

5.22 《DP 区间问题》课堂笔记

一、 本周作业

1. P3146
2. P1220

二、 上周作业

1. 最大字典序

8、背包问题求具体方案

有 N 件物品和一个容量是 M 的背包。每件物品只能使用一次。
第 i 件物品的体积是 w_i ，价值是 v_i 。
求解将哪些物品装入背包，可使这些物品的总体积不超过背包容量，且总价值最大。
输出字典序最大的方案。这里的字典序是指：所选物品的编号所构成的序列。
物品的编号范围是 $1 \dots N$ 。

输入格式
第一行两个整数， N, V ，用空格隔开，分别表示物品数量和背包容积。
接下来有 N 行，每行两个整数 w_i, v_i ，用空格隔开，分别表示第 i 件物品的体积和价值。

输出格式
输出一行，包含若干个用空格隔开的整数，表示最优解中所选物品的编号序列，且该编号序列的字典序最大。
物品编号范围是 $1 \dots N$ 。

数据范围
 $0 < N, V \leq 1000$
 $0 < w_i, v_i \leq 1000$

输入样例
4 7
1 1
2 3
3 4
4 5
输出样例
3 4

完整代码：

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

int _n, _m;
int _w[110];
int _v[110];
int _used[110];
int _dp[110][110];

int main() {
    cin >> _m >> _n;
    for(int i = 1; i <= _m; i++) {
        cin >> _w[i] >> _v[i];
    }

    for(int i = 1; i <= _m; i++) {
        for(int j = 1; j <= _n; j++) {
            if(j < _w[i]) {
                _dp[i][j] = _dp[i - 1][j];
            }
            if(j >= _w[i]) {
                _dp[i][j] = max(_dp[i - 1][j], _dp[i - 1][j - _w[i]] + _v[i]);
            }
        }
    }

    int j = _n; // 剩余容量
    for(int i = _m; i >= 0; i--) {
        if(j >= _w[i] && _dp[i][j] == _dp[i - 1][j - _w[i]] + _v[i]) { // 选了
            _used[i] = 1;
            j -= _w[i];
        }
    }

    for(int i = 1; i <= _m; i++) {
        if(_used[i] != 0) {
            cout << i << " ";
        }
    }

    // cout << _dp[_m][_n];
    return 0;
}
```

2. 宠物精灵

一天，小智和皮卡丘来到了小精灵狩猎场，里面有很多珍贵的野生宠物小精灵。小智也想收服其中的一些小精灵。然而，野生小精灵并不容易收服。对于每一个野生小精灵而言，小智可能需要使用很多个精灵球才能收服它，而在收服过程中，野生小精灵也会对皮卡丘造成一定的伤害（从而减少皮卡丘的体力）。当皮卡丘的体力小于0时，小智就必须结束狩猎（因为他需要给皮卡丘疗伤），而使得皮卡丘体力小于0的野生小精灵也不会被小智收服。当小智的精灵球用完时，狩猎也宣告结束。

我们假设小智遇到野生小精灵时有两个选择：收服它，或者离开它。如果小智选择了收服，那么一定会扔出能够收服该小精灵的精灵球，而皮卡丘也一定会受到相应的伤害；如果选择离开它，那么小智不会损失精灵球，皮卡丘也不会损失体力。

小智的目标有两个：主要目标是收服尽可能多的野生小精灵；如果可以收服的小精灵数量一样，小智希望皮卡丘受到的伤害越小（剩余体力越大），因为他们还要继续冒险。

现在已知小智的精灵球数量和皮卡丘的初始体力，已知每一个小精灵需要的用于收服的精灵球数目和它在被收服过程中会对皮卡丘造成的伤害数目。请问，小智该如何选择收服哪些小精灵以达到他的目标呢？

输入

输入数据的第一行包含三个整数：N(0 < N < 1000)，M(0 < M < 500)，K(0 < K < 100)，分别代表小智的精灵球数量、皮卡丘初始的体力值、野生小精灵的数量。

之后的K行，每一行代表一个野生小精灵，包括两个整数：收服该小精灵需要的精灵球的数量，以及收服过程中对皮卡丘造成的伤害。

输出

输出为一行，包含两个整数：C，R，分别表示最多收服C个小精灵，以及收服C个小精灵时皮卡丘的剩余体力值最多为R。

样例输入1:

10 100 5

7 10

2 40

2 50

1 20

4 20

样例输出1:

3 30

完整代码:

```
#include <iostream>
#include <cstring>

using namespace std;
int _m, _n, _k;
int _a[1010]; // 精灵球
int _b[1010]; // 体力值
int _dp[1010][1010];

void ParseIn() {
    cin >> _n >> _m >> _k;
    for(int i = 1; i <= _k; i++) {
        cin >> _a[i] >> _b[i];
    }
}

void Core() {
    for(int i = 1; i <= _k; i++) {
        for(int j = _n; j >= _a[i]; j--) {
            for(int k = _m; k >= _b[i]; k--) {
                _dp[j][k] = max(_dp[j][k], _dp[j - _a[i]][k - _b[i]] + 1);
            }
        }
    }
}

void WriteOut() {
    int res = 0;
    cout << _dp[_n][_m] << " ";
    for(int i = _m; i >= 0; i--) {
        if(_dp[_n][i] == _dp[_n][_m]) {
            res = i;
        }
    }
    cout << _m - res;
}

int main()
{
    ParseIn();
    Core();
    WriteOut();

    return 0;
}
```

三、 课堂内容

区间 DP 问题

例 1：合并石子

设有N堆石子排成一排，其编号为1,2,3...,N。（N<100）
每堆石子有一定的质量，可以用一个整数来描述，现在要把这N堆石子合并成一堆。每次只能合并相邻的两堆，合并的代价为这两堆石子的质量之和，合并后与这两堆石子相邻的石子将和新堆相邻，合并时由于选择的顺序不同，合并的总代价也不相同。
找出一种合理方案，使总代价最小并输出。

样例输入：

4

4 1 1 4

样例输出：

18

完整代码：

```
#include <iostream>
#include <cstring>
using namespace std;
int _n = 0;
int _num[110];
int _sum[110]; // 前缀和数组
int _dp[110][110];
int main () {
    memset(_dp, 0x3f, sizeof(_dp)); // 初始化dp数组
    cin >> _n;
    for(int i = 1; i <= _n; i++) {
        cin >> _num[i];
        _sum[i] = _sum[i - 1] + _num[i]; // sum数组保存前缀和
        _dp[i][i] = 0;
    }

    for(int len = 2; len <= _n; len++) { // 枚举区间长度
        for(int i = 1; i + len - 1 <= _n; i++) { // 枚举区间起点
            int j = i + len - 1; // 枚举区间终点
            for(int k = i; k < j; k++) { // 枚举分割点
                _dp[i][j] = min(_dp[i][j], _dp[i][k] + _dp[k + 1][j] + _sum[j] - _sum[i - 1]);
            }
        }
    }
    cout << _dp[1][_n]; // 完整区间最小代价
    return 0;
}
```

例 2: P1880 环形区间石子合并

题目描述

[复制Markdown](#) [展开](#)

在一个圆形操场的四周摆放 N 堆石子，现要将石子有次序地合并成一堆，规定每次只能选相邻的2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。

试设计出一个算法,计算出将 N 堆石子合并成 1 堆的最小得分和最大得分。

输入格式

数据的第 1 行是正整数 N ，表示有 N 堆石子。

第 2 行有 N 个整数，第 i 个整数 a_i 表示第 i 堆石子的个数。

输出格式

输出共 2 行，第 1 行为最小得分，第 2 行为最大得分。

输入输出样例

输入 #1

[复制](#)

输出 #1

[复制](#)

```
4
4 5 9 4
```

```
43
54
```

说明/提示

$1 \leq N \leq 100, 0 \leq a_i \leq 20$ 。

算法分析：在上题的基础上将区间扩展为 2 倍，枚举的区间长度不变，只需改变枚举起点就可以。

完整代码：

```
//o(n^3)
#include <iostream>
#include <cstring>
using namespace std;
int _n = 0;
int _num[210];
int _sum[210];
int _dp[210][210];
int _mini = 2147483647;
int main () {
    memset(_dp, 0x3f, sizeof(_dp));
    cin >> _n;
    for(int i = 1; i <= _n; i++) {
        cin >> _num[i];
        _num[i + _n] = _num[i]; // 链
    }

    for(int i = 1; i <= 2 * _n; i++) {
        _sum[i] = _sum[i - 1] + _num[i];
        _dp[i][i] = 0;
    }

    int j = 0; // 枚举区间终点
    for(int len = 2; len <= _n; len++) { // 枚举区间长度
        for(int i = 1; i + len - 1 <= 2 * _n; i++) { // 枚举区间起点
            j = i + len - 1; // 区间终点
            for(int k = i; k < j; k++) { // 枚举分割点
                _dp[i][j] = min(_dp[i][j], _dp[i][k] + _dp[k + 1][j] + _sum[j] - _sum[i - 1]);
            }
        }
    }
    // 环形比较
    for(int i = 1; i <= _n; i++) {
        _mini = min(_mini, _dp[i][i + _n - 1]);
    }
    cout << _mini;
    return 0;
}
```

例 3：P1063 能量项链

题目描述

复制Markdown 展开

在 Mars 星球上，每个 Mars 人都随身佩带着一串能量项链。在项链上有 N 颗能量珠。能量珠是一颗有头标记与尾标记的珠子，这些标记对应着某个正整数。并且，对于相邻的两颗珠子，前一颗珠子的尾标记一定等于后一颗珠子的头标记。因为只有这样，通过吸盘（吸盘是 Mars 人吸收能量的一种器官）的作用，这两颗珠子才能聚合成一颗珠子，同时释放出可以被吸盘吸收的能量。如果前一颗能量珠的头标记为 m ，尾标记为 r ，后一颗能量珠的头标记为 r ，尾标记为 n ，则聚合后释放的能量为 $m \times r \times n$ （Mars 单位），新产生的珠子的头标记为 m ，尾标记为 n 。

需要时，Mars 人就用吸盘夹住相邻的两颗珠子，通过聚合得到能量，直到项链上只剩下一颗珠子为止。显然，不同的聚合顺序得到的总能量是不同的，请你设计一个聚合顺序，使一串项链释放出的总能量最大。

例如：设 $N = 4$ ，4 颗珠子的头标记与尾标记依次为 $(2, 3)(3, 5)(5, 10)(10, 2)$ 。我们用记号 \oplus 表示两颗珠子的聚合操作， $(j \oplus k)$ 表示第 j, k 两颗珠子聚合后所释放的能量。则第 4、1 两颗珠子聚合后释放的能量为：

$$(4 \oplus 1) = 10 \times 2 \times 3 = 60。$$

这一串项链可以得到最优值的一个聚合顺序所释放的总能量为：

$$((4 \oplus 1) \oplus 2) \oplus 3 = 10 \times 2 \times 3 + 10 \times 3 \times 5 + 10 \times 5 \times 10 = 710。$$

输入格式

第一行是一个正整数 N ($4 \leq N \leq 100$)，表示项链上珠子的个数。第二行是 N 个用空格隔开的正整数，所有的数均不超过 1000。第 i 个数为第 i 颗珠子的头标记 ($1 \leq i \leq N$)，当 $i < N$ 时，第 i 颗珠子的尾标记应该等于第 $i + 1$ 颗珠子的头标记。第 N 颗珠子的尾标记应该等于第 1 颗珠子的头标记。

至于珠子的顺序，你可以这样确定：将项链放到桌面上，不要出现交叉，随意指定第一颗珠子，然后按顺时针方向确定其他珠子的顺序。

输出格式

一个正整数 E ($E \leq 2.1 \times 10^9$)，为一个最优聚合顺序所释放的总能量。

输入输出样例

输入 #1	复制	输出 #1	复制
4 2 3 5 10		710	

完整代码：

```
#include <iostream>
using namespace std;
int _n = 0;
int _num[210];
int _sum[210];
int _dp[210][210];
int _maxi = 0;
int main () {

    cin >> _n;
    for(int i = 1; i <= _n; i++) {
        cin >> _num[i];
        _num[i + _n] = _num[i]; // 环项链
    }

    int j = 0;
    for(int len = 3; len <= _n + 1; len++) { // 枚举区间长度，len=3时表示两颗珠子的聚合
        for(int i = 1; i + len - 1 <= 2 * _n; i++) { // 枚举区间起点
            j = i + len - 1; // 区间终点
            for(int k = i + 1; k < j; k++) { // 枚举分割点
                _dp[i][j] = max(_dp[i][j], _dp[i][k] + _dp[k][j] + _num[i] * _num[k] * _num[j]);
            } // 两子区间同用k
        }
    }
    // 环形比较
    for(int i = 1; i <= _n; i++) {
        _maxi = max(_maxi, _dp[i][i + _n]); // 长度为n-1
    }
    cout << _maxi;
    return 0;
}
```

例 4：P4170 涂色

题目描述

[复制Markdown](#) [展开](#)

假设你有一条长度为 5 的木板，初始时没有涂过任何颜色。你希望把它的 5 个单位长度分别涂上红、绿、蓝、绿、红色，用一个长度为 5 的字符串表示这个目标：RGBGR。

每次你可以把一段连续的木板涂成一个给定的颜色，后涂的颜色覆盖先涂的颜色。例如第一次把木板涂成 RRRRR，第二次涂成 RGGGR，第三次涂成 RGBGR，达到目标。

用尽量少的涂色次数达到目标。

输入格式

输入仅一行，包含一个长度为 n 的字符串，即涂色目标。字符串中的每个字符都是一个大写字母，不同的字母代表不同颜色，相同的字母代表相同颜色。

输出格式

仅一行，包含一个数，即最少的涂色次数。

输入输出样例

输入 #1	复制	输出 #1	复制
AAAAA		1	
输入 #2	复制	输出 #2	复制
RGBGR		3	

说明/提示

- 40% 的数据满足 $1 \leq n \leq 10$ 。
- 100% 的数据满足 $1 \leq n \leq 50$ 。

完整代码：

```
#include <iostream>
#include <string>
#include <cstring>
#include <algorithm>
using namespace std;
const int maxn = 55;

int f[maxn][maxn]; //f[i][j] 表示涂i-j区间字符串的最少次数

int main()
{
    string str;
    cin >> str;
    memset(f,0x3f,sizeof(f));

    // 初始化，涂一个字符的次数为1
    for (int i = 0; i < str.length(); i++)
    {
        f[i][i] = 1;
    }

    for (int len = 2; len <= str.length(); len++) // 枚举区间长度
    {
        for (int i = 0; i + len - 1 < str.length(); i++)// 枚举起点
        {
            int j = i + len - 1; // 终点
            if (str[i] == str[j])// 左右端点相同，可以一次涂完
                f[i][j] = min(f[i][j - 1], f[i + 1][j]);
            else
            {
                for (int k = i; k < j; k++) // 枚举分割点
                    f[i][j] = min(f[i][j], f[i][k] + f[k + 1][j]);
            }
        }
    }

    cout << f[0][str.length() - 1];
}
```

例 5：CF607B 删除回文

题目描述

Genos最近在他的手机上下载了祖玛游戏。在祖玛游戏里，存在 n 个一行的宝石，第 i 个宝石的颜色是 C_i 。这个游戏的目标是尽快的消灭一行中所有的宝石。在一秒钟，Genos能很快的挑选出这些有颜色的宝石中的一个回文的，连续的子串，并将这个子串移除。每当一个子串被删除后，剩余的宝石将连接在一起，形成一个新的行列。你的任务是：求出把整个宝石串都移除的最短时间。让我们给你一个提示：如果一个串正着读或倒着读都一样，那么这个串（或子串）叫回文串。在我们这道题中，“回文”指这个宝石串中的第一个珠子的颜色等于最后一个珠子的颜色，第二个珠子的颜色等于倒数第二个珠子的颜色，等等。输入输出格式 输入格式：

第一行包含一个整数 n ($1 \leq n \leq 500$) —— 宝石串的长度。第二行包含 n 个被空格分开的整数，第 i ($1 \leq i \leq n$)个表示这行中第 i 个珠子的颜色。 输出格式：

输出一个整数，把这行珠子移除的最短时间。(样例略) 说明：

在第一个例子中，Genos可以在一秒钟就把这行珠子全部移走。在第二个例子中，Genos一次只能移走一个珠子，所以移走三个珠子花费他三秒。在第三个例子中，为了达到2秒的最快时间，先移除回文串4 4,再移除回文串1 2 3 2 1。

感谢@Administrator2004 提供的翻译

输入输出样例

输入 #1

[复制](#)

输出 #1

[复制](#)

```
3
1 2 1
```

```
1
```

输入 #2

[复制](#)

输出 #2

[复制](#)

```
3
1 2 3
```

```
3
```

输入 #3

[复制](#)

输出 #3

[复制](#)

```
7
1 4 4 2 3 2 1
```

```
2
```

算法分析：

```
n=1:   dp[i][i] = 1;
n=2:   1 2: 2
       1 1: 1

n>=3:
    左右两个端点相对: dp[i][j] = dp[i+1][j-1]
        1 2 1 等价于 2
        1 2 3 1等价于 2 3
        1 1 2 3 4 1等价于 1 2 3 4
    左右两个端点不相等的时候: dp[i][j] = dp[i][k]+dp[k+1][j]
    枚举分割点
    1 2 3 2: 1+ 2 3 2      2
    |       |       | 1 2 + 3 2      4
    |       |       | 1 2 3+ 2      4

    考虑特殊情况: 1 2 3 1 1 5 1 :      删掉3次
    删掉端点:      2 3 1 1 5          删掉4次
    所以左右端点相等时，依然要遍历所有断点
```

完整代码：


```

#include<bits/stdc++.h>
using namespace std;
int a[505];
int dp[505][505];
int main()
{
    int n;
    cin>>n;
    for(int i=1;i<=n;i++){
        cin>>a[i];
    }

    memset(dp,0x3f,sizeof(dp));

    for(int i=1;i<=n;i++){
        dp[i][i]=1;
    }

    for(int l=2;l<=n;l++){
        for(int i=1;i+l-1<=n;i++){
            int j=i+l-1;

            if(a[i]==a[j] && l==2){
                dp[i][j]=1;
                //l==2且左右端点相等的情况，比如3，最短时间就为1
            }
            else if(a[i]==a[j]){
                dp[i][j]=dp[i+1][j-1];
            }

            for(int k=i;k<j;k++){ //就算端点相同，也要遍历断点
                dp[i][j]=min(dp[i][j],dp[i][k]+dp[k+1][j]);
            }
        }
    }

    cout<<dp[1][n];
}

```