动态规划-背包问题

一、AcWing 2. 01背包问题

【题目描述】

有N件物品和一个容量是V的背包。每件物品只能使用一次。

第i件物品的体积是v_i，价值是w_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。

输出最大价值。

【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v_i, w_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】

输出一个整数，表示最大价值。

【数据范围】

$0 < N, V \leq 1000$

$0 < v_i, w_i \leq 1000$

【输入样例】

```
4 5
1 2
2 4
3 4
4 5
```

【输出样例】

```
8
```

【朴素版代码】

```
#include <iostream>
```

```cpp
#include <algorithm>
using namespace std;

const int N = 1010;
int f[N][N];
int v[N], w[N];
int n, m;

int main()
{
    cin >> n >> m;
    for (int i = 1; i <= n; i++) cin >> v[i] >> w[i];
    for (int i = 1; i <= n; i++)
        for (int j = 0; j <= m; j++)
        {
            f[i][j] = f[i - 1][j];//左半边的子集
            if (j >= v[i]) f[i][j] = max(f[i][j], f[i - 1][j - v[i]] +
w[i]);
        }
    cout << f[n][m] << endl;
    return 0;
}
```

【空间优化版代码】

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

const int N = 1010;
int f[N], v[N], w[N];
int n, m;

int main()
{
    cin >> n >> m;
    for (int i = 1; i <= n; i++) cin >> v[i] >> w[i];
    for (int i = 1; i <= n; i++)
        for (int j = m; j >= v[i]; j--)
            f[j] = max(f[j], f[j - v[i]] + w[i]);
    cout << f[m] << endl;
}
```

二、AcWing 3. 完全背包问题

【题目描述】

有N种物品和一个容量是V的背包，每种物品都有无限件可用。

第i种物品的体积是v_i，价值是w_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。

输出最大价值。

【输入格式】

第一行两个整数，N,V，用空格隔开，分别表示物品种数和背包容积。

接下来有N行，每行两个整数v_i,w_i，用空格隔开，分别表示第i种物品的体积和价值。

【输出格式】

输出一个整数，表示最大价值。

【数据范围】

$0 < N,V \leq 1000$

$0 < v_i,w_i \leq 1000$

【输入样例】

```
4 5
1 2
2 4
3 4
4 5
```

【输出样例】

```
10
```

【分析】

```
状态表示：f[i][j]表示从前i种物品选择，选出的物品的总体积不大于j的方案集合
状态转移：
f[i][j] = max(f[i - 1][j], f[i - 1][j - v] + w, ..., f[i - 1][j - kv] + kw)
f[i][j - v] =          max(f[i - 1][j - v], f[i - 1][j - 2v] + w, ..., f[i - 1][j - kv] + (k - 1)w)
因此f[i][j] = max(f[i - 1][j], f[i][j - v] + w)

1.01背包：f[i][j] = max(f[i - 1][j], f[i - 1][j - v[i]] + w[i])
2.完全背包：f[i][j] = max(f[i - 1][j], f[i][j - v[i]] + w[i])
```

【朴素版代码】

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

const int N = 1010;
int f[N][N];
int v[N], w[N];
int n, m;

int main()
{
    cin >> n >> m;
    for (int i = 1; i <= n; i++) cin >> v[i] >> w[i];
    for (int i = 1; i <= n; i++)
        for (int j = 0; j <= m; j++)
        {
            f[i][j] = f[i - 1][j];
            if (j >= v[i]) f[i][j] = max(f[i][j], f[i][j - v[i]] + w[i]);
        }
    cout << f[n][m] << endl;
    return 0;
}
```

【空间优化版代码】

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

```

```cpp
const int N = 1010;
int f[N], v[N], w[N];
int n, m;

int main()
{
    cin >> n >> m;
    for (int i = 1; i <= n; i++) cin >> v[i] >> w[i];
    for (int i = 1; i <= n; i++)
        for (int j = v[i]; j <= m; j++)
            f[j] = max(f[j], f[j - v[i]] + w[i]);
    cout << f[m] << endl;
    return 0;
}
```

三、AcWing 4/5. 多重背包问题 I/II

【题目描述】

有N种物品和一个容量是V的背包。

第i种物品最多有s_i件，每件体积是v_i，价值是w_i。

求解将哪些物品装入背包，可使物品体积总和不超过背包容量，且价值总和最大。

输出最大价值。

【输入格式】

第一行两个整数，N, V，用空格隔开，分别表示物品种数和背包容积。

接下来有N行，每行三个整数v_i, w_i, s_i，用空格隔开，分别表示第i种物品的体积、价值和数量。

【输出格式】

输出一个整数，表示最大价值。

【朴素版数据范围】

$0 < N, V \leq 100$

$0 < v_i, w_i, s_i \leq 100$

【二进制优化版数据范围】

$0 < N \leq 1000$

$0 < V \leq 2000$

$0 < v_i, w_i, s_i \leq 2000$

【输入样例】

```
4 5
1 2 3
2 4 1
3 4 3
4 5 2
```

【输出样例】

```
10
```

【朴素版代码】

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

const int N = 110;
int f[N][N];
int v[N], w[N], s[N];
int n, m;

int main()
{
    cin >> n >> m;
    for (int i = 1; i <= n; i++) cin >> v[i] >> w[i] >> s[i];
    for (int i = 1; i <= n; i++)
        for (int j = 0; j <= m; j++)
            for (int k = 0; k <= s[i] && k * v[i] <= j; k++)
                f[i][j] = max(f[i][j], f[i - 1][j - k * v[i]] + k *
w[i]);
    cout << f[n][m] << endl;
    return 0;
}
```

【二进制优化版代码】

```
#include <iostream>
#include <algorithm>
```

```cpp
using namespace std;

const int N = 12010, M = 2010;
int v[N], w[N], f[M];
int n, m, cnt;

int main()
{
    cin >> n >> m;
    for (int i = 1; i <= n; i++)
    {
        int a, b, s;
        cin >> a >> b >> s;
        int k = 1;
        while (k <= s)
        {
            v[++cnt] = k * a;
            w[cnt] = k * b;
            s -= k;
            k <<= 1;
        }
        if (s > 0)
        {
            v[++cnt] = s * a;
            w[cnt] = s * b;
        }
    }
    for (int i = 1; i <= cnt; i++)
        for (int j = m; j >= v[i]; j--)
            f[j] = max(f[j], f[j - v[i]] + w[i]);
    cout << f[m] << endl;
    return 0;
}
```

四、AcWing 9. 分组背包问题

【题目描述】

有N组物品和一个容量是V的背包。

每组物品有若干个，同一组内的物品最多只能选一个。

每件物品的体积是v_{ij}，价值是w_{ij}，其中i是组号，j是组内编号。

求解将哪些物品装入背包，可使物品总体积不超过背包容量，且总价值最大。

输出最大价值。

【输入格式】

第一行有两个整数N, V，用空格隔开，分别表示物品组数和背包容量。

接下来有N组数据：

每组数据第一行有一个整数S_i，表示第i个物品组的物品数量；

每组数据接下来有S_i行，每行有两个整数v_{ij}, w_{ij}，用空格隔开，分别表示第i个物品组的第j个物品的体积和价值。

【输出格式】

输出一个整数，表示最大价值。

【数据范围】

$0 < N, V \leq 100$

$0 < S_i \leq 100$

$0 < v_{ij}, w_{ij} \leq 100$

【输入样例】

```
3 5
2
1 2
2 4
1
3 4
1
4 5
```

【输出样例】

```
8
```

【代码】

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

```

```cpp
const int N = 110;
int v[N][N], w[N][N], s[N];
int f[N];
int n, m;

int main()
{
    cin >> n >> m;
    for (int i = 1; i <= n; i++)
    {
        cin >> s[i];
        for (int j = 1; j <= s[i]; j++)
            cin >> v[i][j] >> w[i][j];
    }
    for (int i = 1; i <= n; i++)//枚举每一组物品
        for (int j = m; j >= 0; j--)//从大到小枚举所有体积
            for (int k = 1; k <= s[i]; k++)//枚举所有选择
                if (v[i][k] <= j)//第i组的第k件物品体积≤j时才会更新f[j]
                    f[j] = max(f[j], f[j - v[i][k]] + w[i][k]);
    cout << f[m] << endl;
    return 0;
}
```

动态规划-线性DP

一、AcWing 898. 数字三角形

【题目描述】

给定一个如下图所示的数字三角形，从顶部出发，在每一结点可以选择移动至其左下方的结点或移动至其右下方的结点，一直走到底层，要求找出一条路径，使路径上的数字的和最大。

```
        7
      3   8
    8   1   0
  2   7   4   4
4   5   2   6   5
```

【输入格式】

第一行包含整数n，表示数字三角形的层数。

接下来n行，每行包含若干整数，其中第i行表示数字三角形第i层包含的整数。

【输出格式】

输出一个整数，表示最大的路径数字和。

【数据范围】

$1 \leq n \leq 500$

$-10000 \leq$ 三角形中的整数 ≤ 10000

【输入样例】

```
5
7
3 8
8 1 0
2 7 4 4
4 5 2 6 5
```

【输出样例】

```
30
```

【代码】

```cpp
#include <iostream>
#include <cstring>
#include <algorithm>
using namespace std;

const int N = 510;
int a[N][N], f[N][N];//f[i][j]表示所有从起点走到(i,j)的最大值
int n;

int main()
{
    cin >> n;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= i; j++)
            cin >> a[i][j];
    memset(f, 0x8f, sizeof f);//初始化为负无穷
    f[1][1] = a[1][1];

    for (int i = 2; i <= n; i++)
```

```
        for (int j = 1; j <= i; j++)
            f[i][j] = max(f[i - 1][j - 1], f[i - 1][j]) + a[i][j];
    int res = -0x3f3f3f3f;
    for (int i = 1; i <= n; i++)//遍历最后一行找出最大值
        res = max(res, f[n][i]);
    cout << res << endl;
    return 0;
}
```

二、AcWing 895/896. 最长上升子序列 I/II

【题目描述】

给定一个长度为N的数列，求数值严格单调递增的子序列的长度最长是多少。

【输入格式】

第一行包含整数N。

第二行包含N个整数，表示完整序列。

【输出格式】

输出一个整数，表示最大长度。

【朴素版数据范围】

$1 \leq N \leq 1000$

$-10^9 \leq$ 数列中的数 $\leq 10^9$

【优化版数据范围】

$1 \leq N \leq 100000$

$-10^9 \leq$ 数列中的数 $\leq 10^9$

【输入样例】

```
7
3 1 2 1 8 5 6
```

【输出样例】

```
4
```

【朴素版代码$O(n^2)$】

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

const int N = 1010;
int a[N], f[N];//f[i]表示以第i个数结尾的LIS长度
int n, res;

int main()
{
    cin >> n;
    for (int i = 0; i < n; i++) cin >> a[i];
    for (int i = 0; i < n; i++)
    {
        f[i] = 1;//只有a[i]一个数的极端情况
        for (int j = 0; j < i; j++)
            if (a[j] < a[i]) f[i] = max(f[i], f[j] + 1);
        res = max(res, f[i]);
    }
    cout << res << endl;
    return 0;
}
```

【优化版代码$O(nlogn)$】

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

const int N = 100010;
int a[N], f[N];//f表示以最小的数结尾的LIS
int n, cnt;//cnt记录f的长度，即LIS的长度

int main()
{
    cin >> n;
    for (int i = 0; i < n; i++) cin >> a[i];
    f[cnt++] = a[0];
    for (int i = 1; i < n; i++)
    {
        //若a[i] > f[cnt - 1]则说明可使LIS长度+1，因此加入LIS中
        if (a[i] > f[cnt - 1]) f[cnt++] = a[i];
        else

        {
```

```
            //二分查找第一个大于等于a[i]的数，将其替换为a[i]
            int l = 0, r = cnt - 1;
            while (l < r)
            {
                int mid = l + r >> 1;
                if (f[mid] >= a[i]) r = mid;
                else l = mid + 1;
            }
            f[r] = a[i];
        }
    }
    cout << cnt << endl;
    return 0;
}
```

三、AcWing 897. 最长公共子序列

【题目描述】

给定两个长度分别为N和M的字符串A和B，求既是A的子序列又是B的子序列的字符串长度最长是多少。

【输入格式】

第一行包含两个整数N和M。

第二行包含一个长度为N的字符串，表示字符串A。

第三行包含一个长度为M的字符串，表示字符串B。

字符串均由小写字母构成。

【输出格式】

输出一个整数，表示最大长度。

【数据范围】

$1 \leq N, M \leq 1000$

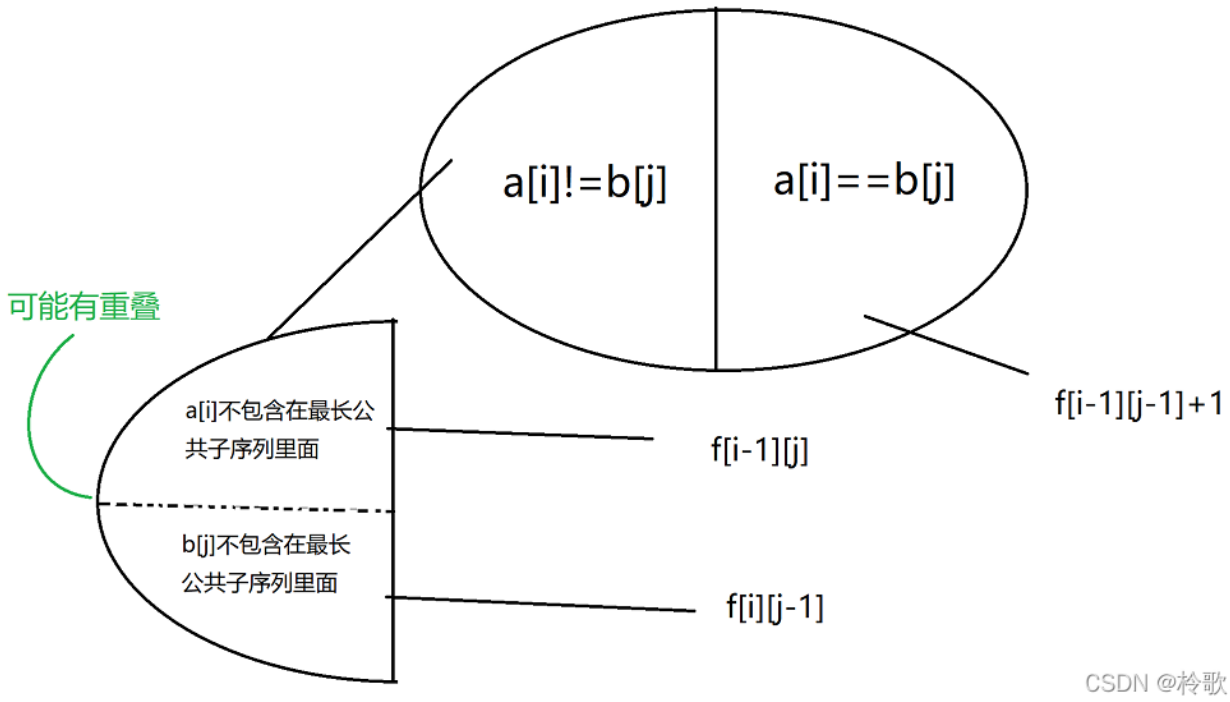
【输入样例】

```
4 5
acbd
abedc
```

【输出样例】

```
3
```

【分析】



【代码】

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

const int N = 1010;
char a[N], b[N];
int f[N][N];
int n, m;

int main()
{
    cin >> n >> m >> a + 1 >> b + 1;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
        {
            f[i][j] = max(f[i - 1][j], f[i][j - 1]);
            if (a[i] == b[j])
                f[i][j] = max(f[i][j], f[i - 1][j - 1] + 1);

        }
```

```
    cout << f[n][m] << endl;
    return 0;
}
```

四、AcWing 902. 最短编辑距离

【题目描述】

给定两个字符串A和B，现在要将A经过若干操作变为B，可进行的操作有：

- 删除：将字符串A中的某个字符删除。
- 插入：在字符串A的某个位置插入某个字符。
- 替换：将字符串A中的某个字符替换为另一个字符。

现在请你求出，将A变为B至少需要进行多少次操作。

【输入格式】

第一行包含整数n，表示字符串A的长度。

第二行包含一个长度为n的字符串A。

第三行包含整数m，表示字符串B的长度。

第四行包含一个长度为m的字符串B。

字符串中均只包含大写字母。

【输出格式】

输出一个整数，表示最少操作次数。

【数据范围】

$1 \leq n, m \leq 1000$

【输入样例】

```
10
AGTCTGACGC
11
AGTAAGTAGGC
```

【输出样例】

```
4
```

【分析】

```
若删除a[i]后与b的前j个字符匹配，则a的前i - 1个字符与b的前j个字符匹配
即f[i][j] = f[i - 1][j] + 1

若在a[i]后添加字符后与b的前j个字符匹配，则a的前i个字符与b的前j - 1个字符匹配
即f[i][j] = f[i][j - 1] + 1

若修改a[i]后与b的前j个字符匹配，则a的前i - 1个字符与b的前j - 1个字符匹配
即f[i][j] = f[i - 1][j - 1] + 1

若a[i]与b[j]相等则不需要修改就匹配，且a的前i - 1个字符与b的前j - 1个字符匹配
即f[i][j] = f[i - 1][j - 1]
```

【代码】

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

const int N = 1010;
char a[N], b[N];
int f[N][N];//f[i][j]表示把a的前i个字母变为b的前j个字母的最少操作次数
int n, m;

int main()
{
    cin >> n >> a + 1 >> m >> b + 1;
    //若a的前0个字母和b的前i个字母匹配，则需要增加字符，增加数量与b的长度有关
    //若a的前i个字母和b的前0个字母匹配，则需要删除字符，删除数量与a的长度有关
    for (int i = 0; i <= m; i++) f[0][i] = i;
    for (int i = 0; i <= n; i++) f[i][0] = i;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
        {
            f[i][j] = min(f[i - 1][j] + 1, f[i][j - 1] + 1);
            if (a[i] == b[j]) f[i][j] = min(f[i][j], f[i - 1][j - 1]);
            else f[i][j] = min(f[i][j], f[i - 1][j - 1] + 1);
        }
    cout << f[n][m] << endl;
    return 0;
}
```

五、AcWing 899. 编辑距离

【题目描述】

给定n个长度不超过10的字符串以及m次询问，每次询问给出一个字符串和一个操作次数上限。

对于每次询问，请你求出给定的n个字符串中有多少个字符串可以在上限操作次数内经过操作变成询问给出的字符串。

每个对字符串进行的单个字符的插入、删除或替换算作一次操作。

【输入格式】

第一行包含两个整数n和m。

接下来n行，每行包含一个字符串，表示给定的字符串。

再接下来m行，每行包含一个字符串和一个整数，表示一次询问。

字符串中只包含小写字母，且长度均不超过10。

【输出格式】

输出共m行，每行输出一个整数作为结果，表示一次询问中满足条件的字符串个数。

【数据范围】

$1 \leq n, m \leq 1000$

【输入样例】

```
3 2
abc
acd
bcd
ab 1
acbd 2
```

【输出样例】

```
1
3
```

【代码】

```
#include <iostream>
```

```cpp
#include <algorithm>
#include <cstring>
using namespace std;

const int N = 15, M = 1010;
char str[M][N];
int f[N][N];
int n, m;

int edit_distance(char a[], char b[])
{
    int la = strlen(a + 1), lb = strlen(b + 1);
    for (int i = 0; i <= lb; i++) f[0][i] = i;
    for (int i = 0; i <= la; i++) f[i][0] = i;
    for (int i = 1; i <= la; i++)
        for (int j = 1; j <= lb; j++)
        {
            f[i][j] = min(f[i - 1][j] + 1, f[i][j - 1] + 1);
            if (a[i] == b[j]) f[i][j] = min(f[i][j], f[i - 1][j - 1]);
            else f[i][j] = min(f[i][j], f[i - 1][j - 1] + 1);
        }
    return f[la][lb];
}

int main()
{
    cin >> n >> m;
    for (int i = 0; i < n; i++) cin >> str[i] + 1;
    while (m--)
    {
        char s[N];
        int limit, res = 0;
        cin >> s + 1 >> limit;
        for (int i = 0; i < n; i++)
            if (edit_distance(str[i], s) <= limit) res++;
        cout << res << endl;
    }
    return 0;
}
```

动态规划-区间DP

一、AcWing 282. 石子合并

【题目描述】

设有N堆石子排成一排，其编号为$1, 2, 3, \ldots, N$。

每堆石子有一定的质量，可以用一个整数来描述，现在要将这N堆石子合并成为一堆。

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。

例如有4堆石子分别为$1, 3, 5, 2$，我们可以先合并1、2堆，代价为4，得到$4, 5, 2$，又合并1、2堆，代价为9，得到$9, 2$，再合并得到11，总代价为$4 + 9 + 11 = 24$。

如果第二步是先合并2、3堆，则代价为7，得到$4, 7$，最后一次合并代价为11，总代价为$4 + 7 + 11 = 22$。

问题是：找出一种合理的方法，使总的代价最小，输出最小代价。

【输入格式】

第一行一个数N表示石子的堆数N。

第二行N个数，表示每堆石子的质量均（不超过1000）。

【输出格式】

输出一个整数，表示最小代价。

【数据范围】

$1 \leq N \leq 300$

【输入样例】

```
4
1 3 5 2
```

【输出样例】

```
22
```

【代码】

```
#include <iostream>
#include <algorithm>

using namespace std;
```

```cpp

const int N = 310;
int f[N][N];//f[i][j]表示将[i, j]合并成一堆的最小代价
int s[N];//前缀和
int n;

int main()
{
    cin >> n;
    for (int i = 1; i <= n; i++) cin >> s[i], s[i] += s[i - 1];
    for (int len = 2; len <= n; len++)
        for (int i = 1; i + len - 1 <= n; i++)
        {
            int j = i + len - 1;
            f[i][j] = 0x3f3f3f3f;
            for (int k = i; k < j; k++)//枚举分界点
                f[i][j] = min(f[i][j], f[i][k] + f[k + 1][j] + s[j] - s[i - 1]);
        }
    cout << f[1][n] << endl;
    return 0;
}
```

动态规划-计数类DP

一、AcWing 900. 整数划分

【题目描述】

一个正整数n可以表示成若干个正整数之和，形如：$n=n_1+n_2+\cdots+n_k$，其中$n_1\geq n_2\geq\cdots\geq n_k,k\geq 1$。

我们将这样的一种表示称为正整数n的一种划分。

现在给定一个正整数n，请你求出n共有多少种不同的划分方法。

【输入格式】

共一行，包含一个整数n。

【输出格式】

共一行，包含一个整数，表示总划分数量。

由于答案可能很大，输出结果请对10^9+7取模。

【数据范围】

$1 \leq n \leq 1000$

【输入样例】

```
5
```

【输出样例】

```
7
```

【分析】

（1）$f[i][j]$表示从$1 \sim i$中选且总体积恰好为j的方案（完全背包问题）

```
f[i][j] = f[i - 1][j] + f[i - 1][j - i] + ... + f[i - 1][j - k * i]
f[i][j - i] =           f[i - 1][j - i] + ... + f[i - 1][j - k * i]
因此f[i][j] = f[i - 1][j] + f[i][j - i]
```

（2）$f[i][j]$表示所有总和是i且恰好表示成j个数的和的方案

```
将状态集合分为最小值是1与最小值大于1两种状态
若最小值是1，则去掉1之后状态为总和是i - 1且恰好表示成j - 1个数的和，即f[i - 1][j - 1]
若最小值大于1，则将每个数减去1后总和将减去j，且仍是j个数的和，即f[i - j][j]
因此f[i][j] = f[i - 1][j - 1] + f[i - j][j]
ans = f[n][1] + f[n][2] + ... + f[n][n]
```

【思路一代码】

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

const int N = 1010, MOD = 1e9 + 7;
int f[N];
int n;

int main()
{
    cin >> n;
```

```
    f[0] = 1;
    for(int i = 1; i <= n; i++)
        for(int j = i; j <= n; j++)
            f[j] = (f[j] + f[j - i]) % MOD;
    cout << f[n] << endl;
    return 0;
}
```

【思路二代码】

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

const int N = 1010, MOD = 1e9 + 7;
int f[N][N];
int n;

int main()
{
    cin >> n;
    f[0][0] = 1;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= i; j++)//总和为i最多用i个数表示
            f[i][j] = (f[i - 1][j - 1] + f[i - j][j]) % MOD;
    int res = 0;
    for (int i = 1; i <= n; i++) res = (res + f[n][i]) % MOD;
    cout << res << endl;
    return 0;
}
```

动态规划-数位统计DP

一、AcWing 338. 计数问题

【题目描述】

给定两个整数a和b，求a和b之间的所有数字中$0 \sim 9$的出现次数。

例如，$a = 1024, b = 1032$，则a和b之间共有9个数如下：

```
1024 1025 1026 1027 1028 1029 1030 1031 1032
```

其中`0`出现10次，`1`出现10次，`2`出现7次，`3`出现3次等等...

【输入格式】

输入包含多组测试数据。

每组测试数据占一行，包含两个整数a和b。

当读入一行为`0 0`时，表示输入终止，且该行不作处理。

【输出格式】

每组数据输出一个结果，每个结果占一行。

每个结果包含十个用空格隔开的数字，第一个数字表示0出现的次数，第二个数字表示1出现的次数，以此类推。

【数据范围】

$0 < a, b < 10^8$

【输入样例】

```
1 10
44 497
346 542
1199 1748
1496 1403
1004 503
1714 190
1317 854
1976 494
1001 1960
0 0
```

【输出样例】

```
1 2 1 1 1 1 1 1 1
85 185 185 185 190 96 96 96 95 93
40 40 40 93 136 82 40 40 40 40
115 666 215 215 214 205 205 154 105 106
16 113 19 20 114 20 20 19 19 16
107 105 100 101 101 197 200 200 200 200
413 1133 503 503 503 502 502 417 402 412
196 512 186 104 87 93 97 97 142 196
398 1375 398 398 405 499 499 495 488 471
294 1256 296 296 296 296 287 286 286 247
```

【分析】

```
分别求出每一位为x时的次数，累加即为1至n中x出现的次数
n = abcdefg，求第4位出现x的次数
(1)000 ~ abc - 1, x ,000 ~ 999，次数为abc * 1000(若x为0，则枚举001 ~ abc - 1)
(2)abc, x, ???
    (2.1)x > d，则次数为0
    (2.2)x == d，则后面可以为000 ~ efg，次数为efg + 1
    (2.3)x < d，则后面可以为000 ~ 999，次数为1000
```

代码

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

//求l位至r位构成的数是多少
int getNum(vector<int> num, int l, int r)
{
    int res = 0;
    for (int i = l; i >= r; i--)
        res = res * 10 + num[i];
    return res;
}

//统计1~n中x出现的次数
int count(int n, int x)
```

```
{
    if (!n) return 0;
    vector<int> num;
    while (n)//把n的每一位提出来
    {
        num.push_back(n % 10);
        n /= 10;
    }
    n = num.size();//n赋值为长度
    int res = 0;
    for (int i = n - 1; i >= 0; i--)//从高位开始枚举，计算第i位为x的次数
    {
        //最高位不能为0，因此若x为0则从第二位开始
        if (!x && i == n - 1) continue;
        //i在最高位时不会出现情况(1)
        if (i < n - 1)
        {
            res += getNum(num, n - 1, i + 1) * pow(10, i);
            //若x为0则前面不能全为0，因此少了10^i次
            if (!x) res -= pow(10, i);
        }
        //情况(2.2)
        if (num[i] == x) res += getNum(num, i - 1, 0) + 1;
        //情况(2.3)
        else if (num[i] > x) res += pow(10, i);
    }
    return res;
}

int main()
{
    int a, b;
    while (cin >> a >> b, a || b)
    {
        if (a > b) swap(a, b);
        for (int i = 0; i < 10; i++)
            cout << count(b, i) - count(a - 1, i) << ' ';
        cout << endl;
    }
    return 0;
}
```

动态规划-状态压缩DP

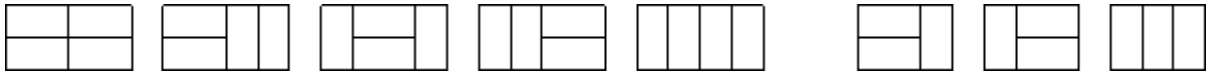
一、AcWing 291. 蒙德里安的梦想

【题目描述】

求把$N \times M$的棋盘分割成若干个1×2的长方形，有多少种方案。

例如当$N = 2, M = 4$时，共有5种方案。当$N = 2, M = 3$时，共有3种方案。

如下图所示：



【输入格式】

输入包含多组测试用例。

每组测试用例占一行，包含两个整数N和M。

当输入用例$N = 0, M = 0$时，表示输入终止，且该用例无需处理。

【输出格式】

每个测试用例输出一个结果，每个结果占一行。

【数据范围】

$1 \leq N, M \leq 11$

【输入样例】

```
1 2
1 3
1 4
2 2
2 3
2 4
2 11
4 11
0 0
```

【输出样例】

```
1
0
1
2
3
5
144
51205
```

【分析】

摆放方块的时候，**先放横着的，再放竖着的**。总方案数等于只放横着的小方块的**合法**方案数。

如何判断当前方案数是否合法？若**所有剩余位置能填充满竖着的小方块**则该方案合法。可以按列来看，**每一列内部所有连续的空着的小方块需要是偶数个**。因此可以预处理用`st[M]`标记某一列中的某种状态是否有奇数个连续0，若有奇数个0则状态不合法，`st[state] = false`，否则为`true`。

这是一道**动态规划**的题目，并且是一道**状态压缩的DP**：用一个N位的二进制数，每一位表示一个物品，`1/0`分别表示该位置上`是/否`有物品。因此可以用$0 \sim 2^N - 1$（N位二进制数对应的十进制数）中的所有数来枚举全部的状态。

状态表示：$f[i][j]$表示已经将前$i-1$列摆好，且从第$i-1$列，伸出到第i列的状态是j的所有方案。

状态转移：既然第i列固定了，我们需要看第$i-2$列是怎么转移到到第$i-1$列的（看最后转移过来的状态）。假设此时对应的状态是k（第$i-2$列到第$i-1$列伸出来的二进制数，比如00100），k也是一个二进制数，1表示哪几行小方块是横着伸出来的，0表示哪几行不是横着伸出来的。它对应的方案数是`f[i - 1, k]`，即前$i-2$列都已摆完，且从第$i-2$列伸到第$i-1$列的状态为k的所有方案数。

这个k需要满足什么条件呢？

首先：**k不能和j在同一行**，因为从$i-1$列到第i列是横着摆放的1×2的方块，那么$i-2$列到$i-1$列就不能是横着摆放的，否则就是1×3的方块了，这与题意矛盾。所以k和j不能位于同一行。

既然不能同一行伸出来，那么对应的代码为`(k & j) == 0`，表示两个数相与，如果有一位都是1结果就不是0，`(k & j) == 0`表示k和j没有一位是同时为1的，即没有一行有冲突。

既然从第$i-1$列到第i列横着摆的，和第$i-2$列到第$i-1$列横着摆的都确定了，那么第$i-1$列空着的格子就确定了，这些空着的格子将来用作**竖着放**。如果某一列有这些空着的位置，**那么该列所有连续的空着的位置长度必须是偶数**，即真正的可行情况为`(k & j) == 0 && st[j | k] == true`（由于状态k和状态j都与第$i-1$列有关，因此该列的状态为两种状态的并集，即`j | k`）。

总共m列，我们假设列下标从0开始，即`第0列，第1列，...，第m-1列`。根据状态表示$f[i][j]$的定义，我们答案是什么呢？返回定义处思考一下，答案是`f[m][0]`，意思是前$m-1$列全部摆好，且从第$m-1$列到m列状态是0（即从第$m-1$列到第m列没有伸出来的）的所有方案，即整个棋盘全部摆好的方案。

【代码】

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

typedef long long LL;
const int N = 12, M = 1 << N;//M表示一列中的所有状态数，最多为2^N - 1个
LL f[N][M];//f[i][j]表示已经将前i-1列摆好，且从第i-1列伸出到第i列的状态是j的所有方案
bool st[M];//标记一列中的某种状态是否有奇数个连续0，若有奇数个0则状态不合法
vector<int> state[M];//真正可行的状态，即第i列的状态j与第i-1列的状态k既没有前后两列伸进伸出的冲突，又没有连续奇数个0
int n, m;

int main()
{
    while (cin >> n >> m, n || m)
    {
        //第一部分初始化，预处理一列中的合法状态
        for (int i = 0; i < 1 << n; i++)
        {
            int cnt = 0;//记录连续0的个数
            bool isValid = true;//是否合法
            //由于有n行，所以一列的状态二进制数有n位，从下到上枚举每一位
            for (int j = 0; j < n; j++)
            {
                if ((i >> j) & 1)//如果该状态的第j位为1

                {
```

```
                    if (cnt & 1)//如果前面的连续0为奇数个
                    {
                        isValid = false;
                        break;
                    }
                    cnt = 0;//清空计数器
                }
                else cnt++;
            }
            if (cnt & 1) isValid = false;//判断最后一段的连续0个数
            st[i] = isValid;
        }

        //第二部分初始化，预处理真正可行的状态
        for (int j = 0; j < 1 << n; j++)//枚举第i列的状态
        {
            state[j].clear();//多组输入，因此每次需要清空之前的内容
            for (int k = 0; k < 1 << n; k++)//枚举第i-1列的状态
            {
                //如果状态j和k在同一行上没有冲突且合并后的这一列上0的个数是合法的
                if ((j & k) == 0 && st[j | k])
                    state[j].push_back(k);
            }
        }

        //动态规划部分
        memset(f, 0, sizeof f);//多组输入，清空内容
        f[0][0] = 1;//全摆放竖着的小方块，方案数为1
        for (int i = 1; i <= m; i++)
            for (int j = 0; j < 1 << n; j++)//遍历当前列(第i列)所有状态j
                for (auto k : state[j])//遍历第i-1列的状态k，如果真正可行，就转移
                    f[i][j] += f[i - 1][k];
        cout << f[m][0] << endl;//表示前m-1列都处理完，并且第m-1列没有伸出至m列的所有方案数
    }
    return 0;
}
```

二、AcWing 91. 最短Hamilton路径

【题目描述】

给定一张n个点的带权无向图，点从$0 \sim n-1$标号，求起点0到终点$n-1$的最短$Hamilton$路径。$Hamilton$路径的定义是从0到$n-1$不重不漏地经过每个点恰好一次。

【输入格式】

第一行输入整数n。

接下来n行每行n个整数，其中第i行第j个整数表示点i到j的距离（记为$a[i,j]$）。

对于任意的x,y,z，数据保证$a[x,x]=0,a[x,y]=a[y,x]$并且$a[x,y]+a[y,z] \geq a[x,z]$。

【输出格式】

输出一个整数，表示最短$Hamilton$路径的长度。

【数据范围】

$1 \leq n \leq 20$

$0 \leq a[i,j] \leq 10^7$

【输入样例】

```
5
0 2 4 5 1
2 0 6 5 3
4 6 0 8 3
5 5 8 0 5
1 3 3 5 0
```

【输出样例】

```
18
```

【分析】

首先想下暴力算法，这里直接给出一个例子。

比如数据有5个点，分别是$0,1,2,3,4$，那么在爆搜的时候，会枚举以下路径情况（只算对答案有贡献的情况的话）：

- $case1:0 \to 1 \to 2 \to 3 \to 4$
- $case2:0 \to 1 \to 3 \to 2 \to 4$
- $case3:0 \to 2 \to 1 \to 3 \to 4$
- $case4:0 \to 2 \to 3 \to 1 \to 4$
- $case5:0 \to 3 \to 1 \to 2 \to 4$

- $case6: 0 \rightarrow 3 \rightarrow 2 \rightarrow 1 \rightarrow 4$

那么观察一下$case1$和$case3$，可以发现，我们在计算从点0到点3的路径时，其实并不关心这两中路径经过的点的顺序，而是只需要这两种路径中的较小值，因为只有较小值可能对答案有贡献。

所以，我们在枚举路径的时候，只需要记录两个属性：**当前经过的点集，当前到了哪个点**。

而当前经过的点集不是一个数。观察到数据中点数不会超过20，我们可以用一个二进制数表示当前经过的点集。其中第i位为`1/0`表示`是/否`经过了点i。

- 状态表示：$f[state][j]$。其中`state`是一个二进制数，表示点集的方法如上述所示。
- 集合：经过的点集为`state`，且当前到了点j上的所有路径。
- 属性：路径总长度的最小值。
- 状态计算：假设当前要从点k转移到j。那么根据$Hamilton$路径的定义，**走到点k的路径就不能经过点j**，所以就可以推出状态转移方程：`f[state][j] = min{ f[state ^ (1 << j)][k] + g[k][j] }`。

其中`g[k][j]`表示从点k到点j的距离，`^`表示异或运算。

`state ^ (1 << j)`是将`state`的第j位改变后的值，即：

- 如果$state$的第j位是1那么将其改为0；
- 否则将$state$的第j位改为1。

由于到达点j的路径一定经过点j，也就是说当$state$的第j位为1的时候，`f[state][j]`才可以被转移，所以`state ^ (1 << j)`其实就是将$state$的第j位改为0，这样也就符合了走到点k的路径就不能经过点j这个条件，这样就可以从点k转移到点j。

所有状态转移完后，根据`f[state][j]`的定义，要输出`f[111…11(n个1)][n-1]`。

那么怎么构造n个1呢，可以直接通过`1 << n`求出`100…0(n个0)`，然后减一即可。

时间复杂度：

枚举所有$state$的时间复杂度是$O(2^n)$

枚举j的时间复杂读是$O(n)$

枚举k的时间复杂度是$O(n)$

所以总的时间复杂度是$O(n^2 * 2^n)$

【代码】

```cpp
#include <iostream>
#include <cstring>
#include <algorithm>
using namespace std;

const int N = 20, M = 1 << N;
int g[N][N], f[M][N];//f[state][j]表示经过的点集为state，且当前到达点j的所有路径
int n;

int main()
{
    cin >> n;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            cin >> g[i][j];
    memset(f, 0x3f, sizeof f);
    f[1][0] = 0;//初始化点集中只有起点0，且到起点的距离为0
    for (int state = 1; state < 1 << n; state++)//枚举所有状态
        if (state & 1)//如果状态中包含起点(可以不加此判断，判断后可优化运行时间)
            for (int j = 0; j < n; j++)//枚举该状态所到达的所有的点
                if (state >> j & 1)//如果该点在状态中(第j位为1)，那么进行状态转移
                    for (int k = 0; k < n; k++)//枚举所有从点k转移到点j的情况
                        if (state ^ (1 << j) >> k & 1)//如果从当前状态中去掉点j后，仍然包含点k，那么才能从点k转移到点j
                            f[state][j] = min(f[state][j], f[state ^ (1 << j)][k] + g[k][j]);
    cout << f[(1 << n) - 1][n - 1] << endl;
    return 0;
}
```

动态规划-树形DP

一、AcWing 285. 没有上司的舞会

【题目描述】

Ural大学有N名职员，编号为$1 \sim N$。

他们的关系就像一棵以校长为根的树，父节点就是子节点的直接上司。

每个职员有一个快乐指数，用整数H_i给出，其中$1 \leq i \leq N$。

现在要召开一场周年庆宴会，不过，没有职员愿意和直接上司一起参会。

在满足这个条件的前提下，主办方希望邀请一部分职员参会，使得所有参会职员的快乐指数总和最大，求这个最大值。

【输入格式】

第一行一个整数N。

接下来N行，第i行表示i号职员的快乐指数H_i。

接下来$N-1$行，每行输入一对整数L, K，表示K是L的直接上司。

【输出格式】

输出最大的快乐指数。

【数据范围】

$1 \leq N \leq 6000$

$-128 \leq H_i \leq 127$

【输入样例】

```
7
1
1
1
1
1
1
1
1 3
2 3
6 4
7 4
4 5
3 5
```

【输出样例】

```
5
```

【代码】

```cpp
#include <iostream>
#include <cstring>
#include <algorithm>
using namespace std;

const int N = 6010;
int f[N][2];//f[i][0]表示不选择结点i，f[i][1]表示选择结点i
int e[N], ne[N], h[N], w[N], idx;
bool st[N];//标记是否有上司
int n;

void add(int u, int v)
{
    e[idx] = v, ne[idx] = h[u], h[u] = idx++;
}

void dfs(int u)
{
    f[u][1] = w[u];//1代表这个boss要来，先加上他来的利益
    for (int i = h[u]; ~i; i = ne[i])
    {
        int j = e[i];
        dfs(j);//递归一直搜
        f[u][0] += max(f[j][0], f[j][1]);//boss不来那小弟就是王了，但要以利益为重，如果小小弟来获利更大，就让小小弟来
        f[u][1] += f[j][0];//boss来了！小弟都承让
    }
}

int main()
{
    cin >> n;
    for (int i = 1; i <= n; i++) cin >> w[i];
    memset(h, -1, sizeof h);
    for (int i = 1; i < n; i++)
    {
        int a, b;
        cin >> a >> b;
        add(b, a);//b是a的直接上司，即b->a
```

```
        st[a] = true;
    }
    int root = 1;
    while (st[root]) root++;//找出根结点
    dfs(root);
    cout << max(f[root][0], f[root][1]) << endl;
    return 0;
}
```

动态规划-记忆化搜索

一、AcWing 901. 滑雪

【题目描述】

给定一个R行C列的矩阵，表示一个矩形网格滑雪场。

矩阵中第i行第j列的点表示滑雪场的第i行第j列区域的高度。

一个人从滑雪场中的某个区域内出发，每次可以向上下左右任意一个方向滑动一个单位距离。

当然，一个人能够滑动到某相邻区域的前提是**该区域的高度低于自己目前所在区域的高度**。

下面给出一个矩阵作为例子：

```
 1  2  3  4 5
16 17 18 19 6
15 24 25 20 7
14 23 22 21 8
13 12 11 10 9
```

在给定矩阵中，一条可行的滑行轨迹为`24-17-2-1`。

在给定矩阵中，最长的滑行轨迹为`25-24-23-...-3-2-1`，沿途共经过**25**个区域。

现在给定你一个二维矩阵表示滑雪场各区域的高度，请你找出在该滑雪场中能够完成的最长滑雪轨迹，并输出其长度（可经过最大区域数）。

【输入格式】

第一行包含两个整数R和C。

接下来R行，每行包含C个整数，表示完整的二维矩阵。

【输出格式】

输出一个整数，表示可完成的最长滑雪长度。

【数据范围】

$1 \leq R, C \leq 300$

$0 \leq$ 矩阵中整数 ≤ 10000

【输入样例】

```
5 5
1 2 3 4 5
16 17 18 19 6
15 24 25 20 7
14 23 22 21 8
13 12 11 10 9
```

【输出样例】

```
25
```

【代码】

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

const int N = 310;
int h[N][N], f[N][N];
int n, m, res;
int dx[4] = { 0, 1, 0, -1 }, dy[4] = { 1, 0, -1, 0 };

int dfs(int x, int y)
{
    if (f[x][y]) return f[x][y];//若该点已被计算过则直接返回
    f[x][y] = 1;//否则假设该点为最低点，滑雪长度为1
    for (int i = 0; i < 4; i++)
    {
        int nx = x + dx[i], ny = y + dy[i];
        //若(nx, ny)在界内且高度低于当前点说明可以滑下去
        if (nx >= 0 && nx < n && ny >= 0 && ny < m && h[nx][ny] < h[x][y])
            f[x][y] = max(f[x][y], dfs(nx, ny) + 1);
```

```
    }
    return f[x][y];
}

int main()
{
    cin >> n >> m;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            cin >> h[i][j];
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            res = max(res, dfs(i, j));
    cout << res << endl;
    return 0;
}
```