**实验报告**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **课程名称** | 《算法分析与设计》 | **实验日期** | 2021 年5月 10日 至2021 年 5 月 15日 | | | | |
| **学生姓名** | 王海轩 | **所在班级** | 计算机195 | | | **学号** | 2019212212194 |
| **实验名称** | LCS算法和背包算法，特别要求举例时采用不同于讲义的数据进行推导 | | | | | | |
| **实验地点** | 勤园13-208 | | | **同组人员** |  | | |

# 问题

LCS算法和背包算法，特别要求举例时采用不同于讲义的数据进行推导

# 解析

LCS算法：

（1）最长公共子序列的结构有如下表示：

设序列X=<x1, x2, …, xm>和Y=<y1, y2, …, yn>的一个最长公共子序列Z=<z1, z2, …, zk>，则：

1.若xm=yn，则zk=xm=yn且Zk-1是Xm-1和Yn-1的最长公共子序列；

2.若xm≠yn且zk≠xm ，则Z是Xm-1和Y的最长公共子序列；

3.若xm≠yn且zk≠yn ，则Z是X和Yn-1的最长公共子序列。

其中Xm-1=<x1, x2, …, xm-1>，Yn-1=<y1, y2, …, yn-1>，Zk-1=<z1, z2, …, zk-1>。

（2）子问题的递归结构

由最长公共子序列问题的最优子结构性质可知，要找出X=<x1, x2, …, xm>和Y=<y1, y2, …, yn>的最长公共子序列，可按以下方式递归地进行：当xm=yn时，找出Xm-1和Yn-1的最长公共子序列，然后在其尾部加上xm(=yn)即可得X和Y的一个最长公共子序列。当xm≠yn时，必须解两个子问题，即找出Xm-1和Y的一个最长公共子序列及X和Yn-1的一个最长公共子序列。这两个公共子序列中较长者即为X和Y的一个最长公共子序列。

由此递归结构容易看到最长公共子序列问题具有子问题重叠性质。例如，在计算X和Y的最长公共子序列时，可能要计算出X和Yn-1及Xm-1和Y的最长公共子序列。而这两个子问题都包含一个公共子问题，即计算Xm-1和Yn-1的最长公共子序列。

与矩阵连乘积最优计算次序问题类似，我们来建立子问题的最优值的递归关系。用c[i,j]记录序列Xi和Yj的最长公共子序列的长度。其中Xi=<x1, x2, …, xi>，Yj=<y1, y2, …, yj>。当i=0或j=0时，空序列是Xi和Yj的最长公共子序列，故c[i,j]=0。

（3）计算最优值

直接利用上节节末的递归式，我们将很容易就能写出一个计算c[i,j]的递归算法，但其计算时间是随输入长度指数增长的。由于在所考虑的子问题空间中，总共只有θ(m\*n)个不同的子问题，因此，用动态规划算法自底向上地计算最优值能提高算法的效率。

计算最长公共子序列长度的动态规划算法LCS\_LENGTH(X,Y)以序列X=<x1, x2, …, xm>和Y=<y1, y2, …, yn>作为输入。输出两个数组c[0…m ,0…n]和b[1…m ,1…n]。其中c[i,j]存储Xi与Yj的最长公共子序列的长度，b[i,j]记录指示c[i,j]的值是由哪一个子问题的解达到的，这在构造最长公共子序列时要用到。最后，X和Y的最长公共子序列的长度记录于c[m,n]中。

背包问题（01背包）算法：

有N件物品和一个容量为V的背包，第i件物品消耗的容量为Ci，价值为Wi，求解放入哪些物品可 以使得背包中总价值最大。

建立模型，即求max(V1X1+V2X2+…+VnXn)；寻找约束条件，W1X1+W2X2+…+WnXn<capacity；寻 找递推关系式，面对当前商品有两种可能性：包的容量比该商品体积小，装不下，此时的价值与前i-1 个的价值是一样的，即V(i,j)=V(i-1,j)；还有足够的容量可以装该商品，但装了也不一定达到当前最优 价值，所以在装与不装之间选择最优的一个，即V(i,j)=max｛V(i-1,j)，V(i-1,j-w(i))+v(i)｝。其中V(i-1,j) 表示不装，V(i-1,j-w(i))+v(i) 表示装了第i个商品，背包容量减少w(i)，但价值增加了v(i)；由此可以 得出递推关系式：

j<w(i) V(i,j)=V(i-1,j)

j>=w(i) V(i,j)=max｛V(i-1,j)，V(i-1,j-w(i))+v(i)｝

# 设计

LCS:

for(i=1;i<=lena;++i)

{

for(j=1;j<=lenb;++j)

{

if(a[i-1]==b[j-1])

dp[i][j]=dp[i-1][j-1]+1;

else

dp[i][j]=max(dp[i-1][j],dp[i][j-1]);

}

}

背包：

for(int i=0;i<n;i++){ //滚动数组优化

for(int j=m;j>=0;j--){

if(j>=w[i]){

dp[j]=max(dp[j],(dp[j-w[i]]+v[i]));

}

}

}

# 分析

Lcs：O （ n 2 ） O（n^2）O（n 2）

背包: O ( n ∗ m ) O(n\*m)O(n∗m)

# **5.源码**

[github源码地址]

https://github.com/whx116/lab9.git