



**中南大學**  
CENTRAL SOUTH UNIVERSITY

# 算法设计与分析

## 实验报告(动态规划)

学生姓名	杨凯楠
学 号	8208201004
专业班级	信息安全 2002 班
指导教师	石峰
学 院	计算机学院
完成时间	2021 年 12 月 18 日

**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。

**二、实验内容：**编程实现讲过的例题：最长公共子序列问题、矩阵连乘问题、凸多边形最优三角剖分问题、电路布线问题等。本实验中的问题，设计出算法并编程实现。

1. 石子合并

在一个圆形操场的四周摆放着  $n$  堆石子( $n \leq 100$ )，现要将石子有次序地合并成一堆。规定每次只能选取相邻的两堆合并成新的一堆,并将新的一堆的石子数,记为该次合并的得分。编一程序，由文件读入堆栈数  $n$  及每堆栈的石子数( $\leq 20$ )。

选择一种合并石子的方案,使得做  $n-1$  次合并,得分的总和最小；

输入数据：

第一行为石子堆数  $n$ ；

第二行为每堆的石子数,每两个数之间用一个空格分隔。

输出数据：

从第一至第  $n$  行为得分最小的合并方案。第  $n+1$  行是空行.从第  $n+2$  行到第  $2n+1$  行是得分最大合并方案。每种合并方案用  $n$  行表示，其中第  $i$  行( $1 \leq i \leq n$ )表示第  $i$  次合并前各堆的石子数(依顺时针次序输出，哪一堆先输出均可)。要求将待合并的两堆石子数以相应的负数表示。

Sample Input

4

4 5 9 4

Sample Output

-4 5 9 -4

-8 -5 9

-13 -9

22 4 -5 -9 4

4 -14 -4

-4 -18

22

2. 最小代价子母树

设有一排数，共  $n$  个，例如：22 14 7 13 26 15 11。任意 2 个相邻的数可以进行归并，归并的代价为该两个数的和,经过不断的归并，最后归为一堆，而全部归并代价的和称为总代价，给出一种归并算法，使总代价为最小。

输入、输出数据格式与“石子合并”相同。

Sample Input

4

12 5 16 4

Sample Output

- 1 2 - 5 1 6 4

1 7 - 1 6 - 4  
- 1 7 - 2 0  
3 7

### 3. 求能获得的最大喜爱度

输入描述：

第一行输入为：能使用的钱的总额  $X$  和商品种类的总数  $N$ ，两个数字通过空格隔开

第二行开始，每行为一个单一的商品，包括的信息是：商品单价  $P$ 、该商品个数  $amt$ 、该商品喜爱度  $fav$

示例：

输入：

10 4

2 2 2

5 2 2

4 1 3

9 1 3

输出：

7 2

1 2

3 1

解释：用 10 块钱，最大的喜爱度是 7，购买 2 种商品：购买 第一种商品 2 个，得到的喜爱度是 4；再购买第三种商品 1 个，得到的喜爱度是 3。

### 4. 一个正整数数组 $arr$ ，数组满足 $0 \leq arr[i] \leq 9$ 。挑出任意个数，组成一个最大数，并且这个数能被 3 整除，并以字符串的形式返回

### 5. \*基因问题

已知两个基因序列如  $s$ : AGTAGT;  $t$ : ATTAG。现要你给序列中增加一些空格后，首先使得两个序列的长度相等，其次两个串对应符号匹配得到的值最大。基因只有四种分别用 A、G、C、T 表示，匹配中不允许两个空格相对应，任意两符号的匹配值由下表给出：

	A	G	C	T	∪
A	5	-2	-1	-2	-4
G	-2	5	-4	-3	-2
C	-1	-4	5	-5	-1
T	-2	-3	-5	5	-2
∪	-4	-2	-1	-2	

### 三、具体设计

石子合并是最小代价子母树的升级版，故而先叙述最小代价子母树。

#### 1. 最小代价子母树

**分析：**设 $dp[i][k]$ 表示以 $i$ 为起点，长度为 $k$ 的直线上各堆石子的最优合并状态

$sum[i][k]$ 表示以 $i$ 为起点，长度为 $k$ 的直线上各堆石子的总分，对于每一条长度为 $k$ 的直线有 $k-1$ 种划分方法，枚举后，就求出最优值

如：4 4 5 9,看成已经合并后的一个大堆，共有3种划分，即

4 459;44 59;445 9;看成是这三种情况下每两个堆的合并

$$\begin{aligned} dp[1][4] &= dp[1][1] + dp[2][3] + sum[1][4] \\ &= dp[1][2] + dp[3][2] + sum[1][4] \\ &= dp[1][3] + dp[4][1] + sum[1][4] \end{aligned}$$

子问题再类似分解

$$\begin{aligned} dp[2][3] &= dp[2][1] + dp[3][2] + sum[2][3] \\ &= dp[2][2] + dp[4][1] + sum[2][3] \end{aligned}$$

$$dp[2][2] = dp[2][1] + dp[3][1] + sum[2][2]$$

$$dp[1][2] = dp[1][1] + dp[2][1] + sum[1][2]$$

$$dp[3][2] = dp[3][1] + dp[4][1] + sum[3][2]$$

$$\begin{aligned} dp[1][3] &= dp[1][1] + dp[2][2] + sum[1][3] \\ &= dp[1][2] + dp[3][1] + sum[1][3] \end{aligned}$$

而 $dp[1][1]=4; dp[2][1]=4; dp[3][1]=5; dp[4][1]=9;$

从上递归或从下递推都能求得最优值

也就是要把 $dp[][]$ 和 $sum[][]$ 这两张表填写完毕

按照假设，这两张表要按列来填

本道题与第一道题类似，反而比第一道题稍微简单一点

#### 详细设计：


```
1. #include <cstring>
2. #include<iostream>
3. #include<algorithm>
4. using namespace std;
5. int sum[20][20] = { 0 };
6. int dp[20][20] = { 0 };
7. int wealth[20] = { 0 };
8. int n = 0;
9. int flag[20][20] = { 0 };
10. int cnt;
11. void setmy() {
12.     cin >> n;
13.     cnt = n;
14.     for (int i = 1; i <= n; i++) {
15.         cin >> wealth[i];
16.     }
17.     memset(dp, -1, sizeof(dp));
```

```

18.     for (int i = 0; i <= n; i++) {
19.         sum[i][1] = wealth[i];
20.     }
21.     for (int i = 2; i <= n; i++) {
22.         for (int j = 1; j <= n - 1 && i + j <= n + 1; j++)
23.             sum[j][i] += sum[j + 1][i - 1] + sum[j][1];
24.     }
25. }
26. int Min(int s, int l) {
27.     if (dp[s][l] != -1) {
28.         return dp[s][l];
29.     }
30.     int temp=0;
31.     cnt--;//-----
32.     dp[s][l] = 10000;
33.     if (l == 1) {
34.         dp[s][l] = 0;
35.         return 0;
36.     }
37.     for (int i = 1; i < l; i++) {
38.         temp = Min(s,i) + Min(s + i,l - i) + sum[s][1];
39.
40.         if (temp < dp[s][l]) {
41.             dp[s][l] = temp;
42.         }
43.     }
44.     return dp[s][l];
45. }
46. int main(void) {
47.     setmy();
48.     cout << Min(1, n);
49.     return 0;
50. }

```

实现:



```

Microsoft Visual Studio 调试控制台
> 4
e12 5 16 4
174
D:\杨凯楠\ykn\算法\实验报告\algorithm

```

调试分析:

这道题我写了很久的记录路径，但还是没成功，是我的水平太差了。其实这道题是我写的第一道动态规划题，本身对动态规划处于初步认识的阶段，所以我在各种出错以后从网上别人写的博客中学习了一下，但之后的动态规划题就都是自己写的了。

## 2. 石子归并

分析：本题是最小代价子母树的升级。最重要的一步就是变环为线，对其取模就好。

Demo: [click here](#)(代码就不贴了，因为和第一道题高度相似，放一个超链接在这里)

### 运行结果：



```
Microsoft Visual Studio 调试控制台
4
4 5 9 4
-----
54
-----
43
D:\杨凯楠\ykn\算法\实验报告\algorithm\Debug\
```

## 3. 求能获得的最大喜爱度

输入描述：

第一行输入为：能使用的钱的总额  $X$  和商品种类的总数  $N$ ，两个数字通过空格隔开

第二行开始，每行为一个单一的商品，包括的信息是：商品单价  $P$ 、该商品个数  $amt$ 、该商品喜爱度  $fav$

示例：

输入：

```
10 4
2 2 2
5 2 2
4 1 3
9 1 3
```

输出：

```
7 2
1 2
3 1
```

解释：用 10 块钱，最大的喜爱度是7，购买2种商品：

购买 第一种商品2 个，得到的喜爱度是 4；再购买第三种商品 1 个，得到的喜爱度是 3。

**分析：**这道题其实是背包问题的升级版，不同的是普通背包问题每种物品只有一种，而这道题每种物品可能不只有一种。本实验有两个限制条件，即在规定的金额内，规定的物品数量内达到最大的喜爱度。

$dp[money]$ 表示用 $money$ 多的钱获得的最大喜爱度。

$mem[j]$ 表示在花费为 $j$ 多钱的情况下新增的商品序号和个数。

### 核心过程:

```
for (int k = 1; k <= dat[i].amt && money - k * dat[i].p >= 0; k++)  
dp[j] = max(dp[j], dp[j - k * dat[i].p] + k * dat[i].fav);
```

### 详细设计:

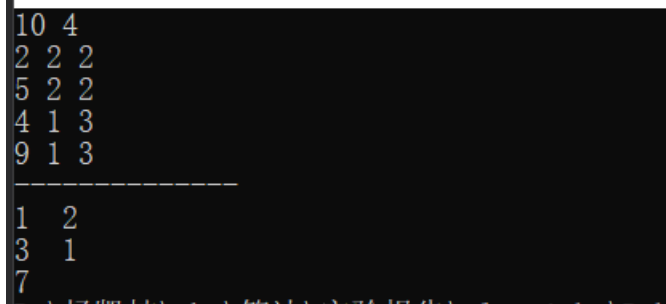
```
1. #include<iostream>  
2. #include<algorithm>  
3. using namespace std;  
4. int money;  
5. int n;  
6. struct Data {  
7.     int p;  
8.     int amt;  
9.     int fav;  
10. };  
11. Data dat[20];  
12. int dp[50] = { 0 };  
13. struct Mem {  
14.     int num;  
15.     int amt;  
16. };  
17. Mem mem[20];  
18. void setmy() {  
19.     cin >> money >> n;  
20.     for (int i = 1; i <= n; i++) {  
21.         cin >> dat[i].p >> dat[i].amt >> dat[i].fav;  
22.     }  
23.  
24. }  
25.  
26. int main(void) {  
27.     setmy();  
28.     int temp = 0;  
29.     for (int i = 1; i <= n; i++) {  
30.         for (int j = money; j > dat[i].p; j--) {  
31.             for (int k = 1; k <= dat[i].amt && money - k * dat[i].p >  
                 = 0; k++) {  
32.                 temp = dp[j];  
33.                 dp[j] = max(dp[j], dp[j - k * dat[i].p] + k * dat[i].  
                     fav);  
34.                 if (dp[j] != temp) {  
35.                     mem[j].amt = k;  
36.                     mem[j].num = i;
```

```

37.         }
38.     }
39. }
40. }
41.     temp = 0;
42.     printf("-----\n");
43.     for (int i = 0; i <= money; i++) {
44.         if (mem[i].num == temp) {
45.             continue;
46.         }
47.         printf("%d %d\n", mem[i].num, mem[i].amt);
48.         temp = mem[i].num;
49.     }
50.     cout << dp[money];
51.     return 0;
52. }

```

实现:



```

10 4
2 2 2
5 2 2
4 1 3
9 1 3
-----
1 2
3 1
7

```

**测试分析:** 本题在输出具体装入问题时采用前后对比法，因为`dp[]`就包含了前一子问题的结果，所以采用从头开始的前后两两对比法，当发生变化时，说明该物品被装入背包内，且装入背包的个数也是很容易确定的。

#### 4. 一个正整数数组 `arr`，数组满足 $0 \leq arr[i] \leq 9$ 。

挑出任意个数，组成一个最大数，并且这个数能被 3 整除，并以字符串的形式返回

**分析:**

本题用动态规划很好解决，而我在这道题中另开辟了一种算法，现在介绍这种算法。首先我们知道，一个数能被3整除的充要条件是这个数的各个位加起来也能被三整除，接着引入离散数学中同余和等价关系的概念，不难得出，数组长度，即数的个数只能有三种情况：`[0],[1],[2]`。而数组中每个数也只有`[0],[1],[2]`三种情况，不难看出，当去掉`n`个数中的至多两个数时，该问题必有最优解，经过分析，最坏时间复杂度为 $O(n^2)$ ，但是如果对数组进行预处理，绝大多数情况下可以在很短时间内解决，且位数越多解题速度越快。

**详细设计:**

```

1. #include<stdio>

```



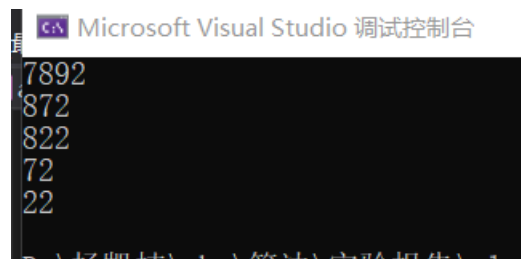
```
2. #include<algorithm>
3. #include<ctime>
4. #include<vector>
5. using namespace std;
6. bool cmp(int a,int b){
7.     return a > b;
8. }
9. int main(void) {
10.     unsigned long seed = time(0);
11.     srand(seed);
12.     int tempsum = 0;
13.     int flag = 0;
14.     unsigned long long num = long long(rand());
15.     int arr[20], len = 0;
16.     printf("%lld\n", num);
17.     vector<int>a,b;
18.     for (len = 0; num; len++) {
19.         arr[len] = num % 10;
20.         num /= 10;
21.         tempsum += arr[len];
22.         a.push_back(arr[len]);
23.         b.push_back(arr[len]);
24.     }
25.     sort(a.begin(), a.end(), cmp);
26.     sort(b.begin(), b.end(), cmp);
27.     if (tempsum % 3 == 0) {
28.         sort(arr, arr + len, cmp);
29.         for (int i = 0; i < len; i++) {
30.             printf("%d", arr[i]);
31.         }
32.         return 0;
33.     }
34.     for (int i = 0; i < len; i++) {
35.         if ((tempsum - arr[i]) % 3 == 0) {
36.             flag = 0;
37.             a.erase(a.begin() + i);
38.             sort(a.begin(), a.end(),cmp);
39.             for (int k = 0; k <a.size(); k++) {
40.                 printf("%d", a[k]);
41.             }printf("\n");
42.             a.insert(a.begin() + i, arr[i]);
43.         }
44.     }
45.     for (int i = 0; i < len-1; i++) {
```

```

46.         for (int j = i + 1; j < len; j++) {
47.             if ((tempsum - arr[i] - arr[j]) % 3 == 0) {
48.                 b.erase(b.begin() + j);
49.                 b.erase(b.begin() + i);
50.                 sort(b.begin(), b.end(), cmp);
51.                 for (int k = 0; k < b.size(); k++) {
52.                     printf("%d", b[k]);
53.                 }
54.                 printf("\n");
55.                 b.insert(b.begin() + i, arr[i]);
56.                 b.insert(b.begin() + j, arr[j]);
57.             }
58.         }
59.     }
60.     return 0;
61. }

```

运行结果:



Microsoft Visual Studio 调试控制台

```

7892
872
822
72
22

```

测试分析:

如图，生成的是数字 7312，可以将所有去掉两位数字以内的三的倍数都求出来，并且输出时就可以将所有从大到小输出，不需要排序，（题目要求是最大的，需要时只用输出第一个就好）。

## 5. 基因问题

已知两个基因序列如s: AGTAGT; t: ATTAG。现要你给序列中增加一些空格后，首先使得两个序列的长度相等，其次两个串对应符号匹配得到的值最大。基因只有四种分别用A、G、C、T表示，匹配中不允许两个空格相对应，

分析:

这道题是最大公共子序列的升级版，同样是从最后一个开始匹配

mv[i],[j]表示 s 串的前 i 个碱基和 t 串的前 j 个碱基的相似度match value

不同的是最后有三种情况

- 1.碱基与碱基匹配
- 2.碱基与空格匹配

### 3.空格与碱基匹配

则状态转移方程为 $mv[i][j]=\max(mv[i][j],mv[i-1][j]+w[s[i]][k],mv[i][j-1]+w[k][t[j]])$

其中 $s[i],t[j]$ 为 $s,t$ 串中第 $i,j$ 个字母,  $w$ 为匹配值,  $k$ 为空格

边界条件: 递推到一条没了以后那么

$$mv[i][0]=mv[i-1][0]+w[s[i]][k], \text{或} mv[0][j]=mv[0][j-1]+w[k][t[j]]$$

双方都完了那么 $f[0][0]=0$ ;

为了简化输入, 提前输好每个串的长度

#### 详细设计:

```
1.  // #include <iostream>
2.  // #include <algorithm>
3.  #include <bits/stdc++.h>
4.  using namespace std;
5.  int lens = 0, lent = 0, k = 5;
6.  int w[6][6] = {
7.      {0,0,0,0,0,0},
8.      {0,5,-2,-1,-2,-4},
9.      {0,-2,5,-4,-3,-2},
10.     {0,-1,-4,5,-5,-1},
11.     {0,-2,-3,-5,5,-2},
12.     {0,-4,-2,-1,-2,0}
13. };
14. struct Mem {
15.     int s;
16.     int t;
17. };
18. //Mem mem[30][30];
19. int s[200] = { 0 };
20. int t[200] = { 0 };
21. int mv[200][200] = { 0 };
22. void setmy() {
23.     cin >> lens >> lent;
24.     char a;
25.     for (int i = 1; i <= lens; i++) {
26.         cin >> a;
27.         switch(a){
28.             case 'A': s[i] = 1; break;
29.             case 'G': s[i] = 2; break;
30.             case 'C': s[i] = 3; break;
31.             case 'T': s[i] = 4; break;
32.         }
33.     }
34.     for (int i = 1; i <= lent; i++) {
```

```

35.         cin >> a;
36.         switch (a) {
37.             case 'A':t[i] = 1; break;
38.             case 'G':t[i] = 2; break;
39.             case 'C':t[i] = 3; break;
40.             case 'T':t[i] = 4; break;
41.         }
42.     }
43. }
44.
45. void solution() {
46.     mv[0][0] = 0;
47.     for (int i = 1; i <= lens; i++) {
48.         mv[i][0] = mv[i - 1][0] + w[s[i]][k];
49.     }
50.     for (int j = 1; j <= lent; j++) {
51.         mv[0][j] = mv[0][j - 1] + w[k][t[j]];
52.     }
53.     for (int i = 1; i <= lens; i++)
54.         for (int j = 1; j <= lent; j++) {
55.             mv[i][j] = max(max(mv[i-1][j-
1]+w[s[i]][t[j]], mv[i - 1][j] + w[s[i]][k]), mv[i][j - 1] + w[k][t[j
]]);
56.         }
57.
58.
59.     printf("%d", mv[lens][lent]);
60.
61. }
62. int main(void) {
63.     setmy();
64.     solution();
65.     return 0;
66. }

```

实现:

```

Microsoft
7 5
ATGATCC
GCTAT
8

```

**测试分析:** 这道题实在不会写具体结果，水平还很低，技术不够，还需要学习。但是根据计算，结果是正确的，（放到平台上可以AC）。

## 四、心得体会

通过本次动态规划实验，我深刻理解到了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，体会了动态规划解决问题的巧妙性。