算法设计与分析

作业（五）

姓　　名	熊恪峥
学　　号	22920202204622
日　　期	2022年3月23日
学　　院	信息学院
课程名称	算法设计与分析

作业（五）

目录

1 题6.4

对于维度序列为$\{5, 10, 3, 12, 5, 50, 6\}$的矩阵链，可以按递推方程(1)填表 1和表 1

$$m_{i,j} = \begin{cases} 0 & i = j \\ \min_{i \le k < j}\{m_{i,k} + m_{k,j} + p_{i-1}p_jp_k\} & i \ne j \end{cases} \tag{1}$$

表格 1: $m_{i,j}$内容

0	150	330	405	1655	2010
inf	0	360	330	2430	1950
inf	inf	0	180	930	1770
inf	inf	inf	0	3000	1860
inf	inf	inf	inf	0	1500
inf	inf	inf	inf	inf	0

表格 2: $s_{i,j}$内容

N	1	2	2	4	2
N	N	2	2	2	2
N	N	N	3	4	4
N	N	N	N	4	4
N	N	N	N	N	5
N	N	N	N	N	N

并且根据$s_{i,j}$中的内容可以得到括号加法为

$$((A_1A_2)((A_3A_4)(A_5A_6)))$$

代码实现见附录：代码实现中的代码1

2 题6.6

$$P(n) = \begin{cases} \Theta(1) & n = 1 \\ \sum_{k=1}^{n-1} P(k)P(n-k) & n \ge 2 \end{cases} \tag{2}$$

要证$P(n) = \Omega(2^n)$只需证$P(n) \ge c \cdot 2^n$

证： $P(n) \ge c \cdot 2^n$

当 $n = 1$， $P(1) = 1 > c \cdot 2$，当$c \le 1$

若 $\forall k < n, P(k) \ge c \cdot 2^k$成立，则

$$\begin{aligned} P(n) &= \sum_{k=1}^{n-1} P(k)P(n-k) \\ &\ge c \cdot 2^1 \times c \cdot 2^{n-1} + c \cdot 2^2 \times c \cdot 2^{n-2} + \cdots + c \cdot 2^{n-1} \times c \cdot 2^1 \\ &= c^2 \cdot (n-1) \cdot (2^n) \\ &\ge c \cdot 2^n \end{aligned}$$

当

$$c \geq \frac{1}{n-1}$$

3 题6.8

由状态转移方程(3)

$$\begin{aligned} f &= \min\{f_1[n]+x_1, f_2[n]+x_2\} \\ f_1[n] &= \begin{cases} e_1+a_{1,1} & j=1 \\ \min\{f_1[j-1], f_2[j-1]+t_{2,j-1}\}+a_{1,j} & j\geq 2 \end{cases} \\ f_2[n] &= \begin{cases} e_2+a_{2,1} & j=1 \\ \min\{f_2[j-1], f_1[j-1]+t_{1,j-1}\}+a_{2,j} & j\geq 2 \end{cases} \end{aligned} \tag{3}$$

可知$f_1[n]$的计算引用了$f_1[n-1]$和$f_2[n-1]$，$f_1[n-1]$的计算引用了$f_1[n-2]$和$f_2[n-2]$，...以此类推，设f_1, f_2中元素引用次数$r_1[n]=1, r_2[n]=1$，有

$$r_1[n-1] = r_1[n]+r_2[n], r_2[n-1] = r_1[n]+r_2[n]$$

易知

$$\begin{aligned} r_1[n] &= O(2^n) \\ r_2[n] &= O(2^n) \end{aligned}$$

则子问题都被多次计算，因此有重叠子问题的性质。

4 题6.10

根据状态转移方程可以实现算法 1

算法 1 LCS问题

Input: 两个字符及其长度，以及备忘录的缓冲区

```
procedure LCS(X,Y,m,n,memo)
    if m == 0 or n == 0 then
        return memo[i][j] = 0
    if memo[i][j] ≠ −1 then
        return memo[i][j]
    if X[m] == Y[n] then
        return memo[i][j] = LCS(X,Y,m − 1,n − 1,memo)+1
    else
        return memo[i][j] = max{ LCS(X,Y,m − 1,n,memo),LCS(X,Y,m,n − 1,memo)}
```

5 题6.11

X_m和Y_n是A开始的字符串，不妨设存在长度≥ 2的最长公共子序列，则

$$X_m = A \dots s \dots$$
$$Y_n = A \dots s \dots$$

其中s可以是任意字符，表示起始的A之后任何一个重合的字符。

若存在一个最长公共子序列lcs_k，lcs_k不以A开头，则不妨设lcs_k以s开头。

则可以构造lcs'_k，令

$$lcs'_k = A + lcs_k = As \dots$$

由于X_m和Y_n是A开始的字符串，按照定义显然lcs'_k是X_m和Y_n的一个最长公共子序列。但

$$len(lcs'_k) = len(lcs_k) + 1 > len(lcs_k)$$

与最长公共子序列的最优性矛盾，因此假设不成立，一切最长公共子序列都以A开头

6 题6.16

由(4)计算w时每一次第二层循环需要1次或2次加法，因此相当于给时间复杂度贡献了常数。而

$$w_{i,j} = \begin{cases} q_{i-1} & j = i-1 \\ w_{i,j-1} + p_i + q_j & 1 \leq i \leq j \leq n \end{cases} \tag{4}$$

使用(??)计算，则需要$j - i + 1 + j - i + 2 = 2j - 2i + 3$次加法。

$$w = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l \tag{5}$$

则相当于给时间复杂度贡献了一个乘积项n，时间复杂度会变成$\mathcal{O}(n^4)$

7 题6.19

首先注意到删除操作不是必要的。例如删除b使得$abcd$与acd相同，等效于插入b使得$abcd$与acd相同。即在一个串中删除一个字符c等于在另一个串中对应位置插入一个c。则有如下三种不同的操作：

- 在串A中插入
- 在串B中插入
- 替换一个字符。由于在A中和B中替换是等效的，不妨设在A中。

那么为了完成A中前i个字符和B中前j个字符的编辑，可以：

1. 在完成了A中前i个字符和B中前$j-1$个字符的编辑之后，在A中插入$B[j]$
2. 在完成了A中前$i-1$个字符和B中前j个字符的编辑之后，在A中插入$A[i]$
3. 在完成了A中前$i-1$个字符和B中前$j-1$个字符的编辑之后，将$A[i]$和$B[j]$改写成一样的字符。

特别地，对于情况3，当$A[i]=B[j]$时，可以不进行任何操作。对于空串ε，为了使它跟字符串S相同，可以插入$len(S)$次字符。

根据以上分析可以列出递推方程(6)。

$$f_{i,j}=\begin{cases} i & j=0 \\ j & i=0 \\ \min\{f_{i,j-1},f_{i-1,j},f_{i-1,j-1}\}+1 & A[i-1]\neq B[j-1] \\ \min\{f_{i,j-1}+1,f_{i-1,j}+1,f_{i-1,j-1}\} & A[i-1]=B[j-1] \end{cases} \tag{6}$$

据此实现算法2。

算法 2 编辑距离问题问题

Input: 两个字符

procedure DISTANCE(X,Y,m,n,$memo$)
    for $i=0\to|A|$ **do**
        $f_{i,0}\leftarrow i$
    for $i=0\to|B|$ **do**
        $f_{0,j}\leftarrow j$
    for $i=1\to|A|$ **do**
        for $i=1\to|B|$ **do**
            if $A[i-1]==B[i-1]$ **then**
                $f_{i,j}\leftarrow\min\{f_{i,j-1}+1,f_{i-1,j}+1,f_{i-1,j-1}\}$
            else
                $f_{i,j}\leftarrow\min\{f_{i,j-1},f_{i-1,j},f_{i-1,j-1}\}+1$
    return $f_{|A|,|B|}$

8 实验一

要求给定两个程序，判断它们的相似性。显然，程序的相似性和代码字符串的相似性无关，而与实际执行逻辑的相似性有关。那么最准确的方式是进行DFA(Data Flow Analysis)和CFA(Control Flow Analysis)，对于相似的程序它们应当能相当准确地反映出相似度。这正是现代IDE对重复代码给出修改建议的方式。但这种方式实现相当复杂，本程序通过对问题进行简化有效地实现了**基于语义的**代码相似性判断。

8.1 两点假设

为了简化问题，首先进行以下两个假设

- 程序的抽象语法树(AST, Abstract Syntax Tree)和实际执行逻辑高度相关
- 抽象语法树中的语句节点和表达式节点是所有节点中和实际执行逻辑最相关的两类节点

根据这两点假设，通过从程序编译时的抽象语法树的语句(Statement)节点和表达式(Expression)节点序列中寻找最长上升子序列可以有效地衡量程序的逻辑相似性。定义逻辑相似度s，其中AST_j是程序代码j的抽象语法树的语句节点和表达式节点序列

$$s = \frac{|LCS_{AST_1,AST_2}|}{\max\{|AST_1|,|AST_2|\}}$$

A 附录：代码实现

代码 1: 矩阵连乘

```python
p = list([5, 10, 3, 12, 5, 50, 6])
N = 6

m = list([ list([0x7fffffff for i in range(0, N + 1)]) for j in range(0, N + 1)])
s = list([ list([0x7fffffff for i in range(0, N + 1)]) for j in range(0, N + 1)])


def pretty_print(i, j):
    if i == j:
        print('A{}'.format(i), end=' ')
    else:
        print("(", end=' ')
        pretty_print(i, s[i][j])
        pretty_print(s[i][j] + 1, j)
        print(")", end=' ')


for i in range(1, N + 1):
    m[i][i] = 0

for i in range(N, 0, -1):
    for j in range(i, N + 1):
        if i == j:
            m[i][j] = 0
        else:
            for k in range(i, j):
                val = m[i][k] + m[k + 1][j] + p[i - 1] * p[j] * p[k]
                if m[i][j] > val:
                    m[i][j] = val
                    s[i][j] = k

for i in range(1, N + 1):
    for j in range(1, N + 1):
        print("inf" if m[i][j] == 0x7fffffff else m[i][j], end=' ' if j != N else '\n')

for i in range(1, N + 1):
    for j in range(1, N + 1):
        print("None" if s[i][j] == 0x7fffffff else s[i][j], end=' ' if j != N else '\n')

pretty_print(1, N)
```