算法导论 课程设计

成绩	

题　　目：动态规划

学院班级：1613012

学　　号：16130120191

姓　　名：罗阳豪

主讲教师：覃桂敏

日　　期：2019 年 04 月 11 日

目录

题目一　矩阵链乘积

1.1 实验题目

实现矩阵链乘积的最优化算法，以下是一些例子

1. <3, 5, 2, 1, 10>

2. <2, 7, 3, 6, 10>

3. <10, 3, 15, 12, 7, 2>

4. <7, 2, 4, 15, 20, 5>

1.2 实验目的

了解动态规划的思想，了解经典动态规划问题的解法，认识使用动态规划解决问题的一般思路和方法

1.3 实验设计与分析

求解矩阵链乘积的本质就是找到一个最优的完全括号化方案，类似于分治的思想，我们可以找到这个问题的子问题。我们从矩阵链中间任何一处将矩阵链分开，将会有 n-1 种分法，最优的分法就是其中一种，但是我们必须再继续求解分开后的两个矩阵链的最优完全括号化方案才能知道当前的分法是否是最优的。所以对每个子矩阵链应用同样的方法，递归求解，直到待求矩阵链长度为一，这样就不再需要括号化了。

如果使用上面的方法递归求解，子问题将非常非常多，但是注意到，这些子问题是大量重复的，最终，划分到最小的子问题的数目是和 n 成正比的（实际上就是 n 个）。所以这个问题应该从下往上计算，先计算最小的子问题，然后再求解更大的子问题，直到最顶上，求解原问题。这就是动态规划的思想，具体算法实际上与课本描述完全一样，就不再赘述

1.4 实验环境

Python 3.7.0

1.4 项目测试

将题目中所给测试用例作为输入，分别得到结果如下

55 ((A0(A1A2))A3)

198 (((AOA1)A2)A3)

678 (A0(A1(A2(A3A4))))

990 (A0(((A1A2)A3)A4))

其中第一个数字为该括号化方案下求解矩阵链乘积的开销（标量乘法的次数），后面是该最优化的括号化方案

题目二 最长公共子序列

2.1 实验题目

实现最长公共子序列算法，以下是一些例子

1. X: xzyzzyx Y: zxyyzxz
2. X: MAEEEVAKLEKHLMLLRQEYVKLQKKLAETEKRCALLAAQANKESSSESFISRLLAIVAD

   Y: MAEEEVAKLEKHLMLLRQEYVKLQKKLAETEKRCTLLAAQANKENSNESFISRLLAIVAG

2.2 实验目的

了解动态规划的思想，了解经典动态规划问题的解法，认识使用动态规划解决问题的一般思路和方法。掌握经典的，最长公共子序列问题，的解决方法。

2.3 实验设计与分析

首先我们刻画最长公共子序列的特征，给出如下不加证明的定理

令 X = <x1, x2, ..., xm>和 Y = <y1, y2, ..., yn>为两个序列，Z = <z1, z2, ..., zk>为 X 和 Y 的任意 LCS，则

1. 如果 xm = yn，则 zk = xm = yn 且 Zk-1 是 Xm-1 和 Yn-1 的一个 LCS
2. 如果 xm != yn，且 zk != xm，则 Z 是 Xm-1 和 Y 的一个 LCS
3. 如果 xm != yn，且 zk != yn，则 Z 是 X 和 Yn-1 的一个 LCS

上述定理意味着，如果 xm = yn，我们应该求解 Xm-1 和 Yn-1 的一个 LCS。如果 xm != yn，我们就要求解两个子问题，Xm-1 和 Y 的 LCS 和 X 和 Yn-1 的 LCS，两个 LCS 较长者为 X 和 Y 的 LCS。

具体算法与课本描述完全一样，在此便不再做赘述

2.4 实验环境

Python 3.7.0

2.5 项目测试

将题目中所给测试用例作为输入，分别得到结果如下

xyzz

MAEEEVAKLEKHLMLLRQEYVKLQKKLAETEKRCLLAAQANKESESFISRLLAIVA

题目三 最长公共子串

3.1 实验题目

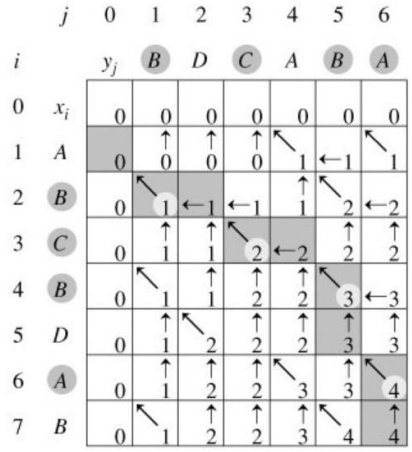
实现最长公共子串算法，用例同“最长公共子序列问题”

3.2 实验目的

了解动态规划的思想，了解经典动态规划问题的解法，认识使用动态规划解决问题的一般思路和方法。掌握经典的，最长公共子串问题，了解最长公共子串和最长公共子序列的相似性和区别。

3.3 实验设计与分析

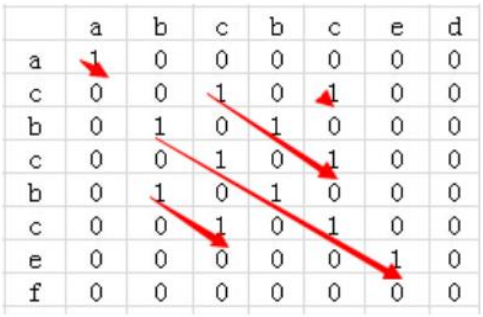
这个问题看起来非常类似“最长公共子序列问题”，实际上也的确类似。对于求解最长公共子序列，我们会通过这样一个矩阵来标识最长子序列和记录子问题的解



i	j	0	1	2	3	4	5	6
	y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑0	↑0	↑0	↖1	←1	↖1
2	B	0	↖1	←1	←1	↑1	↖2	←2
3	C	0	↑1	↑1	↖2	←2	↑2	↑2
4	B	0	↖1	↑1	↑2	↑2	↖3	←3
5	D	0	↑1	↖2	↑2	↑2	↑3	↑3
6	A	0	↑1	↑2	↑2	↖3	↑3	↖4
7	B	0	↖1	↑2	↑2	↑3	↖4	↑4

我们可以看到深色标注的结果，只能是从左下角，往上、左、左上三个方向移动，直到走到左上角，其中每一次往左上角移动等于 LCS 中一个字符，左、上移动表示跳过不相等的字符。实际上最长公共子串与此非常类似，只是不允许“跳过”不相等的字符，所以结果必然是与对角线平行的。

如果我们同样构造一个类似的矩阵



	a	b	c	b	c	e	d
a	1	0	0	0	0	0	0
c	0	0	1	0	1	0	0
b	0	1	0	1	0	0	0
c	0	0	1	0	1	0	0
b	0	1	0	1	0	0	0
c	0	0	1	0	1	0	0
e	0	0	0	0	0	1	0
f	0	0	0	0	0	0	0

这个矩阵标 1 的位置(i， j）就表明对应的 s1[i] == s2[j]。这些 1 沿对角线方向连起来可以画出上图中红色的线，这些线中最长的就表示最长公共子串。所以为了方便之后构造结果，对矩阵做一个小修改

	a	b	c	b	c	e	d
a	1	0	0	0	0	0	0
c	0	0	1	0	1	0	0
b	0	1	0	2	0	0	0
c	0	0	2	0	3	0	0
b	0	1	0	3	0	0	0
c	0	0	2	0	4	0	0
e	0	0	0	0	0	5	0
f	0	0	0	0	0	0	0

当遇到一样时，元素为左上角元素+1，这样只要找到数字最大的单元格，就找到了最长子串。

为了避免最后遍历该矩阵的开销，可以在求解这些单元格的同时维护一个变量用于记录当前的最大值及其在矩阵中的位置。

这样就有了一个时间复杂度为 O(mn)的算法

3.4 实验环境

Python 3.7.0

3.5 项目测试

将题目中所给测试用例作为输入，分别得到结果如下

xz

MAEEEVAKLEKHLMLLRQEYVKLQKKLAETEKRC

题目四 最大和

4.1 实验题目

最大和也叫最大连续子序列和，给定序列 X，其所有连续子序列中，元素之和最大的就是 X 的最大连续子序列和。以下是一个测试用例

1. (-2, 11, -4, 13, -5, -2)

4.2 实验目的

通过经典的“最大和”或“最大连续子序列”问题，加深对于最优化问题的认识，了解最优化问题的一般解题思路。

4.3 实验设计与分析

我们注意到，一个最大连续子序列一定不包含和小于等于 0 的前缀

所以只要从第一个元素往下计算和，当和小于等于 0 时，跳过前缀。在计算的同时记录最大和纪录和产生最大纪录的位置。最后输出结果

这样就有了一个时间复杂度为 O(n)的算法，其伪代码如下

```
MAX_SUM(d):
    res = d[0]
    max_i = 0
    max_j = 1
    i = 0
    j = 0
    max_count = 0
    for item = each in d:
        max_count += item
        j += 1
        if max_count > res:
            res = max_count
            max_i = i
            max_j = j
        if max_count <= 0:
            max_count = 0
            i = j
    return res, max_i, max_j
```

4.4 实验环境

Python 3.7.0

4.5 项目测试

将题目中所给测试用例作为输入，分别得到结果如下

20 1 4

第一个数是最大和，第二和第三个数是最大连续子序列的位置（左闭右开区间）

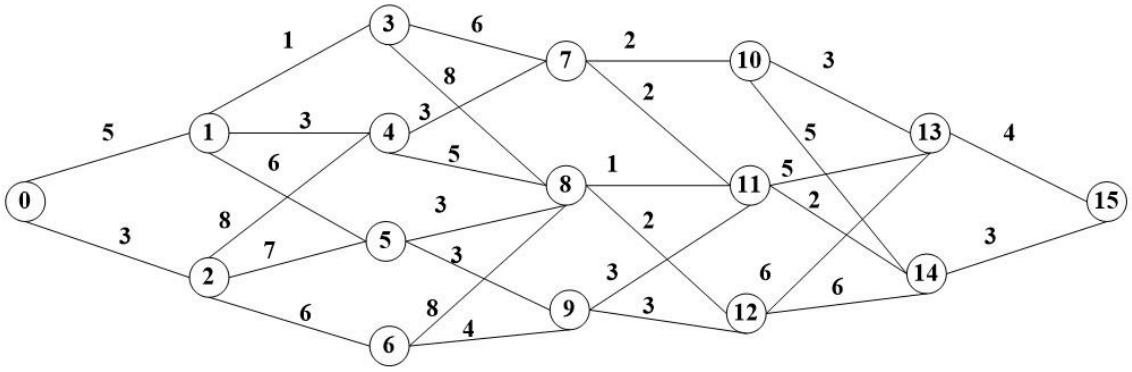
题目 5 多级图的最短路径

5.1 实验题目

多级图中的最短路径。找出下图中从节点 0 到节点 15 的最短路径。

多级图是满足以下条件的图

1. G = (V, E)的 V 可以被分为 K >= 2 个不相交的部分，并且对于 E 中的任意一个边 (a, b)，如果 a 在 Vi 中，则 b 在 Vi+1 中
2. |V1| = |Vk| = 1



5.2 实验目的

了解多级图与一般有向图的区别，了解在具有不同特征的图上求解最短路径的不同。

了解动态规划法求解最短路问题的方法

5.3 实验设计与分析

多级图与一般的图不同，对于一般的图，最短路算法一般操作就是按照一定的顺序，多次，对所有路径做松弛操作，相关的算法当然也能够应用于多级图，但是一般这样开销都会比较大。由于多级图的特殊性质，多级图的最短路求解会稍微简单一点。

多级图可以被分为大于 2 个级，每条路都是联系前后相邻两个级的，第一个级和最后一个级的节点个数都是 1。

所以我们可以考虑按级求解。首先是从第一级，逐一查看以第一级节点为起点的边，获得从起点到第二级各节点的最短路径和开销。

然后查看第二级到第三级各边，获得从起点到第三级各节点的最短路及开销。逐一求解这些子问题，最终得到从起点到终点的最短路。

这样，该算法时间复杂度为 O(n)（n 为边数）

5.4 实验环境

Python 3.7.0

5.5 项目测试

对于上图求解节点 0 到节点 15 的最短路径和最短路径的开销为

([0, 1, 4, 7, 11, 14, 15], 18)