计数类动态规划

信智学院　陈琰宏

计数类动态规划的特点

DP不仅广泛用于各种最优化问题，也常常用于排列组合的个数、概率期望计算等等，因为这些问题往往具有很好的“ 重叠子问题”特性，这些问题往往都起源于排列组合中的组合公式A(m, n)=A(m-1, n-1)+A(m-1, n)

$$C_m^n = C_{m-1}^{n-1} + C_{m-1}^n$$

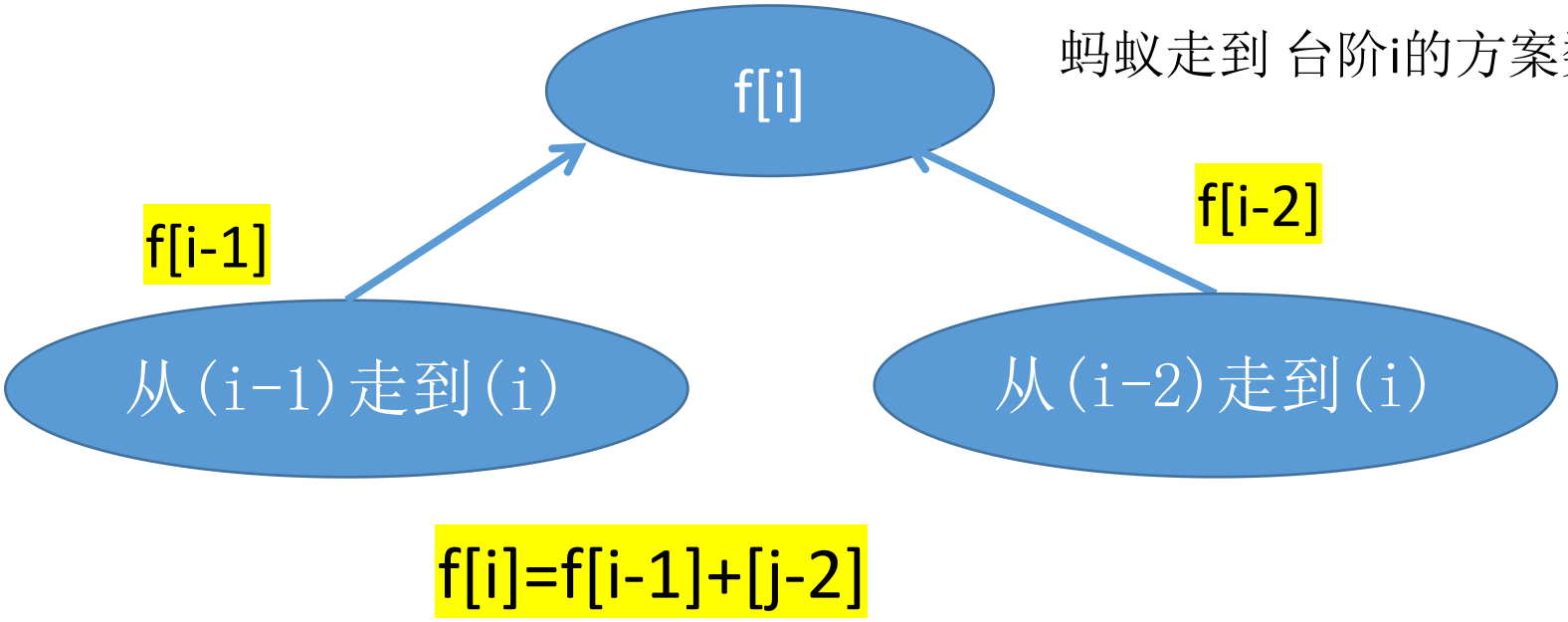
Fibo

楼梯有n个台阶，上楼可以一步上一阶，也可以一步上两阶。一共有多少种上楼的方法？

5	6
8	13

思考

1.状态如何描述？
2. 状态如何计算？
3. 目标状态是什么？
4. 初始条件时什么？

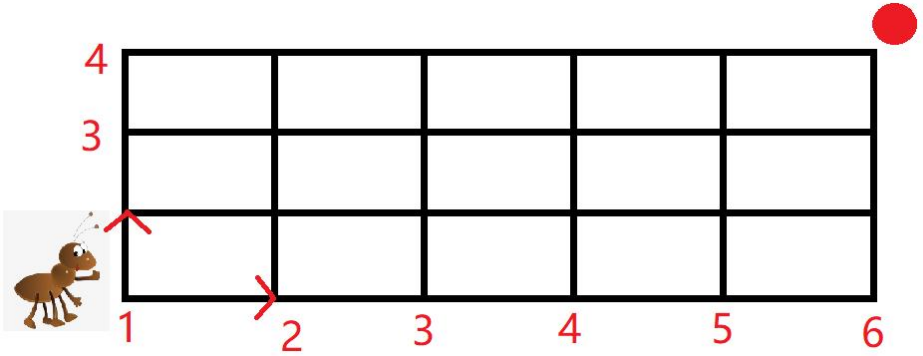


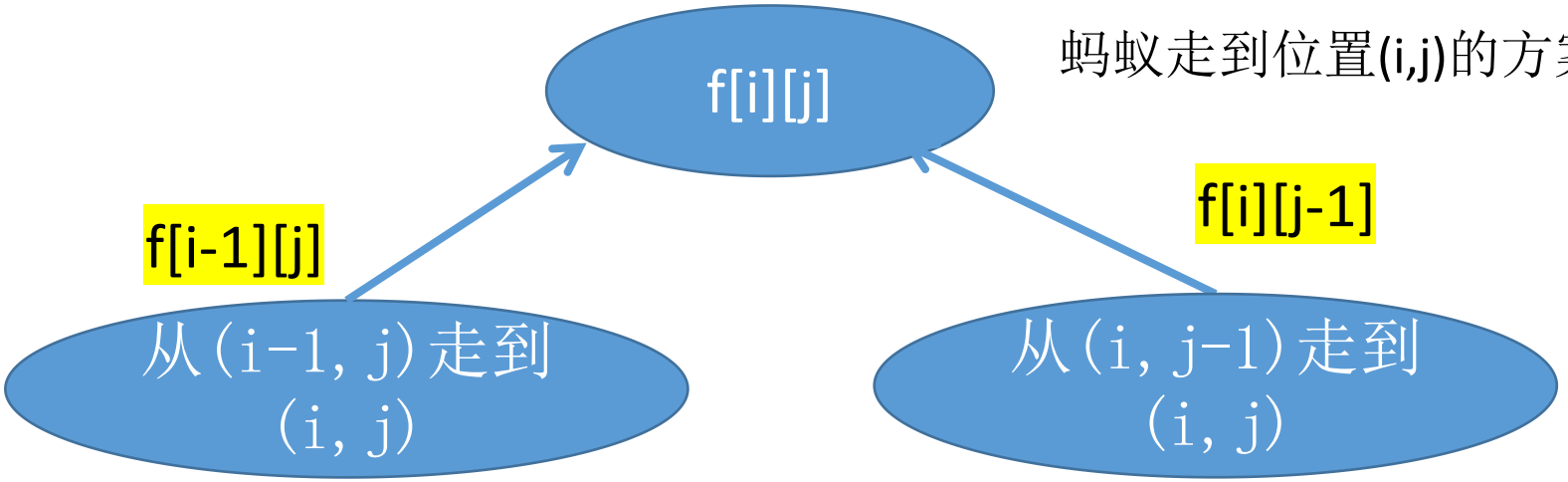
[2757]移动路线

蚂蚁共有3种移动路线：
路线1：(1,1) → (1,2) → (1,3) → (2,3)
路线2：(1,1) → (1,2) → (2,2) → (2,3)
路线3：(1,1) → (2,1) → (2,2) → (2,3)
输入
输入只有一行，包括两个整数m和n（$0 < m+n \le 20$），代表方格矩阵的行数和列数，m、n之间用空格隔开。
输出
输出只有一行，为不同的移动路线的数目。
样例输入
2 3
样例输出
3

思考

1.状态如何描述？
2. 状态如何计算？
3. 目标状态是什么？
4. 初始条件时什么？





$$f[i][j]=f[i-1][j]+f[i][j-1]$$

思考

1.状态如何描述？

f[i][j]表示蚂蚁走到位置(i,j)的方案数

2. 状态如何计算？

f[i][j]=f[i-1][j]+f[i][j-1]

3. 目标状态是什么？

f[m][n]

4. 初始条件时什么？

f[i][1]=1

f[1][i]=1

代码

```
1  #include<cstdio>
2  #include<cstring>
3  #include<iostream>
4  using namespace std;
5  int a[30][30];
6  int main(){
7      int m,n,i,j;
8      scanf("%d%d",&m,&n);
9      for(i=1;i<=m;i++)a[i][1]=1;
10     for(j=1;j<=n;j++)a[1][j]=1;
11     for(i=2;i<=m;i++)
12         for(j=2;j<=n;j++)
13             a[i][j]=a[i-1][j]+a[i][j-1];
14     printf("%d",a[m][n]);
15     return 0;
16 }
```

[2954] 整数划分

一个正整数 n 可以表示成若干个正整数之和，形如：n=n1+n2+…+nk，其中 n1≥n2≥…≥nk,k≥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。 由于答案可能很大，输出结果请对 10^9+7 取模。

5	500
7	168879716

思路

不考虑顺序，要求单调递减，5可以分解为：

5=5

5=4+1

5=3+2

5=3+1+1

5=2+2+1

5=2+1+1+1

5=1+1+1+1+1

```
int a[105],n,sum;
void  dfs(int k,int s){
    if(s>n)return;
    if(s==n){
    /*  for(int i=1;i<k;i++)
            cout<<a[i]<<" ";
        cout<<"\n";*/
        sum++;
        return;
    }

    for(int i=n;i>=1;i--){
        if(i<=a[k-1] && i+s<=n){
            a[k]=i;
            dfs(k+1,s+i);
            a[k]=0;
        }
    }
}
```

```
int main(){
    cin>>n;
    a[0]=n;
    dfs(1,0);
    cout<<sum;
    return 0;
}
```

思路

用背包的思想考虑问题：

把n看做背包容量，1,2,3, … n分别看做n个物体的体积，这n个物体均无使用次数限制。问题可以转化为：问恰好能装满总体积为n的背包的总方案数。

如此问题可转化为完全背包问题。

思路：状态描述

f[i][j]　表示前i个整数（1,2…,i）恰好拼成j的方案数

目标状态：f[n][n]表示前n个整数恰好拼成n的方案数

f[i][j] 可以有哪些状态转移过来呢？

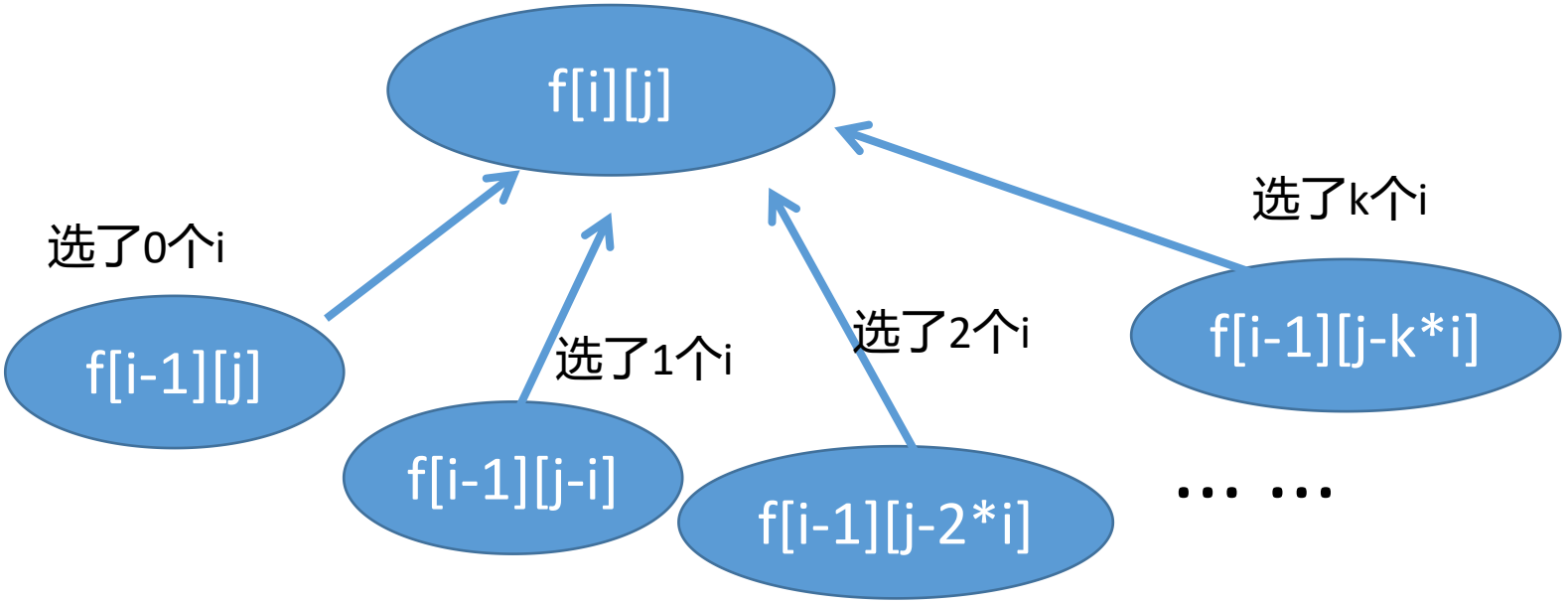
楼梯有n个台阶，上楼可以一步上一阶，也可以一步上两阶。一共有多少种上楼的方法？

$$f(n)=f(n-1)+f(n-2)$$

思路：状态转移

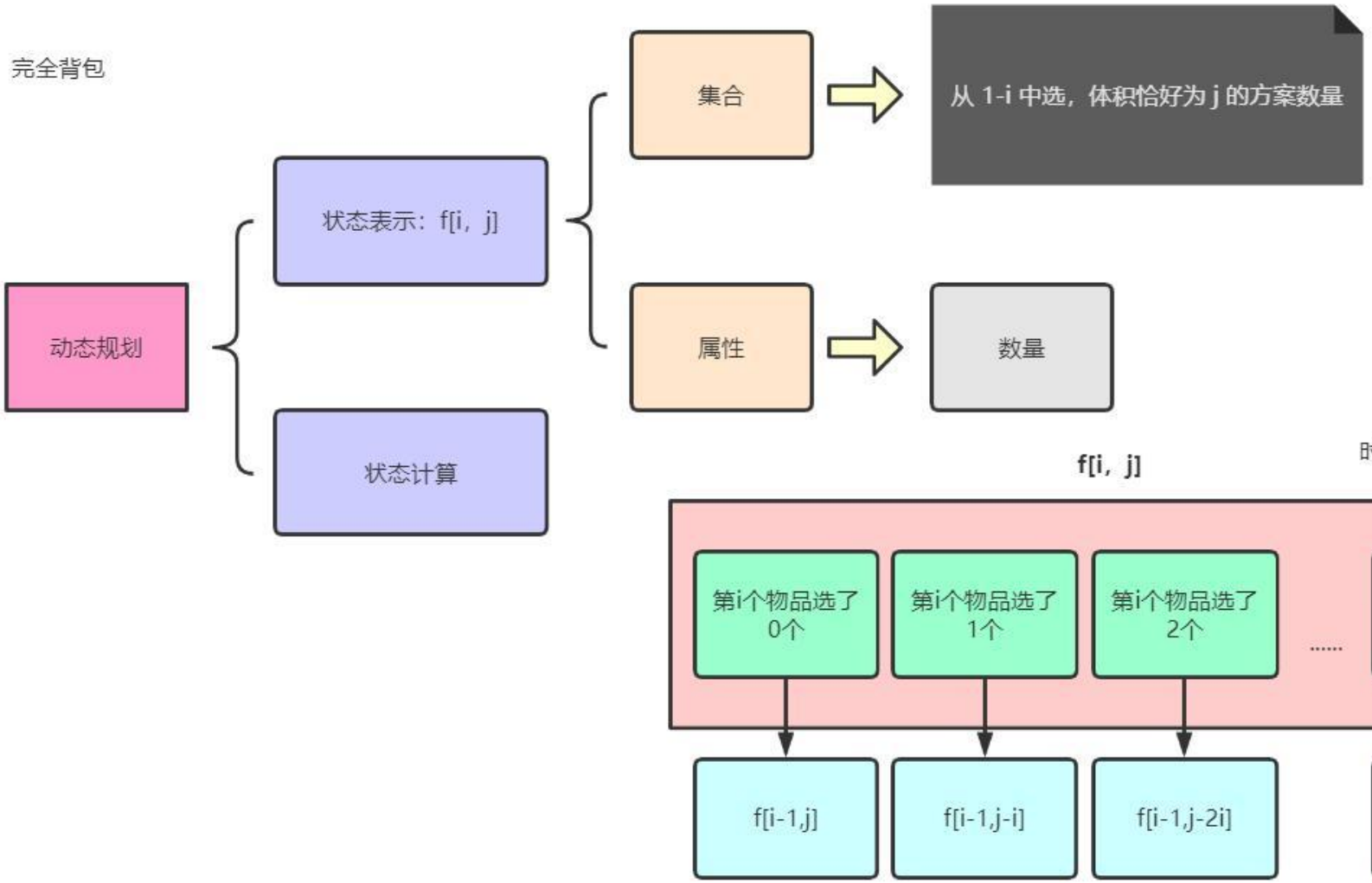
f[i][j] 表示前i个整数（1,2…,i）恰好拼成j的方案数

f[i][j] 可以有哪些状态转移过来呢？



f[i][j]=f[i-1][j]+f[i-1][j-i]+f[i-1][j-2*i]+ ...;

完全背包


动态规划
状态表示：f[i，j]
状态计算
集合
属性
从 1-i 中选，体积恰好为 j 的方案数量
数量
f[i，j]
时间复杂度O(n³)
第i个物品选了0个
第i个物品选了1个
第i个物品选了2个
.....
第i个物品选了s个
f[i-1,j]
f[i-1,j-i]
f[i-1,j-2i]
f[i-1,j-si]

复杂度O(n³)写法：

```
#include <iostream>
using namespace std;
const int N=1e3+10,mod = 1e9+7;

//f[i][j]从前i个数中选，总和为j的划分方法
int f[N][N];
int main(){
    int n;scanf("%d",&n);
    //前i个数总和是0：f[i][0] = 1 (即只有1中方案，就是1个都不选)
    for(int i=0;i<=n;i++) f[i][0]=1;
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=n;j++)
        {
            for(int k=0;k*i <= j;k++)
                f[i][j]=(f[i][j]+f[i-1][j - k*i])%mod;
        }
    }
    printf("%d",f[n][n]);
    return 0;
}
```

初始化说明：

f[0][0]=1表示从0中选(一个数都不选),体积恰好为0的方案数为1。一个数不选,体积自然就是0,方案是存在的。

f[0][1]表示从0中选(一个数都不选)，体积恰好为1的方案数为0 ，所以除了f[0][0]=1，其他都初始化为0

体积小于等于j的方案数：

f[0][j]表示0中选(一个数都不选)，体积不超过j的方案数 应该全部初始化成0

思路：优化状态转移

f[i][j] 表示前i个整数（1,2…,i）恰好拼成j的方案数，把集合选0个i，1个i，2个i，…全部加起来，即可求出方案数。

```
f[i][j]=f[i-1][j]+f[i-1][j-i]+f[i-1][j-2*i]+ ...;
```

同时：

```
f[i][j-i] =      f[i-1][j-i]+f[i-1][j- 2*i]+ ...;
```

因此：

```
f[i][j]=f[i-1][j]+f[i][j-i];
```

思路：初始状态

求最大值时，当都不选时，价值显然是 0

而求方案数时，当都不选时，方案数是 1（即前 i 个物品都不选的情况也是一种方案），所以需要初始化为 1

即：

```
for (int i = 0; i <= n; i ++)
    f[i][0] = 1;
```

复杂度O(n²)写法：

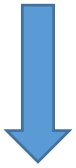
```cpp
#include <iostream>
using namespace std;
const int N = 1e3 + 7, mod = 1e9 + 7;
int f[N][N];
int main() {
    int n;
    cin >> n;

    for (int i = 0; i <= n; i ++) {
        f[i][0] = 1; // 容量为0时，前 i 个物品全不选也是一种方案
    }

    for (int i = 1; i <= n; i ++) {
        for (int j = 0; j <= n; j ++) {
            f[i][j] = f[i - 1][j] % mod; // 特殊 f[0][0] = 1
            if (j >= i) f[i][j] = (f[i - 1][j] + f[i][j - i]) % mod;
        }
    }
    cout << f[n][n] << endl;
}
```

背包优化：二维变一维

$$f[i][j] = f[i-1][j] + f[i][j-i]$$

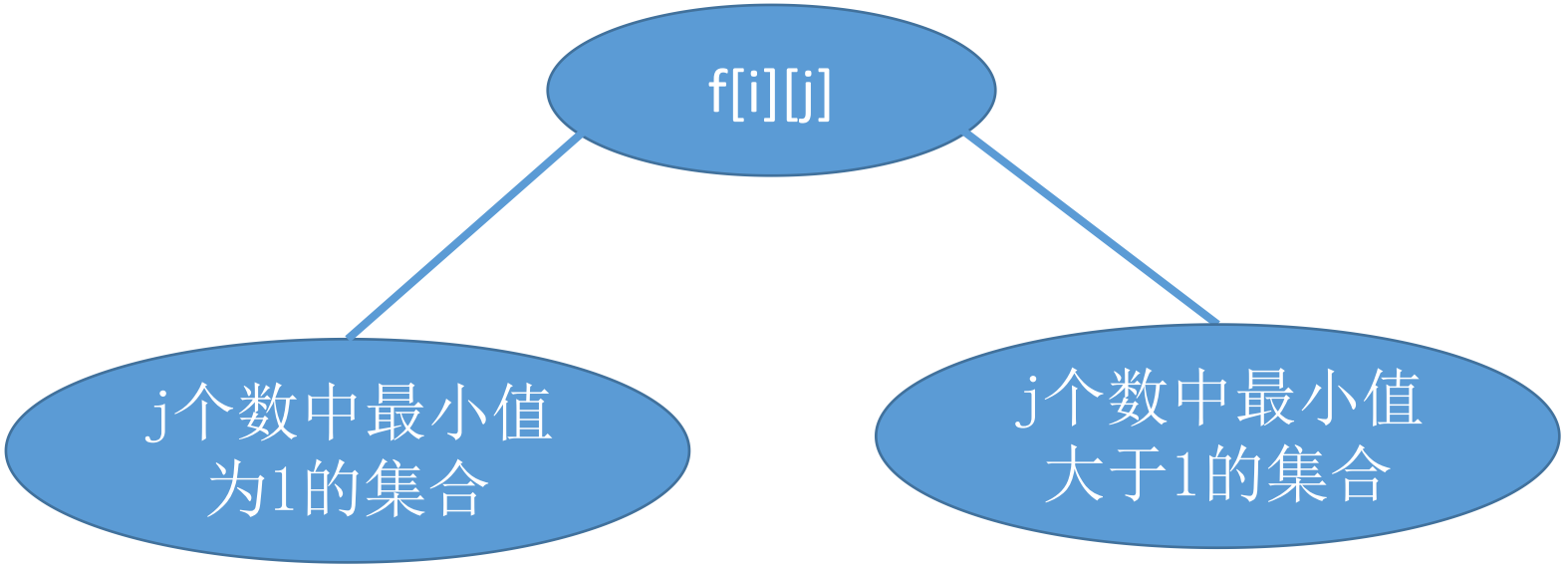


$$f[j] = f[j] + f[j-i]$$

```
 9  int main() {
10      int n;
11      cin >> n;
12      f[0] = 1; // 容量为0时,前 i 个物品全不选也是一种方案
13      for (int i = 1; i <= n; i ++) {
14          for (int j = i; j <= n; j ++) {
15              f[j] = (f[j] + f[j - i]) % mod;
16          }//完全背包的写法
17      }
18      cout << f[n] << endl;
19  }
```

另一种思路

f[i][j] 表示使用j个数能够凑成总和为i的方案(所有总和是 i，并且恰好表示成j个数的和的方案数)



去掉一个1，则状态变为f(i-1, j-1)

将j个数都减去一个1，则总和就减去j，但个数没变；状态变为f(i-j, j)

$$f(i, j)=f(i-1, j-1)+f(i-j, j)$$

另一种思路

状态计算详解：

- 使用j个数能够凑成i，这j个数中最小值为1的集合，去掉一个1，则状态变为：f(i, j)=f(i-1, j-1)
- 使用j个数能够凑成i，这j个数中最小值不为1的集合，将这j个数都减去一个1，那个总和就减去j，但是个数没变；则状态变为：f(i, j)=f(i-j, j)，此时状态表示为j个数能够凑成i-j的所有方案的数量。

将两个数量相加

$$f(i, j)=f(i-1, j-1)+f(i-j, j)$$

总方案数ans=f(n, 1)+f(n, 2)+...+f(n, n) 把n分成1个数的和，2个数的和,...， n个数的和

非背包DP写法：

```
 8  const int N = 1010, mod = 1e9 + 7;
 9  int n;
10  int f[N][N];
11  int main()
12  {
13      cin >> n;
14      f[0][0] = 1;
15      for (int i = 1; i <= n; i ++ )
16          //i最多表示成i个数的和，因此j<=i
17          for (int j = 1; j <= i; j ++ )
18              f[i][j] = (f[i - 1][j - 1] + f[i - j][j]) % mod;
19      int res = 0;
20      for (int i = 1; i <= n; i ++ ) res = (res + f[n][i]) % mod;
21      cout << res << endl;
22      return 0;
23  }
```

记忆化搜索

```
#include <iostream>
#include <algorithm>
using namespace std;
const int N = 1010, M = 1e9 + 7;
int n, f[N][N];//f[u][m]表示总和是u，最大的数n1为m，属性为数量
int dp(int u, int m)
{
    if (f[u][m]) return f[u][m];
    //若u和m相等，则至少有一种方法，即 只有本身一个数
    if (u == m) {
        f[u][m] = 1;
        for (int i = m - 1; i >= 1; i -- ) {
            f[u][m] = (f[u][m] + dp(u - i, min(i, u - i))) % M;
        }
    }
    else {
        for (int i = m; i >= 1; i -- ) {
            f[u][m] = (f[u][m] + dp(u - i, min(i, u - i))) % M;
        }
    }
    return f[u][m];
}
```

[2825] 货币系统

给你一个n种面值的货币系统，求组成面值为m的货币有多少种方案。

【输入样例】

```
3  10          //3种面值组成面值为10的方案
1              //面值1
2              //面值2
5              //面值5
```

【输出样例】

```
10             //有10种方案
```

思考

1. 状态如何描述？

f[j]表示组成面值为j的方案数

2. 状态如何计算？

3. 目标状态是什么？

f[m]

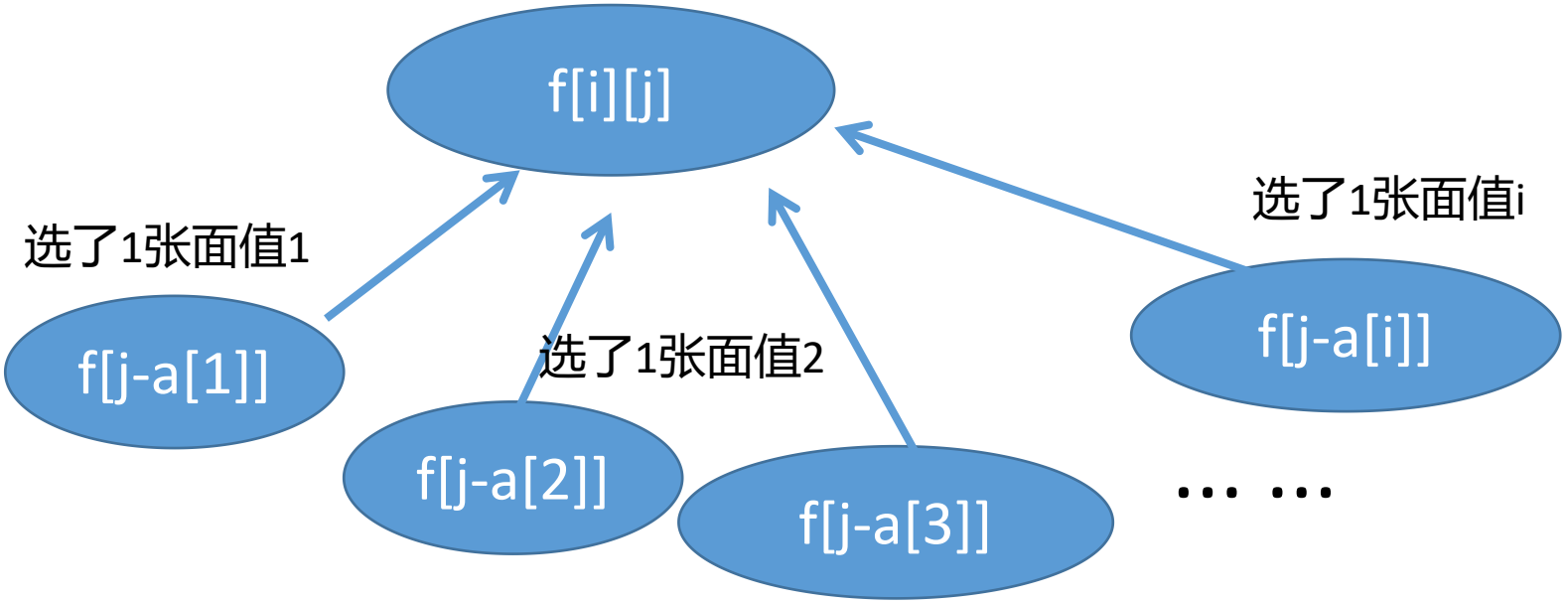
4. 初始条件时什么？

f[0]=1

思路：状态转移

f[j] 表示组成面值为j的方案数

f[j] 可以有哪些状态转移过来呢？




```
f[j]=f[j]+f[j-a[1]]+f[j-a[2]]+ ...+f[j-a[n]];
```

代码：

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  long long int n,m,a[10000],dp[10000];
4  //dp[j]表示 表示组成面值为j的方案数
5  int main(){
6      cin>>n>>m;
7      for(int i=1;i<=n;i++)
8          cin>>a[i];
9      dp[0]=1;
10     for(int i=1;i<=n;i++)
11         for(int j=a[i];j<=m;j++)
12             dp[j]+=dp[j-a[i]];
13     cout<<dp[m];
14 }
```

[7115] 摆花[DP]

小明的花店新开张，为了吸引顾客，他想在花店的门口摆上一排花，共m盆。通过调查顾客的喜好，小明列出了顾客最喜欢的n种花，从1到n标号。为了在门口展出更多种花，规定第i 种花不能超过ai盆，摆花时同一种花放在一起，且不同种类的花需按标号的从小到大的顺序依次摆列。试编程计算，一共有多少种不同的摆花方案。

输入每组输入数据的第一行包含两个正整数n和m，中间用一个空格隔开。第二行有n个整数，每两个整数之间用一个空格隔开，依次表示a1、a2、...an。

数据规模： $0<n\leqslant 100$，$0<m\leqslant 100$，$0\leqslant ai\leqslant 100$。

输出每组输出只有一行，一个整数，表示有多少种方案。注意：因为方案数可能很多，请输出方案数对1000007取模的结果。

[7115] 摆花[DP]

下面是对样例数据的解释：

有2种摆花的方案，分别是(1，1，1，2)，(1，1，2，2)。括号里的1和2表示两种花，比如第一个方案是前三个位置摆第一种花，第四个位置摆第二种花。

样例输入

2 4 //n种花 ，放置 m盆花

3 2//第i 种花不能超过ai盆

样例输出

2

题解

题意可简单描述为要选m盆花，共有n种花可选，每种为a[i]盆。

将花盆数量看作背包容量；
将花看作物品，体积是1，第 i 种物品最多选 ai 个；

问题：将背包装满的方案数是多少？
这是典型的多重背包求方案数问题：

状态表示：f[i,j] 表示前i个物品，总体积是j的方案数；
状态计算：

f[i,j] =
f[i-1,j]+f[i-1,j-1] + ... + f[i- 1, j-a[i]]。

时间复杂度分析

状态总共 n^2 个，计算每个状态的值需要 O(a) 的时间，其中 a 是平均每种花的数量。因此总时间复杂度是 (n^2a)，符合题目数据要求。

```cpp
int n, m, a, f[105];
int main() {
    cin >> n >> m;
    f[0] = 1; // 记录初始状态
    for (int i = 0; i < n; i++) {
        cin >> a;

        // 枚举摆放m~1盆花的状态
        for (int j = m; j > 0; j--)
            // 枚举放j盆花时的状态时放当前花盆的数量
            for (int k = 1; k <= j && k <= a; k++)
                // 当前状态下再加上放k盆花时的状态
                f[j] = (f[j] + f[j - k]) % 1000007;

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

记忆化搜索

```
6   const int N = 110, mod = 1000007;
7   int n, m, a[N], state[N][N];
8   int dfs(int u, int sum)//选第几个物品，当前的和是多少
9   {
10      int res = 0;
11      if(sum == m) return 1;
12      if(sum > m || u > n) return 0;
13      if(state[u][sum]) return state[u][sum];
14      for(int i = 0; i <= min(a[u], m - sum); i ++)//枚举数量
15          res = (res + dfs(u + 1, sum + i)) % mod;
16      state[u][sum] = res;//记录状态
17      return res;
18  }
```

```
19  int main()
20  {
21      cin >> n >> m;
22      for(int i = 1; i <= n; i ++) cin >> a[i];
23
24      cout << dfs(1, 0) << endl;
25      return 0;
26  }
```

[2502] 吃水果

n个小朋友站成一排，等着吃水果。一共有 m 种水果，每种水果的数量都足够多。

现在，要给每个小朋友都发一个水果，要求：在所有小朋友都拿到水果后，恰好有 k 个小朋友拿到的水果和其左边相邻小朋友拿到的水果不同（最左边的小朋友当然不算数，即最左边的小朋友不包含在 k 个小朋友内）。

请你计算，一共有多少种不同的分发水果的方案

一行，三个整数 n,m,k。

输出格式

一个整数，表示合理的分发水果的方案总数量对 3998244353 取模后的结果。

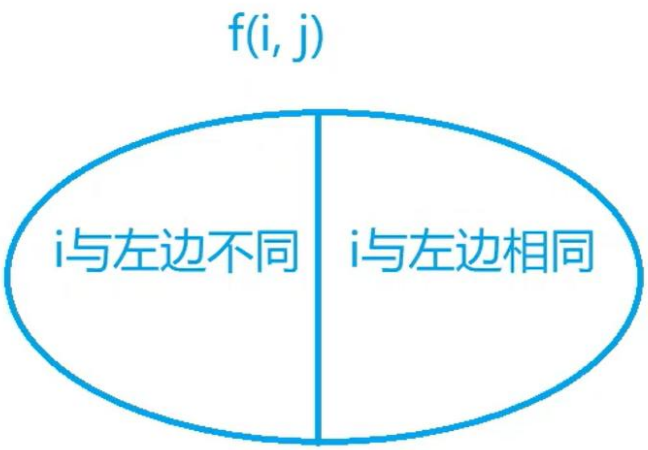
f[i][j]：前i个小朋友中,恰好有j个小朋友拿到的苹果跟左边的小朋友不一样的所有方案数。

//题目要的是给n为同学分完以后 恰好有 k 个小朋友拿到的水果和其左边相邻小朋友拿到的水果不同

//所以答案也就是f[n][k]

1. 如果相同
假如前面 i−1 个小朋友已经选好，则对于每一种方案，第 i个小朋友有 1 种选择，且满足的小朋友数量不变那么这种方案的数量就为f[i−1,j]

2. 如果不同
假如前面 i−1个小朋友已经选好，则对于每一种方案，第 i个小朋友有 m−1种选择，且满足的小朋友数量 +1
那么这种方案的数量就为(m−1)×f[i−1,j−1]
综上所述，得出状态转移方程: fi,j=f[i−1,j]+(m−1)×f[i−1,j−1]



```cpp
typedef long long LL;
const int MOD = 998244353;
const int N = 2010;

int n, m, k;
int f[N][N];

int main()
{
    cin >> n >> m >> k;
    f[1][0] = m;
    for (int i = 2; i <= n; i ++ )
        for (int j = 0; j <= k; j ++ )
            f[i][j] = (f[i - 1][j] + (LL)(m - 1) * f[i - 1][j - 1]) % MOD;
    cout << f[n][k] << endl;

    return 0;
}
```

```
{
    ll n, m, k, i, j;
    cin >> n >> m >> k;
    dp[1][0] = m;
    //第一个同学没法和左边的同学相等（因为没有），
    //所以只能和0个左边的同学不相等 然后其他(j>0的情况)不可能 所以就是0
    // 他有m种选择，怎么选都能和左边0个同学不相等
    for (i = 2; i <= n; i++)
    {
        dp[i][0] = m;
        //dp[i][0]的定义是前i个同学拿完以后，恰有0个同学 和左边的同学不一样，也就是所有同学都相等
        //那么可以都是第一种水果  都是第二种水果 … …   都是第m种水果共m种可能
        for (j = 1; j <= k; j++)
        {
        //思考状态怎么转移，想想第i位同学与第i-1位同学之间的状态可能
        //第i位同学和第i-1位同学的状态有两种  拿一样的或者不一样的

        //两个同学拿的水果是一样的，那么dp[i][j]，前i位同学中j对组合不同
        //但是第i与i-1位同学拿的一样（无法 对 这j对组合做出贡献）
        //所以这j对组合在前i-1位同学中就完成了，所以就是 前i-1位同学中有j对不同，就是dp[i-1][j]
        //所以我们第i位同学和第i-1同学 拿一样的，那么我们就有dp[i][j]=dp[i-1][j]
            dp[i][j] = dp[i - 1][j];
        //然后我们也可以选择拿不一样的
        //那么这j对组合 有一对是第i和第i-1对同学 然后前i-1位同学中再有j-1对 组合，就是dp[i-1][j-1]
        //那么如何转移过来呢
        //dp[i-1][j-1]时，不管第 i-1位同学选啥，第i位同学选的和他不一样有 m-1种选法
        //所以dp[i-1][j-1]的每一种可能，第i位同学都有m-1种拿法和他不一样
        // 所以对第i位同学来说 就是有dp[i - 1][j - 1] * (m - 1)种拿法
            dp[i][j] = (dp[i][j] + dp[i - 1][j - 1] * (m - 1)) % mod;//取模勿忘
        }
    }
    cout << dp[n][k] << endl;//最后输出我们归纳的答案就好啦
```

[2503]有趣的数

我们把一个数称为有趣的，当且仅当：

它的数字只包含 0,1,2,3，且这四个数字都出现过至少一次。

所有的 0 都出现在所有的 1 之前，而所有的 2 都出现在所有的 3 之前。

最高位数字不为 0。

因此，符合我们定义的最小的有趣的数是 2013。

除此以外，4 位的有趣的数还有两个：2031 和2301。

请计算恰好有 n 位的有趣的数的个数。

由于答案可能非常大，只需要输出答案除以 10^9+7 的余数。

分析

采用动态规划思想，每一次决策都基于前一次决策的最优解。

即对一个n位数的解都基于前一个n－1位的数的最优解。

我们对一个数的第n位规定一个状态集：即到这一位为止还有几个数字没有使用（我们有 0 1 2 3 共四个数）。

分析

根据规则来说，共有6种状态：

0——用了2，剩0，1，3

1——用了0，2，剩1，3

2——用了2，3，剩0，1

3——用了0，1，2，剩3

4——用了0，2，3，剩1

5——全部用了

于是我们需要让用户输入位数，然后声明同等位数的数组，在每个元素里是6种状态中所包含的该状态下的“符合条件的数”的个数。

分析

然后用动态规划思想从最小位数开始逐层往上计算。

例：

对于i位状态5的计算，考虑在i-1位时有三种状态可以到达状态5，第3种，此时只能在i位填3，所以＊1；第4种，此时只能在i位填1，所以＊1；第5种，此时能在i位填2或3（参考规则），所以＊2；

```
states[i][5] = (states[j][3] + states[j][4] + states[j][5] * 2) % mod;
```

其他同上。

由于采用动态规划，所以取余并没有什么影响。

最后完成计算只需输出i位的第5种状态中的个数。

```cpp
#include <iostream>
using namespace std;
typedef long long ll;
const int N = 1010;

ll states[N][7];

int main(){
    ll mod = 1000000007;
    ll n;
    cin>>n;

    for(ll i =0;i<6;i++)
    for(ll i=1;i<=n;i++)
    {
        ll j = i-1;
        states[i][0] = 1;
        states[i][1] = (states[j][0] + states[j][1] * 2) % mod;
        states[i][2] = (states[j][0] + states[j][2]) % mod;
        states[i][3] = (states[j][1] + states[j][3] * 2) % mod;
        states[i][4] = (states[j][1] + states[j][2] + states[j][4] * 2) % mod;
        states[i][5] = (states[j][3] + states[j][4] + states[j][5] * 2) % mod;
    }
    cout<<states[n][5]<<endl;
    return 0;
}
```

数学方法

根据题目关系，可以将数分为两组，$(0,1)$ 和 $(2,3)$，两组之间互不影响；

我们设 $(0,1)$ 组有 k 位数，则 $(2,3)$ 组有 $n-k$ 位数，首先由于每个数字都要出现至少一次，所以 k 的取值范围为 $2 \leq k \leq n-2$。

对于 $(0,1)$ 组的 k 个数，由于 0 必须在 1 前且最高位不能为 0，所以这 k 个数只能在后 $n-1$ 位中选，故有 C_{n-1}^{k}。

对于 $(0,1)$ 组中的 k 个数，0 至少有一个，且 0 必须在 1 前，所以排列可以为 $011\ldots11, 0011\ldots11, \ldots, 00\ldots001$ 共 $k-1$ 种。同理 $(2,3)$ 组有 $n-k-1$ 种可能。

故计算公式为 $\sum_{2}^{n-2} C_{n-1}^{k}(k-1)(n-k-1)$。

```cpp
const int N = 1e3 + 7;
const int MOD = 1e9 + 7;
int n;
int C[N][N];

int main() {
    cin >> n;
    for (int i = 0; i <= n; ++i) {
        for (int j = 0; j <=i; ++j) {
            if (!j) C[i][j] = 1;
            else C[i][j] = (C[i-1][j] + C[i-1][j-1]) % MOD;
        }
    }
    int ans = 0;
    for (int k = 2; k <= n - 2; ++k) {
        ans = (ans + 1LL * C[n-1][k] * (k-1) % MOD * (n-k-1)) % MOD;
    }
    cout << ans;
    return 0;
}
```

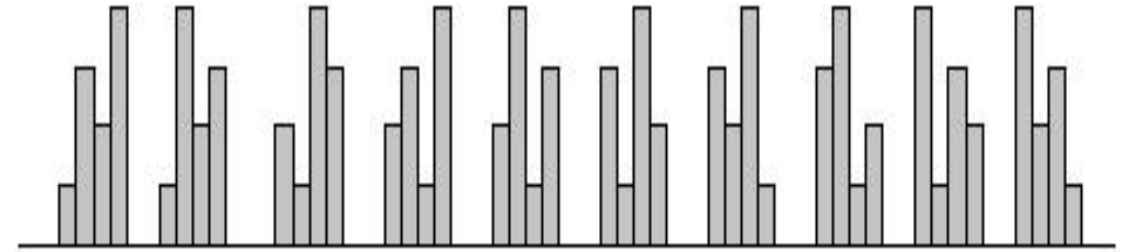
[2506]装饰围栏

有 N 块长方形的木板，长度分别为 1,2,…,N，宽度都是 1。现在要用这 N 块木板组成一个宽度为 N 的围栏，满足在围栏中，每块木板两侧的木板要么都比它高，要么都比它低。也就是说，围栏中的木板是高低交错的。我们称“两侧比它低的木板”处于高位，“两侧比它高的木板”处于低位。显然，有很多种构建围栏的方案。

每个方案可以写作一个长度为 N 的序列，序列中的各元素是木板的长度。

[2506]装饰围栏

把这些序列按照字典序排序，如下图所示，就是 N=4 时，所有满足条件的围栏按照木板长度的字典序排序后的结果。



All cute fences made of $N = 4$ planks, ordered by their catalogue numbers.

现在给定整数 C，求排名为 C 的围栏中，各木板的长度从左到右依次是多少。

[2506]装饰围栏

[3049] 计数问题

给定两个整数 a 和 b，求 a 和 b 之间的所有数字中0~ 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…

[2503]有趣的数

问题1：

$1\sim\underbrace{9\cdots9}_{n个9}$ 中各数有多少？

可以将 $1\sim\underbrace{9\cdots9}_{n个9}$ 数集根据数的个数进行划分：

1位： 1-9

2位： 10-99

……

n位： $\underbrace{10\cdots0}_{n-1个0}-\underbrace{99\cdots9}_{n个9}$

对于一个n位数来说：

取最高位为 a_n，其它位可取

从 $\underbrace{00\cdots0}_{n-1个0}$ 到 $\underbrace{99\cdots9}_{n-1个9}$

即对于 $a_n=1\sim9$，低位皆可依次选取 $\underbrace{00\cdots0}_{n-1个0}$ 到 $\underbrace{99\cdots9}_{n-1个9}$

得子问题2：

从 $\underbrace{00\cdots0}_{n-1个0}$ 到 $\underbrace{99\cdots9}_{n-1个9}$ 的各数字有多少个？

同样可用动态规划集合划分解决。f[n][i]表示该集合中数字i的个数

$$f[n][i]=\begin{cases}00\cdots0-09\cdots9\\10\cdots0-19\cdots9\\\vdots\quad\vdots\\90\cdots0\sim99\cdots9\end{cases}$$

对于每个i可分为最高位上出现的i和低位上的i。

最高位出现在 $\underbrace{0\cdots0}_{n-1个}-\underbrace{0\cdots9}_{n-1个}$

一共 10^{n-1} 个。

其余位正好是 $10\times f[n-1][i]$

[2503]有趣的数

回到问题1：

$1\sim\underbrace{99\cdots9}_{n个9}$ 可根据位数分为

$1\sim\underbrace{99\cdots9}_{n-1个9}$ 和 $\underbrace{100\cdots0}_{n-1个0}\sim\underbrace{99\cdots9}_{n个9}$

对比问题2，最高位不能为0.

同样将数i分为最高位上的和低位的。

$$g[n][i]=\begin{cases}\boxed{100\cdots0}\sim199\cdots9\\200\cdots0\sim299\cdots9\\\vdots\\900\cdots0\sim999\cdots9\end{cases}$$

低位上数位个数由问题2解决

故为 $9\times f[n-1][i]+g[n-1][i]$

高位1~9个数均为 10^{n-1} 个.

问题3：

从1~x各数字有多少个？

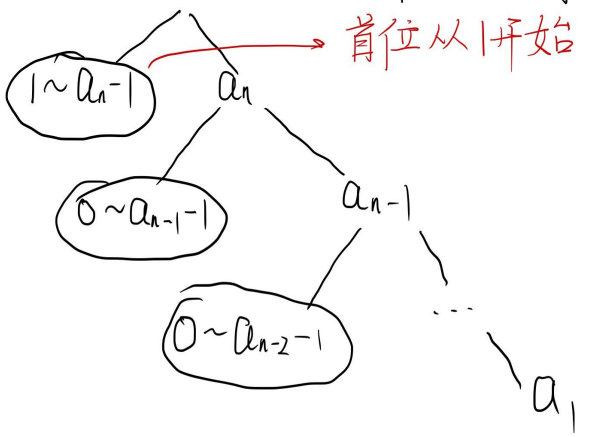
将x以十进制拆分得：

$a_n\ a_{n-1}\ a_{n-2}\cdots a_1$

以位数分类：$1\sim\underbrace{99\cdots9}_{n-1个9}$（问题1）

及 $\underbrace{100\cdots0}_{n-1个0}\sim a_n a_{n-1}a_{n-2}\cdots a_n$，根据

YXC划分法对该数集划分得：



[2503]有趣的数

其中每个左孩子中的解都是容前面同样方法求得的：

对于每个集合，可划分为三个部分：

$a_n a_{n-1} \cdots a_i$ [0 / 1 / 2 / ⋮ / a_{j-1}] [00…0 / 99…9]

第一部分是已经确定的前缀

第二部分是当前位，

第三后缀可从 00…0 迭代至 99…9.

则分别对于每个部分进行统计：

用 last 保存前缀方便计算

[2503]有趣的数