解题过程。

第一部分　作业

算法排版又全乱掉了！

题目 (UD: 15.1.1)
由公式(15.3)和初始条件$T(0) = 1$，证明公式(15.4)成立。

证明：
$T(n) = 1 + \sum_{j=0}^{n-1} T(j)$
$T(n-1) = 1 + \sum_{j=0}^{n-2} T(j), n \geq 2$
相减得:$T(n) - T(n-1) = T(n-1)$
$T(n) = 2T(n-1), n \geq 2$
可知$T(n)$构成公比为2的等比数列
$\because T(0) = 1,$
$\therefore T(n) = 2^n$ □

题目 (UD: 15.1.3)
我们对钢条切割问题进行一点修改，除了切割下的钢条段具有不同价格p_i外，每次切割还要付出固定的成本c。这样，切割方案的收益就等于钢条段的价格之和减去切割的成本。设计一个动态规划算法解决修改后的钢条切割问题。

解答：
相比于原来的问题，修改后需要将每种切割方式得到的$r[i]$减去c，不切割的方式除外，所以公式(15.1)可以改写为：

$$r_n = \max(p_n, \max_{1 \leq i \leq n-1}(p_i + r_{n-i}) - c)$$

Algorithm 1 EXTENDED-BOTTOM-UP-CUT-ROD2.0(p, n, c)

```
let r[0..n] and s[0..n]be new arrays
r[0]=0
for j = 1 to n do
  q = −∞
  for i = 1 to j − 1 do
    if q < p[i] + r[j − i] then
      q = p[i] + r[j − i]
      s[j] = i
    end if
    q = q − c
    if q < p[j] then
      q = p[j]
      s[j] = n
    end if
    r[j] = q
  end for
end for
return r and s
```

题目 (UD: 15.2.2)

设计递归算法MATRIX-CHAIN-MULTIPLY(A, s, i, j)，实现矩阵链最优代价乘法计算的真正计算过程，其输入参数为矩阵序列$(A_1, A_2, ..., A_n)$，MATRIX-CAHIN-ORDER得到的表s，以及下标i和j。（初始调用应为MATRIX-CHAIN-MULTIPLY$(A, s, 1, n)$。）

解答：

直接使用课本中给出的矩阵乘法计算算法MATRIX-MULTIPLY(A, B). MATRIX-CHAIN-MULTIPLY(A, s, i, j)的返回值为一个矩阵。

Algorithm 2 MATRIX-CHAIN-MULTIPLY(A, s, i, j)

```
if i == j then
  return A_i
end if
k = s[i, j]
B =MATRIX-CHAIN-MULTIPLY(A, s, i, k)
C =MATRIX-CHAIN-MULTIPLY(A, s, K + 1, j)
return MATRIX-MULTIPLY(B, C)
```

题目 (UD: 15.2.4)
对输入链长度为n的矩阵链恒发问题，描述其子问题图：它包含多少个顶点？包含多少条边？这些边分别连接那些顶点？

解答：
包含从$m[1,1]$到$m[n,n](i <= j)$共$n+(n-1)+...+1=\frac{n(n+1)}{2}$个顶点（子问题）；
从每个顶点$m[i,j](i < j)$出发都有$2(j-i)$条边，$i = j$的顶点出发没有边，因此总共有$\sum_{j=1}^{n}\sum_{i=1}^{j} = \frac{n(n+1)(n-1)}{3}$条边；（计算过程已省略）
从顶点$m[i,j](i<j)$出发的边连接了所有$m[i,k], m[k+1,j](i \leq k < j)$顶点

题目 (UD: 15.3.3)
考虑矩阵链乘法问题的一个变形：目标改为最大化矩阵序列括号化方案的标量乘法运算次数，而非最小化，此问题具有最优子结构性质吗？

解答：
此变形问题仍具有最优子结构性质。
【以下证明过程基本照抄书上原话】
假设$A_iA_{i+1}...A_j$的最优括号化方案的分割点在A_k和A_{k+1}之间。那么，继续对“前缀”子链$A_iA_{i+1}...Ak$进行括号化时，我们应该直接采用独立求解它时所得的最优方案。如果不采用独立求解$A_iA_{i+1}...Ak$所得的最优方案来对它进行括号化，那么可以将此最优解代入$A_iAi+1...Aj$的最优解中，代替原来对子链$A_iA_{i+1}...Ak$进行括号化的方案（比$A_iA_{i+1}...Ak$最优解的代价更低），显然，这样得到的解比$A_iA_{i+1}...Aj$代价更低：产生矛盾。对子链$A_{k+1}A_{k+2}...Aj$,我们有相似的结论。因此该问题具有最优子结构。

题目 (UD: 15.3.5)
对15.1节的钢条切割问题加入限制条件：假定对于每种钢条长度$i(i=1,2,...,n-1)$,最多允许切割出l_i段长度为i的钢条。证明：15.1节所描述的最优子结构性质不再成立。

解答：
假设该问题仍然具有最优子结构性质。因为子问题的解之间产生了关联。完成首次切割后，两段钢条应采取各自长度独立求解所得的最优解。即使切割后的两段钢条的最优切割方案都满足切割出的长度为i的钢条不多于l_i段，但无法保证合并后长度为i的钢条总数不多于l_i。
反例：切割一根总长度为4的钢条，令l_2=1，其他l_i无所谓。则根据最优子结构假设和15.1中的计算，$r_4 = \max_{1 \leq i \leq 4}(p_i + r_{4-i}) = r_2 + r_2 = 10$。求解$r_2$时都保证了长度为2的钢条不多于1条，但是合并得到的r_4的答案中长度为2的钢条有2条，不合题意。

题目 (UD: 15.3.6)
假定你希望兑换外汇，你意识到与其直接兑换，不如进行多种外币的一系列兑换，最后兑换到你想要的那种外币，可能会获得更大收益。假定你可以交易n种不同的货币，编

号为$1,2,...,n$，兑换从1号货币开始，最终兑换为n号货币。对每两种i和j，给定汇率r_{ij}，意味着你如果有d个单位的货币i，可以兑换为dr_{ij}个单位的货币j。进行一系列的交易需要支付一定的佣金，金额取决于交易的次数。令c_k表示k次交易需要支付的佣金。证明：如果对所有$k=1,2,...,m,c_k=0$，那么寻找最优兑换序列的问题具有最优子结构性质。然后请证明：如果佣金c_k为任意值，那么问题不一定具有最优子结构性质。

解答：

假设：只能从编号小的货币兑换编号大的货币。

先证：如果对所有$k=1,2,...,m,c_k=0$，那么寻找最优兑换序列的问题具有最优子结构性质。

那么这个问题可以转化为：从$d_{ij}(1\le i,j\le n)$中选出若干个数相乘，满足第一数的第一个下标为1，最后一个数的第二个下标为n且除第一个数外每一个数的第一个下标与前一个数的第二个下标相等，求这样的乘积的最大值。

令p_j表示兑换j号货币时获得的最大收益。

假设兑换j号货币之前手里货币全部为i号，则计算兑换到j号货币的最大收益p_j应该使用兑换到i号货币的最大收益p_i的兑换方案，即独立求解p_i所得的最优解。如果不采用独立求解p_i所得的最优解，那么可以将此最优解代入p_j的最优解中代替原来对p_i的兑换方案，显然这样得到的收益比p_j的收益更高，产生了矛盾。因此该问题具有最优子结构性质。

可以得到递推关系式为：

$$p_j=\begin{cases}1 & j=1\\ \max\limits_{1\le i<j}\{p_i\cdot d_{ij}\} & \end{cases} \tag{1}$$

再证：如果佣金c_k为任意值，那么问题不一定具有最优子结构性质。

如果佣金c_k为任意值，那么子问题之间就产生了关联，该问题不再具有最优子结构的性质。假设该问题具有最优子结构，可以构造如下的反例：

假设有1个单位的1号货币需要兑换，令$c_1=0$,对任意$k\ge 2$, $c_k>p_j$（前一小问条件下的最优解），对任意$1\le i,j\le n$,$d_{ij}\ge d_{1n}>0$。

根据最优子结构假设，兑换n号货币时会采用从1号货币兑换到k号$(1\le k<n)$的这一子问题的最优解，再兑换到n号。从赋值情况可以断定任意$1<k<n$都可以成为子问题的最优解，但是当两个子问题最优解合并为整个问题的最优解时，减去c_2能让收益一下变成负数，显然是直接从1号直接兑换至n号收益更大一些，所以最优子结构假设不成立，该问题不具有最优子结构。

而前一问已经证明存在$k=1,2,...,m,c_k=0$的情况使该问题具有最优子结构，因此当c_k为任意值时，该问题不一定具有最优子结构。

题目 (UD: 15.4.3)

设计LCS-LENGTH的带备忘的版本，运行时间为$O(mn)$。

Algorithm 4 LOOKUP-LCS(X, Y, i, j)

解答：

let $c[0...m, 0...n]$ and $b[1...m, 1...n]$ be new tables

for $i = 1$ to m **do**

  for $j = 1$ to n **do**

    $c[i, j] = -\infty$

  end for

end for

for $j = 1$ to n **do**

  $c[i, 0] = 0$

end for

for $i = 1$ to m **do**

  $c[0, j] = 0$

end for

return LOOKUP-LCS(X, Y, m, n)

if $c[i, j] > -\infty$ **then**

  return c[i,j]

end if

if $x_i == y_j$ **then**

  $c[i, j]$=LOOKUP-LCS($X, Y, i-1, j-1$)+1

  $b[i, j] =$ " $\nwarrow$ "

else

  $c[i-1, j]$=LOOKUP-LCS($X, Y, i-1, j$)

  $c[i, j-1]$=LOOKUP-LCS($X, Y, i, j-1$)

  if $c[i-1, j] \geq c[i, j-1]$ **then**

    $c[i, j] = c[i-1, j]$

    $b[i, j] =$ " $\uparrow$ "

  else

    $c[i, j] = c[i, j-1]$

    $b[i, j] =$ " $\leftarrow$ "

  end if

end if

return $c[i, j]$

题目 (UD: 15.4.5)

设计一个$O(n^2)$时间的算法，求一个n个数的序列的最长单调递增子序列。

解答：

根据微积分课上的定义，递增理解为每个数都大于等于前一个数

输入为原序列$A[1...n]$，输出为最长单调递增子序列及其长度

这个问题具有最优子结构（题目无要求故证明略），使用动态规划来求解该问题

使用$c[1...n]$数组来存储以i结尾的LIS的长度,可得如下公式：

$$c[i] = \begin{cases} 1 & i == 1 \\ c[i-1]+1 & A[i] \geq A[i-1] \\ \max\limits_{1 \leq j \leq i-1, A[j] \leq A[i]} \{c[j]\} + 1 & A[i] < A[i-1] \end{cases} \tag{2}$$

二重循环时间复杂度为$O(n^2)$

全部计算完后再$O(n)$遍历该数组找出最大值

使用表格$b[1...n, 1...n]$存储LIS，$b[i][j] = 1$当且仅当$A[j]$在以i为结尾的LIS里，否则为0

题目 (UD: 15.5.1)

设计伪代码CONSTRUCT-OPTIMAL-BST($root$)，输入表为$root$，输出是最优二叉搜索树的结构。例如，对图15-10中的$root$表，应输出（略）与图15-9(b)中的最优二叉搜索树对应。

解答：

输入是表$root$，假设表的长宽n也是已知的（又不一定是C）

输入里都没有k和d是闹哪出...我默认也能用了

直接调用BST一章中的TREE-INSERT(T, z)函数，将z结点插入到树T中相应的位置上

大体思路：使用一个递归函数CONSTRUCT-OPTIMAL-BST$(root, i, j)$把k_i按一定顺序构造成一棵BST，然后把d_i插入到相应的位置上,构造出一棵完整的BSTT，然后调用函数OUTPUT-OPTIMAL-BST先序遍历递归输出。

我由于一开始看错题目了所以想出了这个算法，感觉消耗的时空间都有点大，但是重新思考的时候发现很难跳出原来的思维，而且题目里并没有对时空间复杂度有所要求，所以我就偷懒了orz我认为是会有更好的算法的，期待知道

Algorithm 5 LIS(A)

```
let c[1...n] be a new array
let b[1...n, 1...n] be a new table
for i = 2 to n do
  c[i] = i
end for
c[1] = 1
for i = 1 to n do
  for j = 1 to n do
    b[i, j] = 0
  end for
end for
for i = 2 to n do
  if A[i] ≥ A[i − 1] then
    c[i] = c[i − 1] + 1
  else
    q = −∞
    for j = 1 to i − 1 do
      if A[j] ≤ A[i] and c[j] > q then
        q = c[j]
        temp = j
      end if
    end for
    c[i] = q + 1
    for j = 1 to i − 1 do
      b[i, j] = b[temp, k]
    end for
    b[i, i] = 1
  end if
end for
ans = −1
for i = 1 to n do
  if c[i] > ans then
    ans = c[i]
    tail = i
  end if
end for
return b[i, 1...n] and ans
```

Algorithm 8 OUTPUT-OPTIMAL-BST(t)

CONSTRUCT-OPTIMAL-BST-K($root, i, j, T$)
for $i = 0$ to n **do**
  BST-INSERT(T, d_i)
end for
print(T.root"为根")
OUTPUT-OPTIMAL-BST($T.root.left$)
OUTPUT-OPTIMAL-BST($T.root.right$)

$r = root[i, j]$
TREE-INSERT(T, k_r)
if $i == j$ **then**
  return
end if
CONSTRUCT-OPTIMAL-BST-K($root, i, r-1, T$)
CONSTRUCT-OPTIMAL-BST-K($root, r+1, j, T$)

print($t.left$"为"t"的左孩子")
OUTPUT-OPTIMAL-BST($t.left$)
print($t.right$"为"t"的右孩子")
OUTPUT-OPTIMAL-BST($t.right$)
return

题目 (UD: 15.4)

（整齐打印）考虑整齐打印问题，即在打印机上用等宽字符打印一段文本。输入文本为n个单词的序列，单词长度分别为$l_1, l_2, ..., l_n$个字符。我们希望将此段文本整齐打印在若干行上，每行最多M个字符。“整齐”的标准是这样的。如果某行包含第i到第$j(i \leq j)$个单词，且单词间隔为一个空格符，则行尾的空格符数量为$M - j + i - \sum\limits_{k=i}^{j} l_k$，此值必须为非负的，否则一行内无法容纳这些单词。我们希望能最小化所有行的（除最后一行外）额外空格数的立方之和。设计一个动态规划算法，在打印机上整齐打印一段n个单词的文本。分析算法的时间和空间复杂性。

解答：

记$c[i]$为单词$1-i$排列出的最高整齐度（行尾空格数立方和最小值），假设到第i个单词为止的最后一行从j个单词开始，则$c[i]$的值应为前$j-1$个单词的最优解$c[j-1]$加上最后一行产生的空格数的立方$(M - i + j - \sum\limits_{k=j}^{i} l_k)^3$。（该最优子结构可使用反证法证明，此处略）。可以得到递推关系式为：

$$c[i] = \begin{cases} 0 & i = 0 \\ \min\limits_{1 \leq j \leq i} \{c[j-1] + (M - i + j - \sum\limits_{k=j}^{i} l_k)^3\} & 1 \leq i \leq n \end{cases} \quad (3)$$

其中空格长度可以预先计算好$s[i]$(到第i个单词为止的总长度)后只需要常数时间就可以计算。(空间换时间)

用$p[i]$记录从1到i单词中最后一行开头的单词的序号。

最后调用一个$OUTPUT-NEATLY$函数来输出最优排版方案。

算法因为排版问题跑到下一页去了！

时间复杂度：

$s[0..n]$的计算可在$\Theta(n)$时间内完成，每一次循环的操作都可以在常数时间内完成，二重循环一共是

$\sum\limits_{i=1}^{n} \sum\limits_{j=1}^{i} \Theta(n)$

$= \Theta(n) \sum\limits_{i=1}^{n} \frac{i(i+1)}{2}$

$= \Theta(n) \sum\limits_{i=1}^{n} i^2 + \sum\limits_{i=1}^{n} i$

$= \Theta(n) + \frac{n(n+1)(2n+1)}{6} + \frac{n(n+1)}{2}$

$= \Theta(n) + \Theta(n^3) + \Theta(n^2)$

$= \Theta(n^3)$

因此这个算法的时间复杂度为$\Theta(n^3)$

空间复杂度：

算法中使用到空间只有一些变量和三个一维数组，因此空间复杂度为$\Theta(n)$.

Algorithm 10 OUTPUT-NEATLYp, i

```
let c[0...n],s[0...n] and p[0...n] be new arrays
c[0] = 0
s[0] = 0
p[1] = 0
for i = 1 to n do
  s[i] = s[i − 1] + l_i
end for
for i = 1 to n do
  q = +∞
  for j = 1 to i do
    sp = M − i + j − s[i] + s[j − 1]
    if sp < 0 then
      break
    end if
    if c[j − 1] + sp^3 < q then
      q = c[j − 1] + sp^3
      p[i] = j
    end if
  end for
  c[i] = q
end for
OUTPUT-NEATLY(p, n)
if i == 0 then
  return
end if
OUTPUT-NEATLY(p, p[i] − 1)
for j = p[i] to i do
  print(word_j)
end for
print(endl)
```