动态规划

神奇口袋

神奇的口袋 (百练2755)

- 有一个神奇的口袋, 总的容积是40,

用这个口袋可以变出一些物品, 这些物品的总体积必须是40

- John现在有n ($1 \le n \le 20$)个想要得到的物品,

每个物品的体积分别是$a_1, a_2, \ldots, a_n$

- John可以从这些物品中选择一些, 如果选出的物体的总体积是40, 那么利用这个神奇的口袋, John就可以得到这些物品
- 现在的问题是, John有多少种不同的选择物品的方式

- **输入**

  输入的第一行是正整数n ($1 \le n \le 20$),
  表示不同的物品的数目；

  接下来的n行，每行有一个1到40之间的正整数，
  分别给出$a_1, a_2, \ldots, a_n$的值

- **输出**

  输出不同的选择物品的方式的数目

- 输入样例

  3

  20

  20

  20

- 输出样例

  3

枚举的解法：

枚举每个物品是选还是不选, 共2^{20}种情况

递归解法

```
#include <iostream>
using namespace std;
int a[30];
int N;
int Ways (int w, int k)  { // 从前 k种 物品中选择一些, 凑成 体积w 的做法数目
        if( w == 0 ) return 1;
        if( k <= 0 )  return 0;
        return Ways(w, k-1) + Ways(w - a[k], k-1);
}
int main()   {
        cin >> N;
        for( int i = 1;i <= N; ++ i )
            cin >> a[i];
        cout << Ways(40, N);
        return 0;
}
```

动态规划

```
#include <iostream>
using namespace std;
int a[30];  int N;
int Ways[40][30];  //Ways[i][j]表示从前j种物品里凑出体积i的方法数
int main()   {
     cin >> N;
     memset(Ways, 0, sizeof(Ways));
     for( int i = 1; i <= N; ++ i ) {
          cin >> a[i];  Ways[0][i] = 1;
     }
     Ways[0][0] = 1;
     for( int w = 1 ; w <= 40; ++ w ) {
          for( int k = 1; k <= N; ++ k ) {
               Ways[w][k] = Ways[w][k-1];
               if(w-a[k] >= 0)     Ways[w][k] += Ways[w-a[k]][k-1];
          }
     }
     cout << Ways[40][N];     return 0;
}
```

“我为人人”型递推解法

- 此问题仅在询问容积40是否可达，40是个很小的数
- 可以考虑对**值域空间**——即对容积的可达性进行动态规划
- 定义一维数组 **int sum[41]**
- 依次放入物品，计算每次放入物品可达的容积
- 并在相应空间设置记录，最后判断sum[40] 是否可达，

到达了几次

```
#include <iostream>
using namespace std;
#define MAX 41
int main(){
    int n, i, j, input;
    int sum[MAX];
    for( j = 0; j < MAX; j++ ) sum[j]=0;
    cin >> n;
    for( i = 0; i < n; i++ ){
        cin >> input;
        for( j = 40; j >= 1; j-- )
            if( sum[j] > 0 && j+input <= 40 )
                sum[j+input] += sum[j];
                //如果j有sum[j]种方式可达, 则每种方式加上input就可达 j+input
        sum[input]++;
    }
    cout << sum[40] << endl;
    return 0;
}
```

0-1背包问题（POJ3624)

- 有N件物品和一个容积为M的背包
- 第i件物品的体积w[i], 价值是d[i]
- 求解将哪些物品装入背包可使**<u>价值总和最大</u>**
- 每种物品只有一件，可以选择放或者不放
- (N<=3500, M <= 13000)

0-1背包问题（POJ3624）

- 用 **F[i][j]** 表示取前**i**种物品，使它们总体积不超过**j**的最优取法取得的价值总和

→ 要求F[N][M]

- 边界: **if (w[1] <= j)**

  F[1][j] = d[1];

  else

  F[1][j] = 0;

0-1背包问题 (POJ3624)

- 用 F[i][j] 表示取前i种物品，
- 使它们总体积不超过j的最优取法取得的价值总和

递推: **F[i][j] = max(F[i-1][j], F[i-1][j-w[i]]+d[i])**

- 取或不取第 i 种物品，两者选优

  (j-w[j] >= 0才有第二项)

0-1背包问题 (POJ3624)

F[i][j] = max(F[i-1][j], F[i-1][j-w[i]]+d[i])

- 本题如用记忆型递归，需要一个很大的二维数组，会超内存
- 注意到这个二维数组的下一行的值，

  只用到了上一行的正上方及左边的值

→因此可用滚动数组的思想，只要一行即可

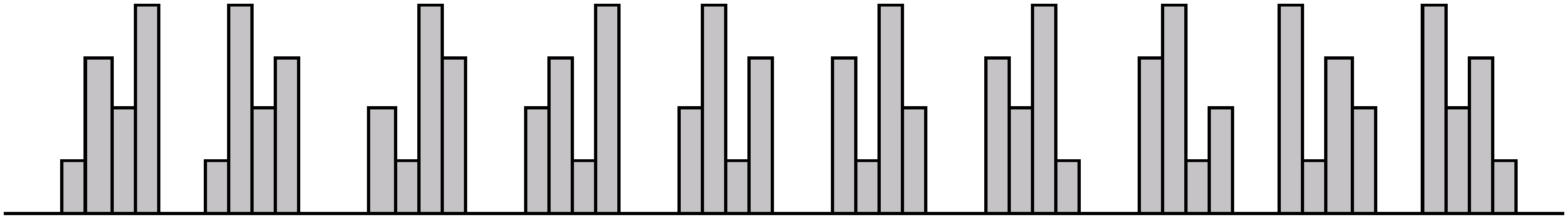
- 即可以用一维数组，用"人人为我" 递推型动归实现

动态规划

美妙的栅栏

例题： POJ 1037 美妙的栅栏

- N 个木棒, 长度分别为1, 2, …, N.
- 构成美妙的栅栏
  - 除了两端的木棒外, 每一跟木棒, 要么比它左右的两根都长, 要么比它左右的两根都短。
  - 即木棒呈现波浪状分布, 这一根比上一根长了, 那下一根就比这一根短， 或反过来



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

例题： POJ 1037 美妙的栅栏

- 问题:
- 符合上述条件的栅栏建法有很多种，对于满足条件的所有栅栏，按照字典序(从左到右，从低到高) 排序
- 给定一个栅栏的排序号，请输出该栅栏，即每一个木棒的长度

例题： POJ 1037 美妙的栅栏

- 输入数据
  - 第一行是测试数据的组数 K (1 <= K <= 100）
  - 接下来的K行, 每一行描述一组输入数据
  - 每一组输入数据包括两个整数 N 和 C
  - N (1 <= N <= 20) 表示栅栏的木棒数, C表示要找的栅栏的排列号
- 输出数据
  - 输出第C个栅栏，即每一个木棒的长度
- 设20个木棒可组成的栅栏数是 T

→ 假设 T 可以用64-bit长整数表示， 1 < C <= T

- 输入样例

  2
  2 1
  3 3

- 输出样例

  1 2
  2 3 1

解题思路

- 问题抽象：
- 给定1到N 这N个数字，将这些数字高低交替进行排列，把所有符合情况的进行一个字典序排列，问第C个排列是一个怎样的排列
- 总体思想
  - **动归 ＋ 排列计数**
- 动归

动归解题思路

- 设 A[i] 为 i 根木棒所组成的**合理方案数目**

  看看能否找出A[i]和A[i-1]或A[i-j]之间的递推关系

  (所有木棒总数是i)

  称 i 根木棒的**合法方案集合**为S(i)

动归解题思路

- 在选定了 某根木棒 x 作为**第一根木棒**的情况下

→ 剩下i-1根木棒的合法方案数目是A[i-1]

BUT 这A[i-1]种方案，并不是每种都能和x形成新的合法方案

  - 第一根比第二根长的方案称为**DOWN**方案
  - 第一根比第二根短的方案称为**UP**方案

则S(i-1)中，

  - 第一根木棒比x长的DOWN方案
  - 第一根木棒比x短的UP方案

→ 才能和x构成S(i)中的方案

动归解题思路

- 置A[i] = 0 (初始化), 先枚举 x

  针对每个x, 枚举 x 后面的那根木棒 y

- 如果y > x (y < x的情况类推), 则:

  A[i] += 以y打头的DOWN方案数

  但以y打头的DOWN方案数, 又和y的长短有关

  所以难以直接从 A[i-1] 或 A[i-j] 推出 A[i]

动归解题思路

- 考虑将A[i]这种粗略的状态描述方式细化, 即**加上限制条件后分类**. 设

  $$A[i] = \sum B[i][k] \qquad k = 1...I$$

  B[i][k] 是S(i)中以第k短的木棒打头的方案数.

  尝试对 B 进行动归第k短指的是 i根木棒中第k短

动归解题思路

- $B[i][k] = \sum B[i-1][M]_{(DOWN)} + \sum B[i-1][N]_{(UP)}$

  M = **k ... i-1**, N = 1… k-1

  还是没法直接推. 于是把B再分类细化:

  $B[i][k] = C[i][k][DOWN] + C[i][k][UP]$

  C[i][k][DOWN] 是S(i)中以第k短 的木棒打头的DOWN方案数

  然后试图对C进行动归

  $C[i][k][UP] = \sum C[i-1][M][DOWN]$

  M = k ... i -1

  $C[i][k][DOWN] = \sum C[i-1][N][UP]$

  N = 1… k-1

  初始条件：C[1][1][UP]=C[1][1][DOWN] = 1

n根木棒的总方案数是:

Sum{C[n][k][DOWN] + C[n][k][UP] }

k=1…n

动归解题思路

- 经验：
- 当选取的状态，难以进行递推时

(分解出的子问题和原问题形式不一样)

→ 考虑将**状态分类细化**，

→ 即增加维度，然后在新的状态上尝试递推

排序计数

- 如1, 2, 3, 4的全排列, 共有 4! 种, 求第10个的排列是(从1计起)?
- 先试首位是1, 后234有3!=6种<10, 说明首位1偏小

→问题转换成求2开头的第(10-6=4)个排列, 而3！ =6 >= 4, 说明首位恰是2

- 第二位先试1 (1没用过), 后面 2!=2 个<4, 1偏小, 换成3 (2用过了) 为第二位, 待求序号也再减去2!, 剩下2了. 而此时2!>=2, 说明第二位恰好是3
- 第三位先试1, 但后面1!<2, 因此改用4. 末位则是1了
- 这样得出, 第10个排列是2-3-4-1

排序计数

- 本题待求方案的序号为C
- 先假设第1短的木棒作为第一根, 看此时的方案数P(1)是否>=C

→ 如果否, 则应该用第二短的作为第一根, C 减去P(1) ;

→ 再看此时方案数P(2)和C比如何

→ 如果还 < C , 则应以第三短的作为第一根, C再减去P(2) …

- 若发现 第 i 短的作为第一根时， 方案数已经不小于C

→ 则确定应该以第 i 短的作为第一根

→ C减去第 i 短的作为第一根的所有方案数, 然后再去确定第二根…

排序计数

- 试第1根时, 假设用的是第k短的:

  P(n, k) = C[n][k][up] + C[n][k][down];

  试第i根时, (i从1开始算, i及其右边一共n-i+1根)

  P(n-i+1, k) = C[n-i+1][k][up] + C[n-i+1][k][down];

```
//POJ1037 A decorative fence by Guo Wei
#include <iostream>
#include <algorithm>
#include <cstring>
using namespace std;
const int UP =0;    const int DOWN =1;
const int MAXN = 25;
long long C[MAXN][MAXN][2];
//C[i][k][DOWN] 是S(i)中以第k短的木棒打头的DOWN方案数,
//C[i][k][UP] 是S(i)中以第k短的木棒打头的UP方案数, 第k短指i根中第k短
void Init(int n) {
   memset(C, 0, sizeof(C));
   C[1][1][UP] = C[1][1][DOWN] = 1;
   for( int i = 2 ; i <= n; ++ i )
          for( int k = 1; k <= i; ++ k ) { //枚举第一根木棒的长度, 第k短
             for( int M = k; M <i; ++M ) //枚举第二根木棒的长度, 比第一根长
                   C[i][k][UP] += C[i-1][M][DOWN];
             for( int N = 1; N <= k-1; ++N ) //枚举第二根木棒的长度, 比第一根短
                   C[i][k][DOWN] += C[i-1][N][UP];
          }
   //总方案数是 Sum{ C[n][k][DOWN] + C[n][k][UP] }  k = 1.. n;
}  //复杂度 n^2
```

$$C[i][k][UP] = \sum_{M = k \dots i-1} C[i-1][M][DOWN]$$

$$C[i][k][DOWN] = \sum_{N = 1 \dots k-1} C[i-1][N][UP]$$

```
void Print(int n, long long cc)  { //n根木棒, 求第cc个排列
    long long skipped = 0; //已经跳过的方案数
    int seq[MAXN];          //最终要输出的答案
    int used[MAXN];         //木棒是否用过
    memset(used, 0, sizeof(used));
    for( int i = 1; i<= n; ++ i ) {  //依次确定每一个位置i的木棒
            int k = 0;
            int No = 0;  //长度为k的木棒是剩下的木棒里的第No短的, No从1开始算
            for( k = 1; k <= n; ++k ) { //枚举位置i的木棒的长度k
                skipped = 0;
                if(!used[k]) {
                        ++ No;  //k是剩下的木棒里的第No短的
                        if( i == 1 )
                            skipped = C[n][No][UP] + C[n][No][DOWN];
                        else {
                            if( k > seq[i-1] && ( i <=2 || seq[i-2]>seq[i-1]))
                                    skipped = C[n-i+1][No][DOWN];   //合法放置
                            else if( k < seq[i-1] && (i<=2 || seq[i-2]<seq[i-1]))
                                    skipped = C[n-i+1][No][UP];        //合法放置
                        }
```

```
                    if( skipped >= cc )
                        break;
                    else
                        cc -= skipped;
                }
            }
            used[k] = true;
            seq[i] = k;
        }
        for( int i = 1; i <= n; ++i )
            if( i < n)  printf("%d ", seq[i]);
            else        printf("%d", seq[i]);
        printf("\n");
}
```

```
int main() {
		int T, n;
		long long c;
		Init(20);
		scanf("%d", &T);
		while(T--) {
			scanf("%d %lld", &n, &c);
			Print(n, c);
		}
		return 0;
}
```