论题 2-12 作业

姓名：陈劭源　　　　学号：161240004

1 [TC] Problem 15.1-1

We use mathematical induction to prove this.

For the base step, $n = 1$, we have the initial condition $T(0) = 1$, and $2^0 = 1$, so $T(n) = 2^n$ holds for $n = 1$.

For the induction step, assume that for all non-negative integer $k < n$, $T(k) = 2^k$ holds. Then we have

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

By mathematical induction, we have $T(n) = 2^n$ for all non-negative integer n.

2 [TC] Problem 15.1-3

The recurrence should be changed to

$$r_n = \max_{1 \le i \le n} (p_i + r_{n-i} - c)$$

with initial condition $r_0 = c$.

```
MODIFIED-CUT-ROD($p, n$)
    // assume $n > 0$
 1  let $r[0..n]$ and $s[0..n]$ be new arrays
 2  $r[0] = c$
 3  for $j = 1$ to $n$
 4      $q = -\infty$
 5      for $i = 1$ to $j$
 6          if $q < p[i] + r[j-i] - c$
 7              $q = p[i] + r[j-i] - c$
 8              $s[j] = i$
 9      $r[j] = q$
10  return $r$ and $s$
```

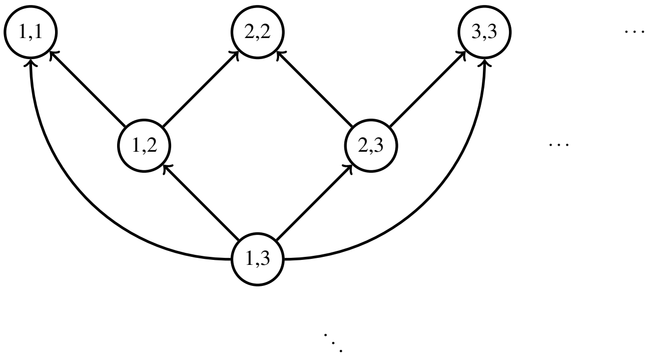
3 [TC] Problem 15.2-2

```
MATRIX-CHAIN-MULTIPLY($A, s, i, j$)
1  if $i == j$
2      return $A[i]$
3  else
4      return MATRIX-CHAIN-MULTIPLY($A, s, i, s[i,j]$) $\cdot$ MATRIX-CHAIN-MULTIPLY($A, s, s[i,j]+1, j$)
```

The implementation of matrix multiplication is shown below.

```
MATRIX-MULTIPLY$(A,B,m,n,l)$
1  Let $C$ be a new $m \times l$ matrix
2  for $i = 1$ to $m$
3      for $j = 1$ to $l$
4          $C[i][j] = 0$
5          for $k = 1$ to $n$
6              $C[i][j] = C[i][j] + A[i][k] \cdot B[k][j]$
7  return $C$
```

4 [TC] Problem 15.2-4



It has $\frac{n(n+1)}{2}$ vertices and $\sum_{i=0}^{n-1} 2n(i-n) = \frac{(n-1)n(n+1)}{3}$ edges.
The edges are $\langle (i,j),(i,k) \rangle$ and $\langle (i,j),(k+1,j) \rangle$, where $1 \le i < j \le n$ and $i \le k < j$.

5 [TC] Problem 15.3-3

Yes. We have to choose a split with maximized number of scalar multiplications, each choice leaving two subproblems to be solved. Now we are going to prove that this is an optimal substructure.

Suppose that one of the subproblem solutions used in the optimal solution is not optimal, i.e. there exists some solution to a subproblem which needs more scalar multiplications. Replacing the solution for the subproblem with the optimal one gives a solution with more scalar multiplications, which leads to contradiction. Therefore, this problem exhibit optimal substructure.

6 [TC] Problem 15.3-5

Here is a counterexample, where the final solution is optimal, but the solution to its subproblem is not optimal.

Assume that the pieces of length 1, 2, 3, 4, 5 are worth 5, 1, 1, 1, 1, respectively, and each kind of piece should appear no more than twice. Consider cutting up a rod of length 5 to pieces to obtain the maximum revenue. We can easily verify that cutting the rod into pieces of length 1, 3, 1 obtains the maximum revenue. In this solution, the subproblem is to cut a rod of length-4, whose solution is 1, 3, which is not optimal (1, 1, 2 is an optimal solution). Therefore, the optimal-substructure property described in Section 15.1 no longer holds.

7 [TC] Problem 15.3-6

When $c_k = 0$, let d_i denote the maximum amount of currency i that we can achieve by a sequence of exchanges. d_1 is the amount of currency 1 you initially have. Then d_j is the maximum of $d_i \times r_{ij}$ with respect to i, where j is the index of the currency which can be exchanged for currency j. Suppose that the subproblem solution, say d_{i_0}, used in the optimal solution is not optimal, i.e. there exists d_{i_1}, such that $d_{i_1} > d_{i_0}$. Replacing d_{i_0} for d_{i_1} gives a better sequence of exchanges, which lead to contradiction. Therefore, the problem exhibits optimal structure when $c_k = 0$.

When c_k are arbitrary values, we can find a counterexample. Assume that you initially have 10.0 units of currency 1, the exchange rates are $r_{1,2} = 2.0, r_{1,3} = 3.0, r_{1,4} = 0.0, r_{2,3} = 2.0, r_{2,4} = 0.0, r_{3,4} = 1.0$, and the commissions are $c_1 = 0.0, c_2 = 5.0, c_3 = 20.0$. We can verify that $\langle 1,3\rangle, \langle 3,4\rangle$ is a best sequence of exchanges. However, $\langle 1,3\rangle$ is not the optimal solution to the subproblem exchanging currency 1 for currency 3. $\langle 1,2\rangle, \langle 2,3\rangle$ is a better solution. Therefore, when commissions are arbitrary, the problem does not necessarily exhibit optimal structure.

8 [TC] Problem 15.4-3

```
MEMOIZED-LCS-LENGTH($X, Y$)
1  $m = X.length$
2  $n = Y.length$
3  for $i = 0$ to $m$
4      for $j = 0$ to $n$
5          $c[i, j]$ = NULL
6          $b[i, j]$ = NULL
7  LOOKUP-LENGTH($X, Y, m, n$)
8  return $c$ and $b$
```

```
LOOKUP-LENGTH($X, Y, i, j$)
 1  if $c[i, j] \neq$ NULL
 2      return $c[i, j]$
 3  if $i == 0$ or $j == 0$
 4      $c[i, j] = 0$
 5  elseif $X[i] == Y[i]$
 6      $c[i, j]$ = LOOKUP-LENGTH($X, Y, i-1, j-1$) + 1
 7      $b[i, j]$ = "$\nwarrow$"
 8  elseif $c[i-1, j] \geq c[i, j-1]$
 9      $c[i, j]$ = LOOKUP-LENGTH($X, Y, i-1, j$)
10      $b[i, j]$ = "$\uparrow$"
11  else
12      $c[i, j]$ = LOOKUP-LENGTH($X, Y, i, j-1$)
13      $b[i, j]$ = "$\leftarrow$"
14  return $c[i, j]$
```

9 [TC] Problem 15.4-5

```
LIS(X)
1  $n = X.length$
2  let $b[1..n, 1..n]$ and $m[1..n, 1..n]$ be new tables // $m[i,j] = \min_{\substack{1 \le k_1 < \cdots < k_j \le i \\ X[k_1] \le \cdots \le X[k_j]}} X[k_j]$
3  $m[1,1] = X[1]$
4  for $i = 2$ to $n$
5      $m[i,1] = \text{MIN}(X[i], m[i-1,1])$
6  $i = 1$
7  while $m[i,n] \neq +\infty$
8      $i = i+1$
9      $m[i,1] = +\infty$
10     for $j = 2$ to $n$
11         if $X[j] \ge m[i-1, j-1]$ and $X[j] < m[i, j-1]$
12             $m[i,j] = X[j]$
13             $b[i,j] = $ “$\nwarrow$”
14         else
15             $m[i,j] = m[i, j-1]$
16             $b[i,j] = $ “$\uparrow$”
17 return $b$ and $i-1$ // $i-1$ is the length of LIS
```

This procedure fills tables m and b whose sizes are at most $n \times n$, so the running time is $O(n^2)$.

```
PRINT-LIS(X, b, i, j)
1  if $i == 1$
2      print $X[i]$
3      return
4  if $b[i,j] ==$ “$\nwarrow$”
5      PRINT-LIS$(X, b, i-1, j-1)$
6      print $X[i]$
7  else
8      PRINT-LIS$(X, b, i, j-1)$
```

The initial call is PRINT-LIS$(X, b, l, X.length)$, where l is the length of LIS.

10 [TC] Problem 15.5-1

```
CONSTRUCT-OPTIMAL-BST(root)
1  $n =$ length of the first dimension of $root$
2  print “$k$” $_{root[1,n]}$ “is the root”
3  RECURSIVE-CONSTRUCT-OPTIMAL-BST$(root, 1, root[1,n]-1,$ LEFT$)$
4  RECURSIVE-CONSTRUCT-OPTIMAL-BST$(root, root[1,n]+1, n,$ RIGHT$)$
```

```
RECURSIVE-CONSTRUCT-OPTIMAL-BST($root, i, j, p$)
1  if $i > j$
2      if $p$ == LEFT
3          print "$d$" $_j$ "is the left child of $k$" $_{j+1}$
4      else
5          print "$d$" $_j$ "is the right child of $k$" $_{i-1}$
6  else
7      if $p$ == LEFT
8          print "$k$" $_{root[i,j]}$ "is the left child of $k$" $_{j+1}$
9      else
10         print "$k$" $_{root[i,j]}$ "is the right child of $k$" $_{i-1}$
11     RECURSIVE-CONSTRUCT-OPTIMAL-BST($root, i, root[i,j]-1$, LEFT)
12     RECURSIVE-CONSTRUCT-OPTIMAL-BST($root, root[i,j]+1, j$, RIGHT)
```

11 [TC] Problem 15-4

Assume that for all $1 \le i \le n$, $1 \le l_n \le M$. Let $m[i]$ denote the minimum sum over several lines containing the first i words, including the last line. We have the following recursive formula:

$$m[i] = \min_{\substack{1 \le j \le i \\ l_{i-j+1}+\cdots+l_i+j-1 \le M}} m[i-j] + \left(M - j + 1 - \sum_{k=i-j+1}^{i} l_k\right)^3$$

with initial condition $m[0] = 0$.

Since the number of the extra space characters in the last line should not be added, the final answer is

$$\min_{l_k+\cdots+l_n+n-k \le M} m[k-1]$$

Then procedure PRINTING-NEATLY(l, n, M) on the next page, returns a stack T, storing the indices of the words, after each of which there is a new line. The running time is $O(n^2)$, and the space requirement is $\Theta(n)$.

```
PRINTING-NEATLY($l, n, M$)
let $m[0..n]$ be new array
$m[0] = 0$
for $i = 1$ to $n$
    $S = M + 1$
    $m[i] = +\infty$
    for $j = 1$ to $i$
        $S = S - l_{i-j+1} - 1$
        if $S < 0$
            break
        if $m[i-j] + S^3 < m[i]$
            $m[i] = m[i-j] + S^3$
            $s[i] = i - j$
$a = +\infty$
$S = M + 1$
for $i = n - 1$ downto $0$
    $S = S - l_{i+1} - 1$
    if $S < 0$
        break
    if $m[i] < a$
        $a = m[i]$
        $j = i$
let $T$ be a new empty stack
while $j \neq 0$
    $T.push(j)$
    $j = s[j]$
return $T$
```