

**《算法分析与技术》实验报告**

学生姓名： 谭哲文（8202191123）

指导老师： 李 敏

2023年06月

**实验四 动态规划**

**一、实验目的：**

理解动态规划的基本思想，理解动态规划算法的两个基本要素最优子结构性质和子问题的重叠性质。熟练掌握典型的动态规划问题。掌握动态规划思想分析问题的一般方法，对较简单的问题能正确分析，设计出动态规划算法，并能快速编程实现。

**实验 4.1 最长公共子序列**

**（1）实验内容**

一个给定序列的子序列是在该序列中删去若干元素后得到的序列。确切地说，若给定序列

X=<x1, x2,…, xm>，则另一序列 Z=<z1, z2,…, zk>是 X 的子序列是指存在一个严格递增

的下标序列 <i1, i2,…, ik>，使得对于所有 j=1,2,…,k 有

易证最长公共子序列问题也有最优子结构性质

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：

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

ii. 若 xm≠yn 且 zk≠xm ，则 Z 是 Xm-1 和 Y 的最长公共子序列；

iii.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. 。所以最长公共子序列问题具有最优子结构性质。

**（2）问题描述分析与算法设计思想**

蛮力法求解最长公共子序列：

需要遍历出所有的可能，时间复杂度是O(n³)，太慢了

动态规划求解最长公共子序列：

分析规律：

设X=<x1,x2,x3,x4...,xm>，Y=<y1,y2,y3,y4...,yn>为两个序列，Z=<z1,z2,z3,z4...,zk>是他们的任意公共子序列

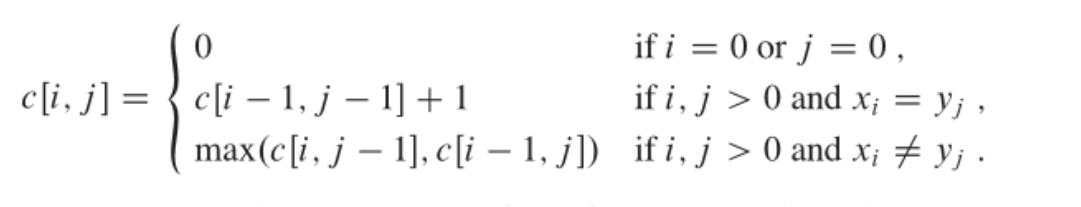
经过分析，我们可以知道：

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

所以如果用一个二维数组c表示字符串X和Y中对应的前i，前j个字符的LCS的长度话，可以得到以下公式：



文字意思就是：

设

p1表示X的前 i-1 个字符和Y的前 j 个字符的LCS的长度

p2表示X的前 i 个字符和Y的前 j-1 个字符的LCS的长度

p表示X的前 i-1 个字符和Y的前 j-1 个字符的LCS的长度

p0表示X的前 i 个字符和Y的前 j 个字符的LCS的长度

如果X的第 i 个字符和Y的第 j 个字符相等，则p0 = p + 1

如果X的第 i 个字符和Y的第 j 个字符不相等，则p0 = max(p1,p2)

做法：

因此，我们只需要从c[0][0]开始填表，填到c[m-1][n-1]，所得到的c[m-1][n-1]就是LCS的长度

但是，我们怎么得到LCS本身而非LCS的长度呢？

也是用一个二维数组b来表示：

在对应字符相等的时候，用↖标记

在p1 >= p2的时候，用↑标记

在p1 < p2的时候，用←标记

若想得到LCS，则再遍历一次b数组就好了，从最后一个位置开始往前遍历：

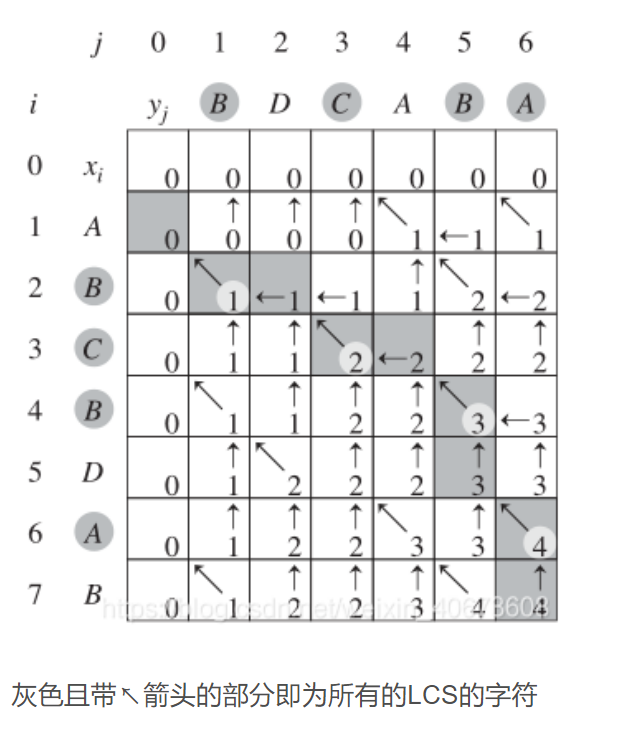
如果箭头是↖，则代表这个字符是LCS的一员，存下来后 i-- , j--

如果箭头是←，则代表这个字符不是LCS的一员，j--

如果箭头是↑ ，也代表这个字符不是LCS的一员，i--

如此直到i = 0或者j = 0时停止，最后存下来的字符就是所有的LCS字符

比如说求ABCBDAB和BDCABA的LCS:



**（3）代码实现**

#include <iostream>

#include <string>

#include <stack>

using namespace std;

void LCS(string s1, string s2)

{

    int m = s1.length() + 1;

    int n = s2.length() + 1;

    int\*\* c;

    int\*\* b;

    c = new int\* [m];

    b = new int\* [m];

    for (int i = 0; i < m; i++)

    {

        c[i] = new int[n];

        b[i] = new int[n];

        for (int j = 0; j < n; j++)

            b[i][j] = 0;

    }

    for (int i = 0; i < m; i++)

        c[i][0] = 0;

    for (int i = 0; i < n; i++)

        c[0][i] = 0;

    for (int i = 0; i < m - 1; i++)

    {

        for (int j = 0; j < n - 1; j++)

        {

            if (s1[i] == s2[j])

            {

                c[i + 1][j + 1] = c[i][j] + 1;

                b[i + 1][j + 1] = 1;          //1表示箭头为  左上

            }

            else if (c[i][j + 1] >= c[i + 1][j])

            {

                c[i + 1][j + 1] = c[i][j + 1];

                b[i + 1][j + 1] = 2;          //2表示箭头向  上

            }

            else

            {

                c[i + 1][j + 1] = c[i + 1][j];

                b[i + 1][j + 1] = 3;          //3表示箭头向  左

            }

        }

    }

    for (int i = 0; i < m; i++)                //输出c数组

    {

        for (int j = 0; j < n; j++)

        {

            cout << c[i][j] << ' ';

        }

        cout << endl;

    }

    stack<char> same;                   //存LCS字符

    stack<int> same1, same2;             //存LCS字符在字符串1和字符串2中对应的下标，方便显示出来

    for (int i = m - 1, j = n - 1; i >= 0 && j >= 0; )

    {

        if (b[i][j] == 1)

        {

            i--;

            j--;

            same.push(s1[i]);

            same1.push(i);

            same2.push(j);

        }

        else if (b[i][j] == 2)

            i--;

        else

            j--;

    }

    cout << s1 << endl;                     //输出字符串1

    for (int i = 0; i < m && !same1.empty(); i++)      //输出字符串1的标记

    {

        if (i == same1.top())

        {

            cout << 1;

            same1.pop();

        }

        else

            cout << ' ';

    }

    cout << endl << s2 << endl;                //输出字符串2

    for (int i = 0; i < n && !same2.empty(); i++)      //输出字符串2的标记

    {

        if (i == same2.top())

        {

            cout << 1;

            same2.pop();

        }

        else

            cout << ' ';

    }

    cout << endl << "最长公共子序列为：";

    while (!same.empty())

    {

        cout << same.top();

        same.pop();

    }

    cout << endl << "长度为：" << c[m - 1][n - 1] << endl;

    for (int i = 0; i < m; i++)

    {

        delete[] c[i];

        delete[] b[i];

    }

    delete[]c;

    delete[]b;

}

int main()

{

    string s1 = "ABCPDSFJGODIHJOFDIUSHGD";

    string s2 = "OSDIHGKODGHBLKSJBHKAGHI";

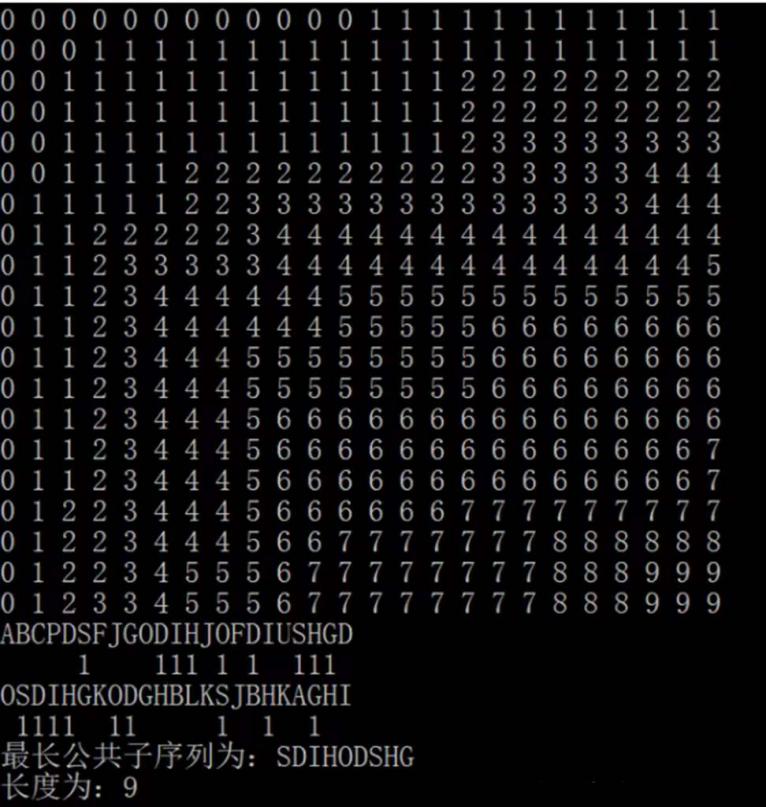
    LCS(s1, s2);

    system("pause");

    return 0;

}

运行结果：



1. **算法及问题分析**

### ****计算最优值：****

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。

### ****时间复杂度：****

由于只需要填一个m行n列的二维数组，其中m代表第一个字符串长度，n代表第二个字符串长度

所以时间复杂度为O（m\*n）

**实验 4.9 皇宫看守**

**（1）实验内容**

太平王世子事件后，陆小凤成了皇上特聘的御前一品侍卫。皇宫以午门为起点，直到后宫嫔妃们的寝宫，呈一棵树的形状；某些宫殿间可以互相望见。大内保卫森严，三步一岗，五步一哨，每个宫殿都要有人全天候看守，在不同的宫殿安排看守所需的费用不同。可是陆小凤手上的经费不足，无论如何也没法在每个宫殿都安置留守侍卫。

请你编程计算帮助陆小凤布置侍卫，在看守全部宫殿的前提下，使得花费的经费最少。

输入数据：输入数据由文件名为intput.txt的文本文件提供。输入文件中数据表示一棵树，描述如下：

第1行 n，表示树中结点的数目。

第2行至第n+1行，每行描述每个宫殿结点信息，依次为：该宫殿结点标号i（0<i<=n），在该宫殿安置侍卫所需的经费k，该边的儿子数m，接下来m个数，分别是这个节点的m个儿子的标号r1，r2，...，rm。

对于一个n（0 < n <= 1500）个结点的树，结点标号在1到n之间，且标号不重复。

输出数据：输出到output.txt文件中。输出文件仅包含一个数，为所求的最少的经费。

**（2）问题描述分析与算法设计思想**

首先这道题的题意是：给定一棵树，要在一些节点上放置守卫，每个守卫可以看护当前节点以及与此节点连通的节点，在不同节点放置守卫的代价不同，如何选取节点使代价最小，这是个典型的树形DP问题，显然每个节点有放置守卫和不放置守卫两种，但是从计算的过程看，不放置守卫的状态由两种，一种是有其父节点上的守卫看护，一种是由其子节点的守卫看护，因此可将每个节点的看护情况分为三种：

1.该节点由父节点处放置的守卫看护

2.该节点由子节点处放置的守护看护

3.该节点由在该节点放置的守卫看护

下面考虑状态转移的过程，建立数组f[i][3]，其中

1. f[i][0]表示第i个节点由父节点处放置的守卫看护下的最小代价
2. f[i][1]表示第i个节点由子节点处放置的守卫看护下的最小代价
3. f[i][2]表示第i个节点由在该节点放置的守卫看护下的最小代价

那么可以写出转移关系：

1. f[i][0] += min(f[j][1], f[j][2]);
2. f[i][1] = min(f[i][1], sum - min(f[j][1], f[j][2]) + f[j][2]);
3. f[i][2] += min(min(f[j][0], f[j][1]), f[j][2]);

其中j为i的子节点，sum为所有子节点j的min(f[j][1], f[j][2])的和，1和3的意义很明显，2的意义代表，如果第i个节点由子节点守卫，那么所有子节点都不能由父节点守卫，并且每个子节点都得到了守卫，且至少有一个子节点处放置了守卫

**（3）代码实现**

#include <iostream>

#include <cstring>

#include <algorithm>

#include <cmath>

using namespace std;

const int N = 1510;

int n;

int h[N], e[N], ne[N], idx, w[N];

int f[N][3];

bool st[N];

void add(int a, int b)

{

    e[idx] = b, ne[idx] = h[a], h[a] = idx++;

}

void dfs(int u)

{

    f[u][2] = w[u];

    int sum = 0;

    for (int i = h[u]; ~i; i = ne[i])

    {

        int j = e[i];

        dfs(j);

        f[u][0] += min(f[j][1], f[j][2]);

        f[u][2] += min(min(f[j][0], f[j][1]), f[j][2]);

        sum += min(f[j][1], f[j][2]);

    }

    f[u][1] = 1e9;

    for (int i = h[u]; ~i; i = ne[i])

    {

        int j = e[i];

        f[u][1] = min(f[u][1], sum - min(f[j][1], f[j][2]) + f[j][2]);

    }

}

int main()

{

    cout << "Sample Input:" << endl;

    cin >> n;

    memset(h, -1, sizeof h);

    for (int i = 1; i <= n; i++)

    {

        int id, cost, cnt;

        cin >> id >> cost >> cnt;

        w[id] = cost;

        while (cnt--)

        {

            int ver;

            cin >> ver;

            add(id, ver);

            st[ver] = true;

        }

    }

    int root = 1;

    while (st[root])

        root++;

    dfs(root);

    cout << "Sample Out:" << endl;

    cout << min(f[root][1], f[root][2]) << endl;

    system("pause");

    return 0;

}

运行结果：

输入样例：

6

1 30 3 2 3 4

2 16 2 5 6

3 5 0

4 4 0

5 11 0

6 5 0

