第 3-1 讲: 动态规划

姓名：林凡琪　　**学号：**211240042

评分：_____　　**评阅：**_____

2022 年 9 月 7 日

请独立完成作业，不得抄袭。
若得到他人帮助, 请致谢。
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。

1　作业 (必做部分)

题目 1 (TC 15.1-1)

解答：
For $n = 0$, this holds since $2^0 = 1$
For $n > 0$, substituting into the recurrence, we have
we have

$$\begin{aligned} T(n) &= 1 + \sum_{j=0}^{n-1} 2^j \\ &= 1 + (2^n - 1) \\ &= 2^n \end{aligned}$$

题目 2 (TC 15.1-3)

解答：
We can modify BOTTOM-UP-CUT-RODBOTTOM-UP-CUT-ROD algorithm from section 15.1 as follows:

```
MODIFIED-CUT-ROD(p, n, c)
let r[0..n] be a new array
r[0] = 0
for j = 1 to n
    q = p[j]
    for i = 1 to j - 1
        q = max(q, p[i] + r[j - i] - c)
```

```
        r[j] = q
    return r[n]
```

We need to account for cost c on every iteration of the loop in lines 5-6 but the last one, when i = j (no cuts).

We make the loop run to j - 1 instead of j, make sure c is subtracted from the candidate revenue in line 6, then pick the greater of current best revenue q and p[j] (no cuts) in line 7.

题目 3 (TC 15.2-2)

解答:

```
MATRIX-CHAIN-MULTIPLY(A, s, i, j)
if i == j
    return A[i]
if i + 1 == j
    return A[i] * A[j]
b = MATRIX-CHAIN-MULTIPLY(A, s, i, s[i, j])
c = MATRIX-CHAIN-MULTIPLY(A, s, s[i, j] + 1, j)
return b * c
```

题目 4 (TC 15.2-4)

解答:

The vertices of the subproblem graph are the ordered pair v_{ij} , where $i \leq j$.

If $i = j$, the vertex v_{ij} has no output edge. If $i < j$, for each k, s.t. $i \leq k < j$, the subproblem graph contains edges (v_{ij}, v_{ik}) and $(v_{ij}, v_{k+1,j})$, and these edges indicate that to solve the subproblem of optimally parenthesizing the product $A_i \cdots A_j$, we need to solve subproblems of optimally parenthesizing the products $A_i \cdots A_k$ and $A_{k+1} \cdots A_j$

The number of vertices is

$$\sum_{i=1}^{n} \sum_{j=i}^{n} = \frac{n(n+1)}{2}.$$

.

The number of edges is

$$\sum_{i=1}^{n} \sum_{j=i}^{n} (j-i) = \frac{(n-1)n(n+1)}{6}.$$

题目 5 (TC 15.3-3)

解答：
Yes, this problem also exhibits optimal substructure. If we know that we need the subproduct $(A_l \cdot A_r)$, then we should still find the most expensive way to compute it —otherwise, we could do better by substituting in the most expensive way.

题目 6 (TC 15.3-5)

解答：
The optimal substructure property doesn't hold because the number of pieces of length ii used on one side of the cut affects the number allowed on the other. That is, there is information about the particular solution on one side of the cut that changes what is allowed on the other.

To make this more concrete, suppose the rod was length 44, the values were $l_1 = 2, l_2 = l_3 = l_4 = 1$, and each piece has the same worth regardless of length. Then, if we make our first cut in the middle, we have that the optimal solution for the two rods left over is to cut it in the middle, which isn't allowed because it increases the total number of rods of length 11 to be too large.

题目 7 (TC 15.3-6)

解答：
First we assume that the commission is always zero. Let k denote a currency which appears in an optimal sequence s of trades to go from currency 1 to currency n. p_k denote the first part of this sequence which changes currencies from 1 to k and q_k denote the rest of the sequence. Then p_k and q_k are both optimal sequences for changing from 1 to k and k to nnrespectively. To see this, suppose that p_k wasn't optimal but that p_k was. Then by changing currencies according to the sequence $p'_k q_k$ we would have a sequence of changes which is better than ss, a contradiction since ss was optimal. The same argument applies to q_k

Now suppose that the commissions can take on arbitrary values. Suppose we have currencies 1 through 6, and $r_{12} = r_{23} = r_{34} = r_{45} = 2$, $r_{13} = r_{35} = 6$, and all other exchanges are such that $r_{ij} = 100$ Let $c_1 = 0$, $c_2 = 1$, and $c_k = 10$ for $k \geq 3$.

The optimal solution in this setup is to change 1 to 3, then 3 to 5, for a total cost of 13. An optimal solution for changing 1 to 3 involves changing 1 to 2 then 2 to 3, for a cost of 5, and an optimal solution for changing 3 to 5 is to change 3 to 4 then 4 to 5, for a total cost of 5. However, combining these optimal solutions to subproblems means making more exchanges overall, and the total cost of combining them is 1818, which is not optimal.

题目 8 (TC 15.4-3)

解答：

```
MEMOIZED-LCS-LENGTH(X, Y, i, j)
if c[i, j] > -1
```

```
        return c[i, j]
    if i == 0 or j == 0
        return c[i, j] = 0
    if x[i] == y[j]
        return c[i, j] = LCS-LENGTH(X, Y, i - 1, j - 1) + 1
    return c[i, j] = max(LCS-LENGTH(X, Y, i - 1, j), LCS-LENGTH(X, Y, i, j - 1))
```

题目 9 (TC 15.4-5)

解答:

Given a list of numbers L, make a copy of L called L' and then sort L' .

```
PRINT-LCS(c, X, Y)
  n = c[X.length, Y.length]
  let s[1..n] be a new array
  i = X.length
  j = Y.length
  while i > 0 and j > 0
      if x[i] == y[j]
          s[n] = x[i]
          n = n - 1
          i = i - 1
          j = j - 1
      else if c[i - 1, j]   c[i, j - 1]
          i = i - 1
      else j = j - 1
  for i = 1 to s.length
      print s[i]
```

```
MEMO-LCS-LENGTH-AUX(X, Y, c, b)
  m = |X|
  n = |Y|
  if c[m, n] != 0 or m == 0 or n == 0
      return
  if x[m] == y[n]
      b[m, n] =
      c[m, n] = MEMO-LCS-LENGTH-AUX(X[1..m - 1], Y[1..n - 1], c, b) + 1
  else if MEMO-LCS-LENGTH-AUX(X[1..m - 1], Y, c, b)   MEMO-LCS-LENGTH-AUX(X, Y[1..n - 1], c, b)
      b[m, n] = ↑
      c[m, n] = MEMO-LCS-LENGTH-AUX(X[1..m - 1], Y, c, b)
  else
      b[m, n] = ←
      c[m, n] = MEMO-LCS-LENGTH-AUX(X, Y[1..n - 1], c, b)
```

```
MEMO-LCS-LENGTH(X, Y)
```

```
let c[1..|X|, 1..|Y|] and b[1..|X|, 1..|Y|] be new tables
MEMO-LCS-LENGTH-AUX(X, Y, c, b)
return c and b
```

Then, just run the LCS algorithm on these two lists. The longest common subsequence must be monotone increasing because it is a subsequence of L' which is sorted. It is also the longest monotone increasing subsequence because being a subsequence of L' only adds the restriction that the subsequence must be monotone increasing. Since $|L| = |L'| = n$, and sorting L can be done in $o(n^2)$ time, the final running time will be $O(|L||L'|) = O(n^2)$.

题目 10 (TC 15.5-1)

解答:

```
CONSTRUCT-OPTIMAL-BST(root, i, j, last)
if i == j
    return
if last == 0
    print root[i, j] + "is the root"
else if j < last
    print root[i, j] + "is the left child of" + last
else
    print root[i, j] + "is the right child of" + last
CONSTRUCT-OPTIMAL-BST(root, i, root[i, j] - 1, root[i, j])
CONSTRUCT-OPTIMAL-BST(root, root[i, j] + 1, j, root[i, j])
```

---

2 作业 (选做部分)

题目 1 (TC Problem 15-4: Printing neatly)

解答:

---

3 Open Topics

Open Topics 1 (通信系统)
某个通信系统由 n 个设备串联构成，每个设备可能有多个厂商生产，均有带宽和价格参数。系统的总带宽决定于某个设备的最小带宽，总价格是各个设备的价格总和。请你设计一个算法，以 “带宽/造价” 为最优目标，确定该通信系统的构成

请按照 “最优子结构确定、确定递归表达式、非递归实现” 步骤完成设计和讲解.

Open Topics 2 (TC Problem 15-3: Bitonic euclidean traveling-salesman problem)

4 反馈