浙江工业大学

算法分析与设计实验报告

(2021 级)

实验题目	实验 3
学生姓名	温家伟
学生学号	202103151422
专业班级	大数据分析 2101
所在学院	理学院
提交日期	2023-4-17

实验目的

掌握动态规划法的解题步骤

第三章 动态规划

实验内容

1.0-1背包

给定n(n<=100)种物品和一个背包。物品i的重量是wi(wi<=100)，价值为vi(vi<=100)，背包的容量为C(C<=1000)。
应如何选择装入背包中的物品，使得装入背包中物品的总价值最大? 在选择装入背包的物品时，对每种物品i只有两个选择：装入或不装入。不能将物品i装入多次，也不能只装入部分物品i。

输入格式:

共有n+1行输入：
第一行为n值和c值，表示n件物品和背包容量c;
接下来的n行，每行有两个数据，分别表示第i(1≤i≤n)件物品的重量和价值。

输出格式:

输出装入背包中物品的最大总价值。

输入样例:

在这里给出一组输入。例如：

```
5 10
2 6
2 3
6 5
5 4
4 6
```

输出样例:

在这里给出相应的输出。例如：

```
15
```

2.矩阵连乘问题

给定n个矩阵{A1,A2,…,An}（n<=40），其中Ai与Ai+1是可乘的，i=1，2…，n。第i个矩阵的维数用p_{i-1}，p_i来表示。如何确

定计算矩阵连乘积的计算次序，使得依此次序计算矩阵连乘积需要的数乘次数最少。例如，给定三个连乘矩阵{A1，A2，A3}的维数
数组p为：10,100,5,50，即分别是10 ×100，100×5和5×50，采用（A1A2）A3，乘法次数为10×100×5+10×5×50=7500次，而采用A1（A2A3），乘法次数为100×5×50+10×100×50=75000次乘法，显然，最好的次序是（A1A2)A3，乘法次数为7500次。

输入格式:

输入有两行。第一行一个n表示矩阵的个数；第二行有n+1个数，分别为p_0，p_1…p_n。

输出格式:

一个数，表示最少的乘法次数。

输入样例:

```
6
30 35 15 5 10 20 25
```

输出样例:

```
15125
```

3.最长公共子序列长度

求两个字符串的最长公共子序列长度。

输入格式:

输入长度≤100的两个字符串。

输出格式:

输出两个字符串的最长公共子序列长度。

输入样例1:

```
ABCBDAB
BDCABA
```

输出样例1:

```
4
```

输入样例2:

```
ABACDEF
PGHIK
```

输出样例2:

```
0
```

4.最优二叉搜索树

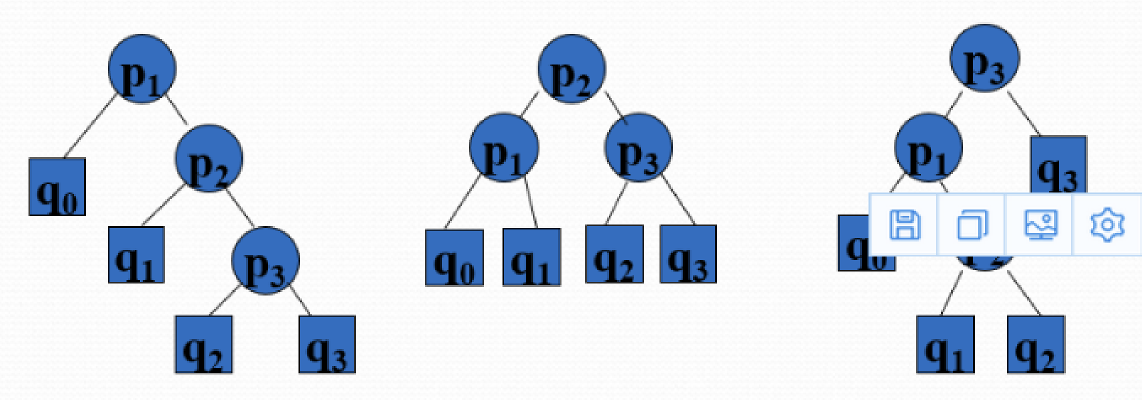
设 $S=\{x_1,x_2,\dots,x_n\}$是有序集，且$x_1<x_2<\dots<x_n$，表示有序集S的二叉搜索树利用二叉树的结点来存储有序集合中的元素。在该二叉搜索树中搜索一个元素x，结果有两种情形：

1. 在二叉搜索树的内结点中找到$x=x_i$(即找到了实结点)，设这种情况的概率为p_i
2. 在二叉搜索树的叶节点中确定$x\in(x_i,x_{i+1})$(即找到了虚结点)，设这种情况的概率为q_i

设根节点的深度为0，存储元素x_i的结点深度为c_i，叶子节点(x_j,x_{j+1})的结点深度为d_j，在该二叉搜索树中进行一次搜索所需的平均比较次数为m，那么有公式：

$$m=\sum_{i=1}^{n}p_i(1+c_i)+\sum_{j=0}^{n}q_jd_j$$

m又称为二叉搜索树 T 的平均路长，本题的要求是对于有序集 S 及其存取概率分布$(q_0,p_1,q_1,\dots,p_n,q_n)$，在所有表示有序集 S 的二叉搜索树中找出一颗具有最小平均路长的二叉搜索树。



n=3的有序集对应的二叉搜索树的三种形态

输入格式:

第一行给出有序集中的元素数量 n (2<n<10)

第二行给出内结点（实结点）的存取概率分布 $p_1,\dots,p_n$

第三行给出叶节点（虚结点）的存取概率分布 $q_0,q_1,\dots,q_n$

输出格式:

第一行输出最小 m，保留两位小数。

第二行先序遍历输出二叉树，对于实结点输出其编号，对于虚结点输出`.`，结点之间用空格分隔，若有多个满足要求的二叉树，则输出根节点编号最小的。

输入样例:

```
3
0.50 0.1 0.05
0.15 0.1 0.05 0.05
```

输出样例:

```
1.50
1 . 2 . 3 . .
```

代码长度限制	16 KB

实验结果及相应代码

1.0-1背包

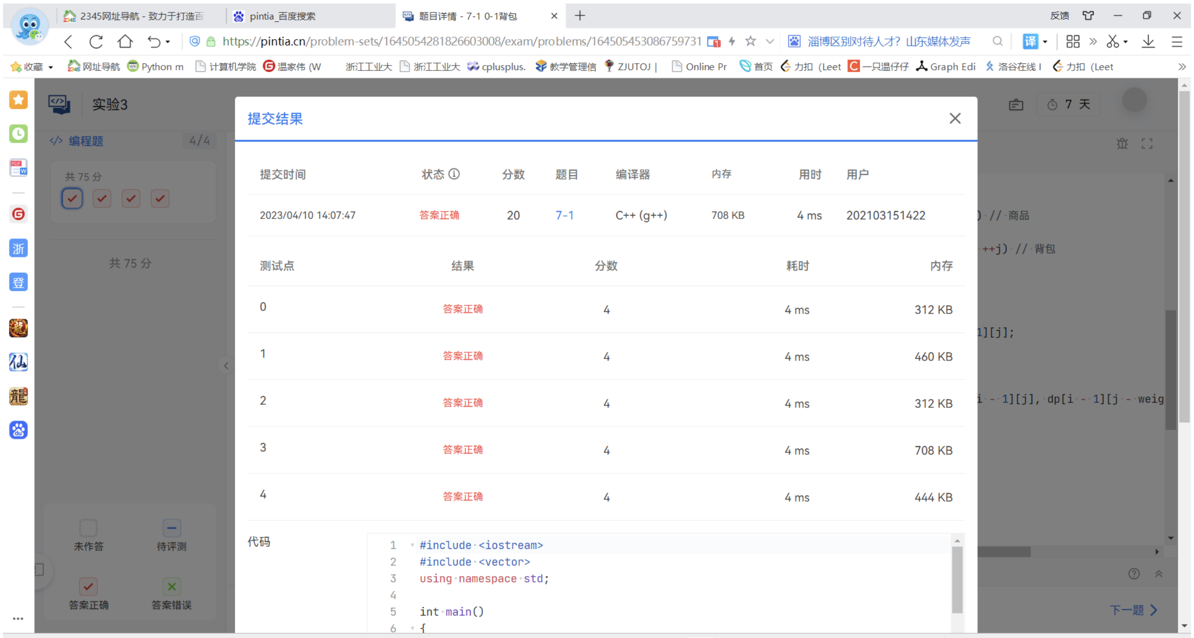
1.1 PTA提交代码截图

C++ (g++)

```cpp
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    int n, c;
    cin >> n >> c;
    vector<int> weight;
    vector<int> values;
    weight.resize(n, 0);
    values.resize(n, 0);
    for (size_t i = 0; i < n; ++i)
    {
        cin >> weight[i] >> values[i];
    }
    vector<vector<int>> dp;
    // 状态: dp[i][j]—前i个物品放入大小为j的背包能达到的最大价值
    dp.resize(n + 1);
    // 初始化: dp[0][j] = dp[i][0] = 0
    // 没装物品价值为0
    for (auto& e : dp)
    {
        e.resize(c + 1, 0);
    }
    for (int i = 1; i < n + 1; ++i) // 商品
    {
        for (int j = 1; j < c + 1; ++j) // 背包
        {
            // 第i个商品大于j, 装不下
            if (weight[i - 1] > j)
            {
                dp[i][j] = dp[i - 1][j];
            }
            else
            {
                dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - wei
            }
        }
    }
    cout << dp[n][c] << endl;
    return 0;
}
```

1.2 PTA提交代码结果截图



1.3 算法分析

> 定义状态$dp[i][j]$前i个物品放入大小为j的背包能达到的最大价值。初始化：
> $dp[0][j] = dp[i][0] = 0$（没装物品价值为0）。转移方程：
> $dp[i][j] = max\{dp[i-1][j], dp[i-1][j-wi]+vi\}$

2.矩阵连乘问题

2.1 PTA提交代码截图

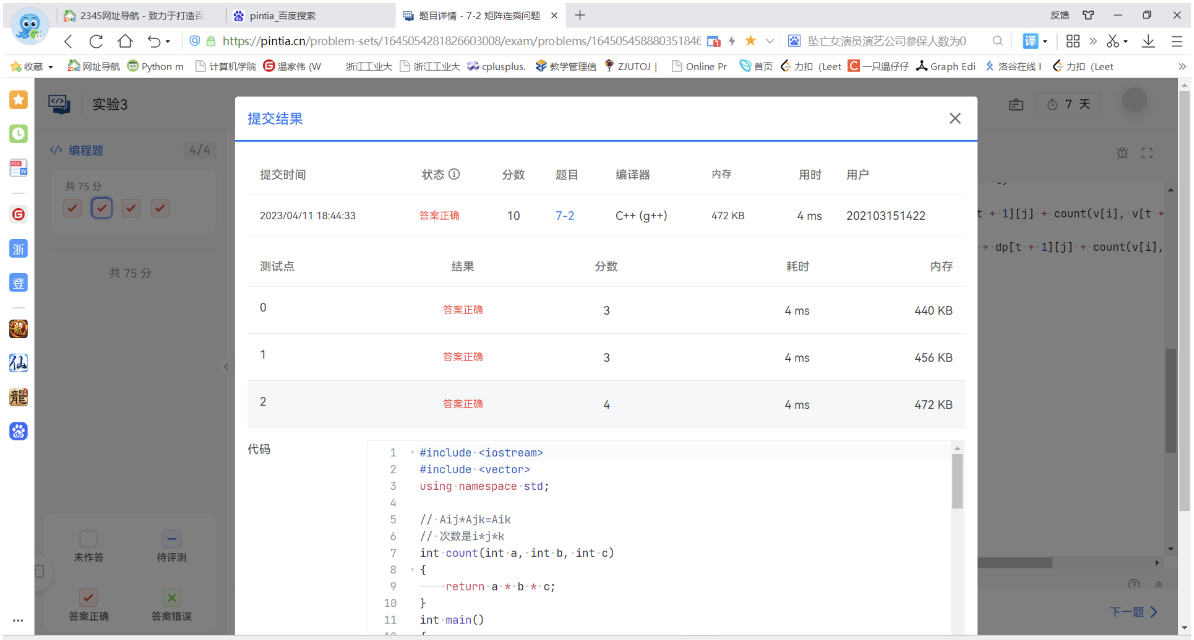
C++ (g++)

```cpp
#include <iostream>
#include <vector>
using namespace std;

// Aij*Ajk=Aik
// 次数是i*j*k
int count(int a, int b, int c)
{
	return a * b * c;
}
int main()
{
	int n;
	cin >> n;
	vector<int> v(n + 1, 0);
	for (auto& e : v)
	{
		cin >> e;
	}
	vector<vector<int>> dp(n);
	// dp[i][j]表示第i+1个矩阵到第j+1个矩阵的最优乘法次数解
	// i+1 = j+1时，即i = j时  dp[i][j] = 0	所以，对角线为0
	for (auto& e : dp)
	{
		e.resize(n, 0);
	}
	for (int k = 0; k < n - 1; ++k)
	{
		for (int i = 0; i < n - 1 - k; ++i)
		{
			int j = i + k + 1;
			int min = 99999999;
			for (int t = i; t < j; ++t)
			{
				if (dp[i][t] + dp[t + 1][j] + count(v[i], v[t +
				{
					min = dp[i][t] + dp[t + 1][j] + count(v[i],
				}
			}
			dp[i][j] = min;
		}
	}/*
	for (auto ei : dp)
	{
		for (auto ej : ei)
```

```
		{
			cout << ej << ' ';
		}
		cout << endl;
	}*/
	cout << dp[0][n - 1] << endl;
	return 0;
}
```

2.2 PTA提交代码结果截图



2.3 算法分析

> 先写一个工具函数count，用于计算$A_{ij}\times A_{jk}=A_{ik}$的运算次数：$i\times j\times k$。然后定义状态：$d[i][j]$表示第$i+1$个矩阵到第$j+1$个矩阵的最优乘法次数解。初始化：$i+1=j+1$时，即$i=j$时 $dp[i][j]=0$。所以，对角线为0。转移方程：
>
> $$dp[i][j]=\begin{cases}0 & \text{i=j}\\ \min\limits_{i\le t<j}\{dp[i][t]+dp[t+1][j]+v_iv_{t+1}v_{j+1}\} & \text{i<j}\end{cases}$$

3.最长公共子序列长度

3.1 PTA提交代码截图

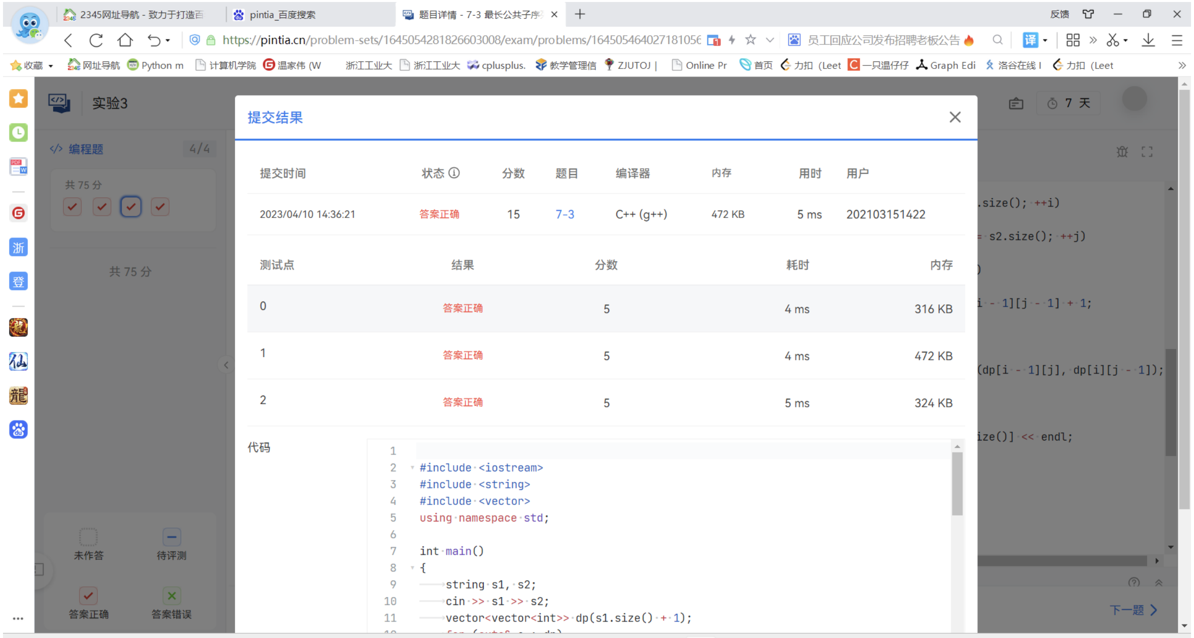
C++ (g++)

```cpp
using namespace std;

int main()
{
    string s1, s2;
    cin >> s1 >> s2;
    vector<vector<int>> dp(s1.size() + 1);
    for (auto& e : dp)
    {
        e.resize(s2.size() + 1, 0);
    }
    bool flag = true;
    for (auto e : s2)
    {
        for (auto f : s1)
        {
            if (e == f)
            {
                flag = false;
                break;
            }
        }
    }
    if (flag)
    {
        cout << 0 << endl;
    }
    else
    {
        for (size_t i = 1; i <= s1.size(); ++i)
        {
            for (size_t j = 1; j <= s2.size(); ++j)
            {
                if (s1[i] == s2[j])
                {
                    dp[i][j] = dp[i - 1][j - 1] + 1;
                }
                else
                {
                    dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]
                }
            }
        }
        cout << dp[s1.size()][s2.size()] << endl;
    }
    return 0;
```

```
}
```

3.2 PTA提交代码结果截图



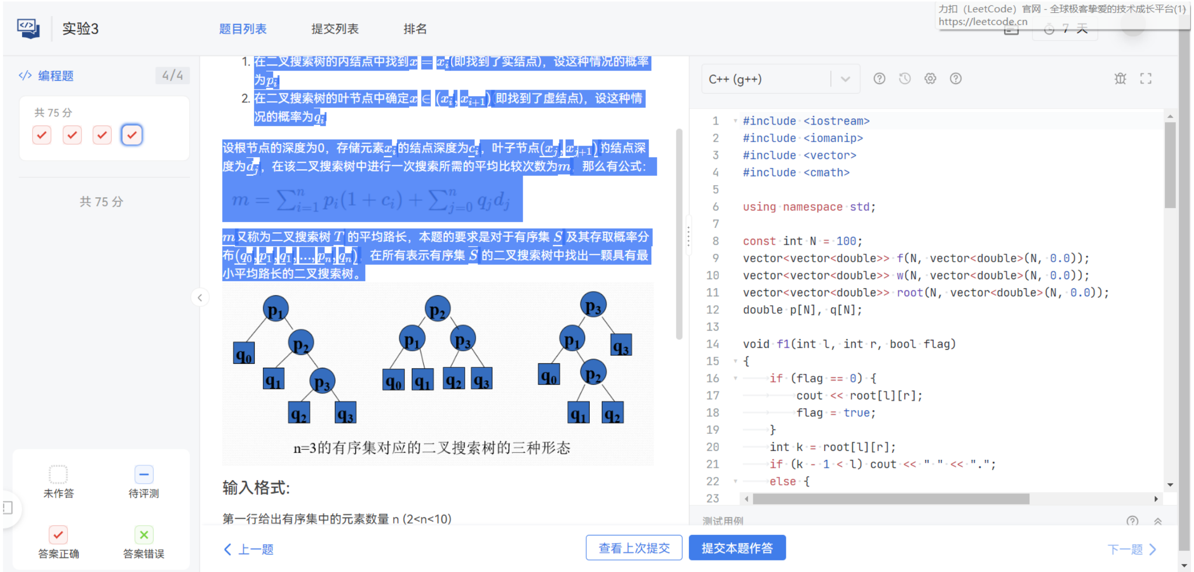
3.3 算法分析

1. 定义状态
   设f[i][j] 表示s1的前i个字符与s2的前j个字符的LCS长度。
2. 找出状态转移方程
   当s1[i] == s2[j]时，$f[i][j] = f[i-1][j-1]+1$，表示s1的前i个字符和s2的前j个字符中最后一个字符相同，因此LCS长度加1。
   当s1[i] != s2[j]时，$f[i][j] = max(f[i-1][j], f[i][j-1])$，表示s1的前i个字符和s2的前j个字符中最后一个字符不同，LCS长度不变，需要在s1的前i-1个字符和s2的前j个字符中找到一个LCS。
3. 初始化状态
   初始化$f[0][j] = 0$，表示s1为空字符串时，与s2的前j个字符的LCS长度为0。
   初始化f[i][0] = 0，表示s2为空字符串时，与s1的前i个字符的LCS长度为0。
4. 计算状态
   根据状态转移方程，从f[1][1]开始计算，依次填充f数组的每个元素，直到$f[m][n]$，其中m和n分别为s1和s2的长度。
   最终的答案即为$f[m][n]$，表示s1和s2的最长公共子序列的长度。

时间复杂度为O(mn)，空间复杂度为O(mn)。

4.最优二叉搜索树

4.1 PTA提交代码截图



4.2 PTA提交代码结果截图

4.3 算法分析

> 假设有 n 个键 $k_1, k_2, \ldots, k_n$ 和 $n+1$ 个虚拟键 $d_0, d_1, \ldots, d_n$，其中 d_0 表示所有小于 k_1 的键，d_i 表示所有位于 k_i 和 k_{i+1} 之间的键，d_n 表示所有大于 k_n 的键。假设每个键 k_i 的概率为 p_i，每个虚拟键 d_i 的概率为 q_i，且 $\sum_{i=1}^{n} p_i + \sum_{i=0}^{n} q_i = 1$。我们的目标是构建一个最优二叉搜索树，使得搜索代价最小。
> 为了方便计算，我们先定义一些变量。设 $e[i,j]$ 表示从 i 到 j 的最小搜索代价，$w[i,j] = \sum_{k=i}^{j} p_k + \sum_{k=i-1}^{j} q_k$ 表示从 i 到 j 所有键的概率之和。我们还定义 $root[i,j]$ 表示从 i 到 j 的最优子树的根节点，$1 \le i \le j \le n$。显然，当 $i=j$ 时，$e[i,j] = p_i, q_{i-1}, q_j$，即只有一个键或一个虚拟键的情况。当 $i<j$ 时，我们可以枚举从 i 到 j 的最优子树的根节点 r，则：
>
> $$e[i,j] = \min_{i \le r \le j} \{e[i,r-1] + e[r+1,j] + w[i,j]\}$$
>
> 注意到 $e[i,j]$ 是由 $e[i,r-1]$ 和 $e[r+1,j]$ 得到的，因此在计算 $e[i,j]$ 的时候，$e[i,r-1]$ 和 $e[r+1,j]$ 应该已经被计算好了。因此我们采用动态规划的方式，从小到大计算 $e[i,j]$。最终的结果是 $e[1,n]$。
> 在计算 $e[i,j]$ 的同时，还需要计算 $root[i,j]$。具体来说，当 $i=j$ 时，$root[i,j] = i$，即只有

> 一个键或一个虚拟键的情况。当 $i<j$ 时，我们可以枚举从 i 到 j 的最优子树的根节点 r，则：
>
> $$root[i,j]=\underset{i\le r\le j}{\operatorname{argmin}}\{e[i,r-1]+e[r+1,j]+w[i,j]\}$$
>
> 其中 argmin 表示取使得函数取得最小值的参数值。注意到 $root[i,j]$ 是由 $e[i,j]$ 得到的，因此在计算 $e[i,j]$ 的时候，$root[i,j]$ 也应该已经被计算好了。最终的结果是 $root[1,n]$。
> 最后，我们可以通过 $root$ 数组来构建最优二叉搜索树。具体来说，我们首先构建根节点，其键为 $k_{root[1,n]}$，然后递归地构建左右子树。对于 $[i,root[i,j]-1]$ 这个区间的键，我们递归地构建左子树，对于 $[root[i,j]+1,j]$ 这个区间的键，我们递归地构建右子树。递归的边界条件是 $i>j$，此时返回空节点。