Problem Set 6 Answer Sheet

151220131 谢旻晖

Problem 7.1

假设集合 A 中的 $s_1, s_2...s_n$ 已经按升序排好序,$\sum A$ 为 A 中所有元素之和。

设 $f(i, sum)$ 是这样的一个函数，指示了集合$s_1, s_2, ...s_i$ 中是否存在子集使得元素的和为 sum, 如果存在返回 true, 否则返回 false.

如果 $i <= 0$ or $sum < 0$ or $sum > \sum A$, 则

$$f(i, sum) = false$$

其余情况下易得转移方程为

$$f(i, sum) = (s_i == sum || f(i-1, sum) || f(i-1, sum - s_i))$$

据此可写出动态规划算法,计算 $f(i, sum)$ 时,从 i:1->n,sum:0->S 自底向上计算。最终 $f(n, S)$ 为所求答案。

Problem 7.2

设 $f(n)$ 为将 n 变为 1 最少需要的操作,$f(n)$ 的值为 $f(n-1)$、$f(n/2)$(如果 n 是偶数)、$f(n/3)$(如果 n 是 3 的倍数) 三者中最小的加一。由此可得转移方程为:
首先,

$$temp = 1 + f(n-1)$$

当 $n\%2 == 0$ 时,

$$temp = min(temp, 1 + f(\frac{n}{2}))$$

当 $n\%3 == 0$ 时,

$$temp = min(temp, 1 + f(\frac{n}{3}))$$

最终 $temp$ 的值是 $f(n)$ 的值:

$$f(n) = temp$$

据此可写出动态规划算法，$f(1) = 0$, 计算 $f(i)$ 时, 从 i:2→n 自底向上计算。最终 $f(n)$ 为所求答案。

Problem 7.3

新建数组 $f[1..n]$,$pre[1..n]$,$f[i]$ 记录下当前以 $A[i]$ 结束的最长非递减子序列的长度。$pre[i]$ 用于记录下以 $A[i]$ 为结尾的最大非递减子序列的前驱。当新的一个元素加入时，判断之前的所有元素是否小于等于当前元素，取其中小于的，也即可以形成最长单调非递减子序列的元素，更新 pre 和 f.

算法见 **Algorithm 1**

Algorithm 1 LONGEST_INCREASING_SUBsq

```
f[1...n] = {1}
pre[1...n] = {1, 2...n}
for i in [2, n] do
    for j in [1, i − 1] do
        if A[i] ≥ A[j] and f[i] < f[j] + 1 then
            f[i] = f[j] + 1
            pre[i] = j
        end if
    end for
end for
find the max element in f[ ] is f[max]
//输出序列，利用栈将序列倒序
stack.push(f(max))
while pre[max]! = max do
    max = pre[max]
    stack.push(f(max))
end while
此时栈中的序列即为最长非递减子序列
return stack
```

Problem 7.4

1.

$A[1...n]$ 中的所有的 0 自然的将数组分为了若干个子数组，求出所有子数组中乘积最大的即可。

2.

利用两个数组 $min[1...n]$,$max[1...n]$,$min[i]$ 和 $max[i]$ 分别存储以 $A[i]$ 结尾的乘积最小和乘积最大的子数组。

易得转移方程为:

$$max[i] = \begin{cases} A[1] & i == 1 \\ max(A[i], min[i-1] * A[i], max[i-1] * A[i]) & else \end{cases}$$

$$min[i] = \begin{cases} A[1] & i == 1 \\ min(A[i], min[i-1] * A[i], max[i-1] * A[i]) & else \end{cases}$$

其中注意 $A[i]$ 自己也可以构成一个平凡的数组。

最后算法求出 max[1..n] 中的最大值即为最大的乘积, 并向前进行重构解，算法伪代码见 **Algorithm 2**。

Algorithm 2 MAX_PRODUCT_SUBARRAY

```
max[1...n]
min[1...n]
max[1] = min[1] = A[1]
for i=2 to n do
    max[i] = max(A[i], min[i-1] * A[i], max[i-1] * A[i])
    min[i] = min(A[i], min[i-1] * A[i], max[i-1] * A[i])
end for
find the max value in max[ ] is max[index]
从 index 往前开始对 A[ ] 连乘，乘到 A[small] 时总乘积突然变小，则题目所求的数组为 A[small + 1, small + 2, ..., index]
return A[small + 1, small + 2, ..., index]
```

3.

2. 中的方法并不失一般性。

Problem 7.5

1.

LCS 问题，直接利用书上的解法求解。定义 $c[i, j]$ 表示 X_i 和 Y_j 的 LCS 长度，如果 $i = 0$ 或 $j = 0$,LCS 长度为 0. 有如下转移方程:

$$c[i, j] = \begin{cases} 0 & i = 0 \ or \ j = 0 \\ c[i-1, j-1] + 1 & if \ X_i == Y_j \\ max(c[i, j-1], c[i-1, j]) & if \ X_i \neq Y_j \end{cases}$$

算法见 **Algorithm 3**, 输出 LCS 的程序见 **Algorithm 4**

2.

只需将转移方程改为

$$c[i, j] = \begin{cases} 0 & i = 0 \ or \ j = 0 \\ c[i, j-1] + 1 & if \ X_i == Y_j \\ max(c[i, j-1], c[i-1, j]) & if \ X_i \neq Y_j \end{cases}$$

Algorithm 3 LCS_LEN

```
1: b[1...m, 1...n]
2: c[0...m, 0...n]
3: for i=1 to m do
4:    c[i, 0] = 0
5: end for
6: for j=0 to n do
7:    c[0, j] = 0
8: end for
9: for i=1 to m do
10:   for j=1 to n do
11:      if X_i == Y_j then
12:         c[i, j] = c[i − 1, j − 1] + 1
13:         b[i, j] =↖
14:      else if c[i − 1, j] ≥ c[i, j − 1] then
15:         c[i, j] = c[i − 1, j]
16:         b[i, j] =↑
17:      else
18:         c[i, j] = c[i, j − 1]
19:         b[i, j] =←
20:      end if
21:   end for
22: end for
```

Algorithm 4 PRINT_LCS(i,j)

```
1: if i == 0 or j == 0 then
2:    return
3: end if
4: if b[i, j] ==↖ then
5:    PRINT_LCS(i − 1, j − 1)
6:    print X_i
7: else if b[i, j] ==↑ then
8:    PRINT_LCS(i − 1, j)
9: else
10:   PRINT_LCS(i, j − 1)
11: end if
```

算法也对应位置作改动即可，不再赘述。

3.

1. 和 2. 的综合体.

利用一个数组 count[1..m] 记录下 X 中字符出现的次数，初始化为 0。当次数小于 k 时，$X_i == Y_j$ 时 $c[i,j] = c[i,j-1]+1$，同时对应字符出现次数加一; 当次数大于等于 k 时,，$X_i == Y_j$ 时 $c[i,j] = c[i-1,j-1]+1$.

Problem 7.6

求 A 和 B 的 LCS 长度为 l, 则最短公共超序列长度为 $m+n-l$.

Problem 7.8

首先因为前向和后向的字符串不能够重叠, 双子串一定是在原串的左边一半中和右边一半中, 令 $m = \lfloor \frac{n}{2} \rfloor$, 所以将字符串的两个子串 $T[1...m]$ 和 $T[m+1...n]$ 取出，将第二个子串倒序，令倒序后的字符串为 $M[1...m]$，这样原问题就被转化为求 $T[1..m]$ 和 $M[1...m]$ 的相同的最长连续子串的长度。

$c[i,j]$ 指示了 $T[1...i]$ 和 $M[1...j]$ 最长连续子串的长度，如果 $T[i] == M[j]$, 则 $c[i,j] =$ 以 $T[i-1]$ 和 $M[j-1]$ 结尾的最长连续子串的长度加一，否则就为 0。　　从上得，转移方程为

$$c[i,j] = \begin{cases} c[i-1,j-1]+1 & if\ X_i == Y_j \\ 0 & else \end{cases}$$

最终返回 c[1...m,1...m] 中的最大值。算法伪代码和之前类似, 稍作改动，见 **Algorithm 5**.

Problem 7.9

新建 $c[1...n]$,$c[i]$ 指示以 $s[1...i]$ 是否可以重建为由合法单词组成的序列, 新建 $pre[1...n]$ 用于重构解,$pre[i]$ 指示最后一个合法单词为 $s[pre[i]+1....i]$。　　易得有如下转移方程:

$$c[i] = \begin{cases} TRUE & i == 0 \\ c[i-1]\&\&dict(s[i])|\ |c[i-2]\&\&dict(s[i-1,i])|\ |.....|\ |c[0]\&\&dict(s[1,2...i]) & else \end{cases}$$

算法伪代码见 **Algorithm 6**, 重构解算法伪代码见 **Algorithm 7**.

Problem 7.10

1.

设原来的字符串为 $s[1...n]$, 新建 $c[1...n,1...n]$,$c[i][j]$ 指示 $s[i...j]$ 的最长回文子序列的长度：如果 $s[i] == s[j]$，则 $c[i][j] = c[i+1][j-1]+2$, 否则 $c[i][j] = max(c[i][j-1], c[i-1][j])$. 此外还有 base case 单个的字符

Algorithm 5 LCString_LEN

```
c[0...m, 0...m]
for i=1 to m do
    c[i, 0] = 0
end for
for j=0 to m do
    c[0, j] = 0
end for
for i=1 to m do
    for j=1 to m do
        if X_i == Y_j then
            c[i, j] = c[i − 1, j − 1] + 1
        else
            c[i, j] = 0
        end if
    end for
end for
return max of c[1..m][1...m]
```

Algorithm 6 DICT

```
c[0...n]
pre[1...n] = 0//全部初始化为 0
c[0] = TRUE
for i = 1 to n do
    bool judge = FALSE
    for j = i − 1 to 2 do
        if c[j]&&dict(s[j + 1...i]) == TRUE then
            judge = TRUE
            pre[i] = j
            Break
        end if
        c[i] = judge
    end for
end for
return c[n]
```

Algorithm 7 PRINT_DICT(n)

```
1: begin = pre[n]
2: if begin ≠ 0 then
3:    PRINT_DICT(begin)
4:    print s[begin + 1, ...n]
5: else
6:    print s[1...n]
7: end if
```

c[i][i]=1.

因此有转移方程为:

$$c[i][j] = \begin{cases} 1 & i == j \\ 2 & j - i == 2 \&\& s[i] == s[j] \\ 2 + c[i+1][j-1] & s[i] == s[j] \\ max(c[i][j-1], c[i-1][j]) & s[i] \neq s[j] \end{cases}$$

为了自底向上的扩展，考虑长度的递增，算法伪代码见 **Algorithm 8**.

此外亦可使用 LCS 的办法，求原串和颠倒后的原串的 LCS，就是满足条件的最长回文子序列。

Algorithm 8 LONGEST_HUIWEN

```
1: c[1...n][1...n]
2: for i = 1 to n do
3:    c[i][i] = 1
4: end for
5: for len = 2 to n do
6:    for i = 0 to n − len + 1 do
7:       j = i + len − 1
8:       if s[i] == s[j] then
9:          if len == 2 then
10:            c[i][j] = 2
11:         else
12:            2 + c[i + 1][j − 1]
13:         end if
14:      else
15:         max(c[i][j − 1], c[i − 1][j])
16:      end if
17:   end for
18: end for
19: return c[1][n]
```

2.

新建 $c[0...n]$, 其中 $c[0] = 0$,$c[i]$ 指示以 $s[1...i]$ 可以拆分的最少的回文数量。

易得有如下转移方程: **其中的整数集 K 为所有使得 A[k+1,k+2...i] 为回文的整数**, 即 $K = \{k \in Z | A[k+1, k+2...i]$是回文$\}$.

$$c[i] = \begin{cases} 0 & i == 0 \\ 1 + \min\limits_{k \in K} c[k] & else \end{cases}$$

算法伪代码见 **Algorithm 9**, 最终 $c[n]$ 的值为字符串可以拆分为的最少的回文数量。其中函数 **isHuiWen** 指示了某个串是否是回文的。

Algorithm 9 HUIWEN

```
1: c[0...n]
2: c[0] = 0
3: for i = 1 to n do
4:    temp = +∞
5:    for k = 0 to i − 1 do
6:       if isHuiWen(A[k+1,k+2,...i])==TRUE then
7:          temp = min(temp, 1 + c[k])
8:       end if
9:       c[i] = temp
10:   end for
11: end for
12: return c[n]
```

Problem 7.11

设字符串为 $s[1...n]$,$c[i][j]$ 指示了 $s[i...j]$ 最小分割代价。如果 i 到 j 中没有分割点了，那么代价就为 0；如果有分割点，最小分割代价为选择其中的某个分割点使得产生最小的左右子串最小分割代价和加上字符串本身的长度。因此，易得转移方程为: 其中 M 为分割点

$$c[i][j] = \begin{cases} 0 & \text{if none M in } s[i...j] \\ j - i + 1 + \min\limits_{M}(c[i][M] + c[M+1][j]) & else \end{cases}$$

算法伪代码见 **Algorithm 10**. 时间复杂度为 $O(mn^2)$

Algorithm 10 PARTING_STRING

```
1: c[1...n, 1...n]
2: for len = 2 to n do
3:   for i = 0 to n − len + 1 do
4:     j = i + len − 1
5:     if no M in s[i...j] then
6:       c[i][j] = 0
7:     else
8:       temp = +∞
9:       for each M_i in s[i...j] do
10:        temp = min(temp, c[i][M_i] + c[M_i + 1][j])
11:      end for
12:      c[i][j] = temp + len
13:    end if
14:  end for
15: end for
16: return c[1][n]
```

Problem 7.13

设 $T(root)$ 指示以 $root$ 为根的树的最小顶点覆盖的大小,那么如果选了 $root$,$T(root) = 1+\sum_{w\in Children(root)} T(w)$; 如果没选 $root$,$root$ 的孩子们就都要选,$T(root) = |Children(root)| + \sum_{z\in Grandchildren(root)} T(z)$.$T(root)$ 应为上面两者中较小的那个.

由上述，可以得到 $T(root)$ 的转移方程为：

$$T(root) = \begin{cases} 0 & root\text{为叶节点} \\ min(1 + \sum_{w\in Children(root)} T(w), |Children(root)| + \sum_{z\in Grandchildren(root)} T(z)) & else \end{cases}$$

据此可以写出一个动态规划算法，自树叶结点向上计算，利用一个 $O(|V|)$ 的空间存储最小顶点覆盖大小。

Problem 7.15

首先假设两两旅馆之间的距离均小于等于 200 公理，即不存在一天走完都找不到旅馆休息的问题。 新建 $c[1...n]$,$pre[1...n]$ 其中 $c[1] = (200 - a_1)^2$,$c[i]$ 指示到 a_i 最小总惩罚,$pre[i]$ 指示这个最小总惩罚停留序列的前一站为 $a_{pre[i]}$, 用以重构解.

易得有如下转移方程:

$$c[i] = \begin{cases} (200 - a_1)^2 & i == 1 \\ \min\limits_{\text{all } a_S \text{ that } a_i - a_S < 200} ((200 - a_i + a_S)^2 + c[S]) & else \end{cases}$$

算法见 **Algorithm 11**, 最优位置序列的重构解程序见 **Algorithm 12**

Algorithm 11 TRAVERING

$c[1...n] = +\infty$//初始化为正无穷
$c[1] = (200 - a_1)^2$
$pre[1...n] = 0$//全部初始化为 0
for $i = 2$ **to** n **do**
  for all a_S that $a_i - a_S < 200$ **do**
    if $c[i] > c[S] + (200 - a_i + a_S)^2$ **then**
      $c[i] = c[S] + (200 - a_i + a_S)^2$
      $pre[i] = S$
    end if
  end for
end for
return $c[n]$

Algorithm 12 PRINT_TRAVERING(n)

if $n \neq 0$ **then**
  PRINT_TRAVERING($pre[n]$)
  print n
end if