[11]递推与递归

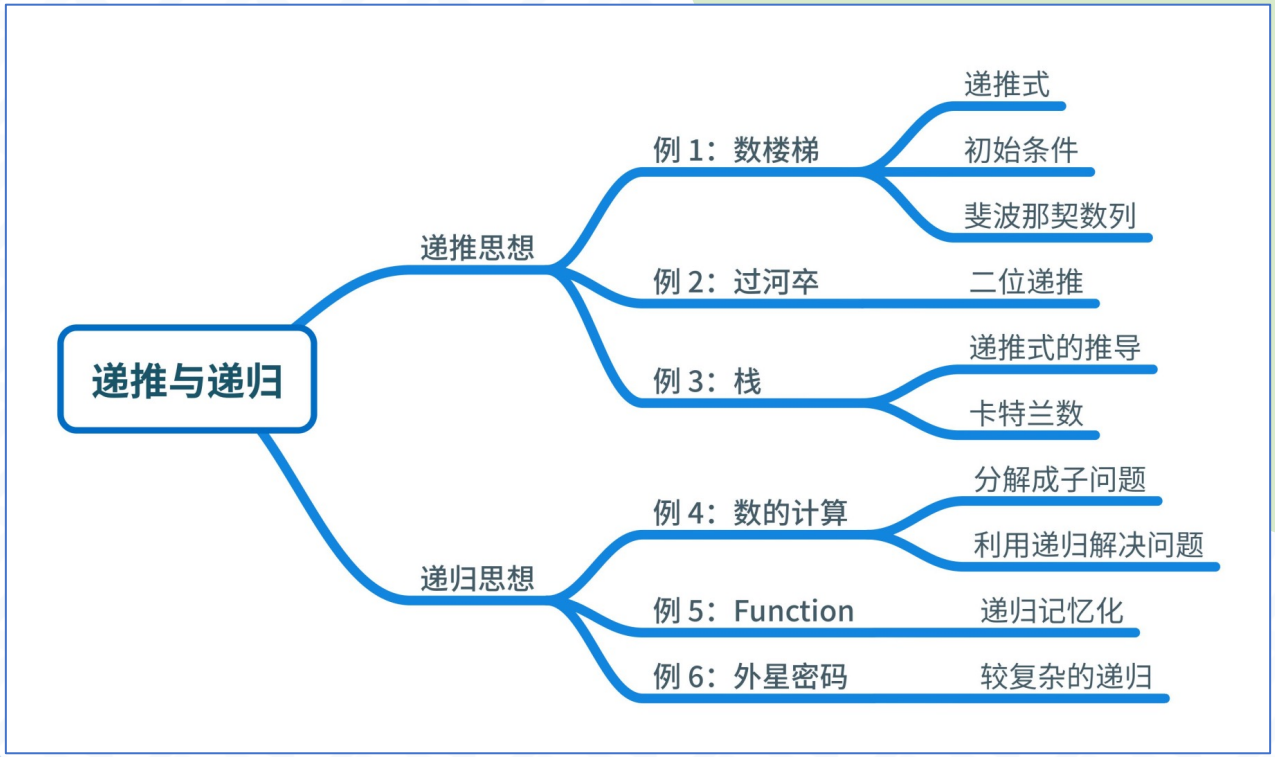
深入浅出程序设计竞赛
第 2 部分 – 初涉算法
V 2021-02

洛谷

版权声明

- 其它《深基》配套资源、购买本书等请参阅：https://www.luogu.com.cn/blog/kkksc03/IPC-resources
- 如果课件有任何错误，请在这里反馈 https://www.luogu.com.cn/discuss/show/296741

本章知识导图


递推与递归
递推思想
例 1：数楼梯
递推式
初始条件
斐波那契数列
例 2：过河卒
二位递推
例 3：栈
递推式的推导
卡特兰数
递归思想
例 4：数的计算
分解成子问题
利用递归解决问题
例 5：Function
递归记忆化
例 6：外星密码
较复杂的递归

第 11 章 递推与递归

递推思想

知道递推式，也知道初始条件，从初始条件开始往上顺推直到求得目标解的思想就是递推。

请翻至课本 P154

数楼梯

例 11.1 数楼梯 (洛谷P1255)

楼梯有 $N(N \leq 5000)$ 阶，上楼可以一步上一阶，也可以一步上二阶，计算共有多少种不同的走法。

例如，n=4时有以下5种走法：

$0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4$

$0 \rightarrow 1 \rightarrow 2 \rightarrow 4$

$0 \rightarrow 1 \rightarrow 3 \rightarrow 4$

$0 \rightarrow 2 \rightarrow 3 \rightarrow 4$

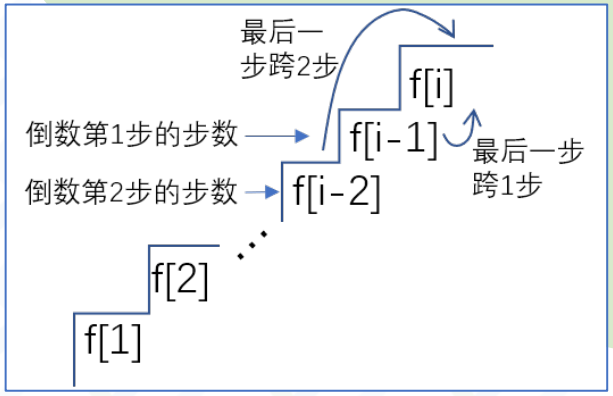
$0 \rightarrow 2 \rightarrow 4$

数楼梯

分析：可使用回溯法枚举所有走法，但是数据范围稍大就会超时。

想要走到第1000个台阶，必须先走到第999个台阶或者第998个台阶，然后一步跨到第1000个。

1到第1000级的走法数量 = 1到第999级的走法数量 + 1到第998级的走法数量



令从第1个到第 i 个台阶的走法数量是 f[i]，可得：

$$f[i]=f[i-2]+f[i-1]$$

数楼梯

后一项等于前面两项之和。这不就是斐波那契数列吗？

即使归纳得到了这个式子（称为递推式），还不能说明这是一个斐波那契数列。

计算这个数列中的某个元素需要得到它前面的两项元素就行。如果知道 f[1] 和 f[2] 的值，就可以推导出整个数列。f[1] 和 f[2] 是这个数列的初始条件。

对于这个问题而言初始条件是什么呢？

- n=1 显然只有一种走法。
- n=2 可以分两步走，或者两步并一步走，有两种走法。

所以可以确定 f[1]=1，f[2]=2。

数楼梯

现在有了递推式，有了初始条件，就可以获得完整的数列了。经过计算，可以得到这个数列前面几项是 ：

1、2、3、5、8、13、21 ...

这就是斐波那契数列从第 2 项开始的序列。而斐波那契的初始条件就是 f[1]=f[2]=1。

像这样知道递推式，也知道初始条件，从初始条件开始往上顺推直到求得目标解的思想就是递推。

数楼梯

由于斐波那契数字增长得很快，所以需要使用高精度计算。

这里使用了“模拟与高精度”一章中提供的封装好的大整数结构体。

```
int main() {
    cin >> N;
    Bigint f[5010];
    f[1] = Bigint(1);
    f[2] = Bigint(2);
    for (int i = 3; i <= N; i++)
        f[i] = f[i - 2] + f[i - 1];
    f.print();
    return 0;
}
```

过河卒

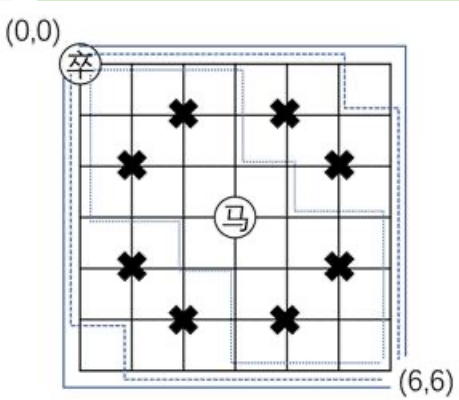
例 11.2 过河卒（洛谷 P1002，NOIP2002普及组）

棋盘左上角 (0,0) 有一个过河卒，需要走到右下角 (n,m)，卒每次可以向下或者向右一格。

棋盘上有一个固定的马，该马所在的点和所有跳跃一步可达的点称为对方马的控制点，卒无法经过。

求卒从起点到终点所有的路径条数。

如图，当卒要从 (0,0) 往右或下走到 (6,6)，马在 (3,3)。马能跳到的位置已经打上了叉，卒不能走到这些点，一共有 6 种合法方案。



过河卒

分析：暴力枚举还是枚举往右或往下然后回溯搜索，依然会超时。

如果那个马不存在，从左上角到右下角一共有多少种走法？

记从原点 (0,0) 走到坐标 (i, j) 的方法数量是 f[i, j] 。

当卒从起点开始，笔直往右或者笔直往下，只有唯一的一种走法。

所以 k≥0 时，f[k,0]=f[0,k]=1，这就是递推的初始条件。

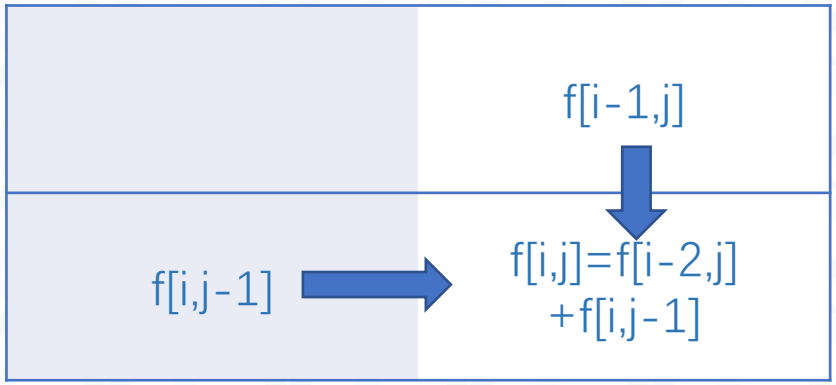
过河卒

记从原点 (0,0) 走到坐标 (i, j) 的方法数量是 f[i, j] 。

如何从点 (0,0) 到点 (1,1) 呢？

要么是从点 (0,1) 走下一格，要么是从点 (1,0) 往右走一格，即 f[1,1]=f[0,1]+f[1,0]。

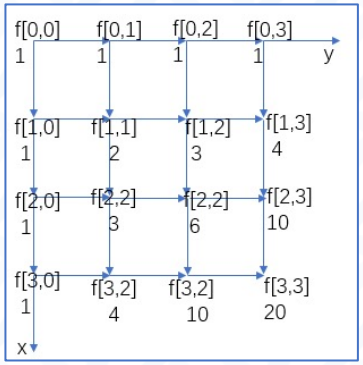
当 i>0 且 j>0时，f[i, j] = f[i-1,j] + f[i,j-1]，这就是递推式。

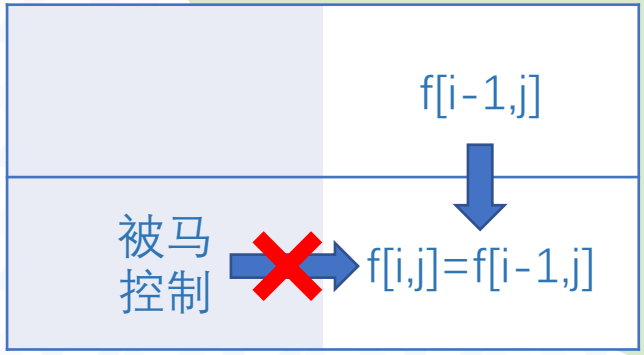


过河卒

有了递推式，有了初始条件，就可以求出完整的 f 数组的值了。

如果没有马，f 数组如左图。





如果有些点因为马的把守而不能走呢？无法从马的控制点转移到下一个点，那么就不进行转移，如右图。

此外初始条件和递推范围变为f[0,0]=1，递推范围 $i\geqslant 0,j\geqslant 0,ij\neq 0$。

过河卒

在实现时，需要预处理哪些点是马的控制点，然后对所有的点进行递推操作。只有转移来的格点存在，才会累加方案数。

```
#include <iostream>
#define MAXN 22
using namespace std;
long long f[MAXN][MAXN] = {0};
int ctrl[MAXN][MAXN], n, m, hx, hy;
int d[9][2] = {{0,0},{1,2},{1,-2},{-1,2},
  {-1,-2}, {2,1}, {2,-1}, {-2,1}, {-2,-1}
};// 马的控制范围相对于马位置的偏移量

int main() {
  cin >> n >> m >> hx >> hy;
  for(int i = 0; i < 9; i++) {
    int tmpx=hx+d[i][0],tmpy=hy+d[i][1];
    if(tmpx>=0&&tmpx<=n&&tmpy>=0&&tmpy<=m)
      // 判断在棋盘范围内
      ctrl[tmpx][tmpy] = 1;
      // 记录马的控制点
  }
  f[0][0] = 1 - ctrl[0][0];
  //若原点就是马控制点，那初始数量是0，否则1
  for (int i = 0; i <= n; i++)
    for (int j = 0; j <= m; j++) {
      if (ctrl[i][j])continue;
      //如果这个点是控制点，那么跳过
      if (i != 0) f[i][j] += f[i - 1][j];
      //若不在横轴上就加上面路径数
      if (j != 0) f[i][j] += f[i][j - 1];
      //该点不在纵轴上就加左边的路径数
    }
  cout << f[n][m]; // 输出答案
  return 0;
}
```

栈

例 11.3 栈（洛谷 P1044，NOIP2003 普及组）

有一个单端封闭的管子，将 $N(1 \le N \le 18)$个不同的小球按顺序放入管子的一端。在将小球放入管子的过程中也可以将管子最顶上的一个或者多个小球倒出来。

请问倒出来方法总数有多少种？

例如，将小球 [1 2 3] 依次加入到管子中，倒出来的方法可以是 [1 2 3][3 2 1][2 3 1] [2 1 3] [1 3 2]

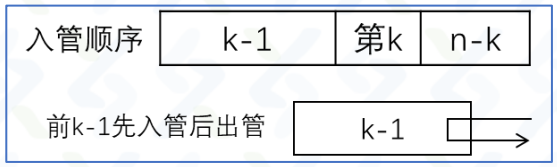
但 [3 1 2] 不行，因为加入 3 之前，管子里面已有 1 和 2 了，如果 3 最先出去，那么接下来出去的只能是 2，而 1 被压在最底下。

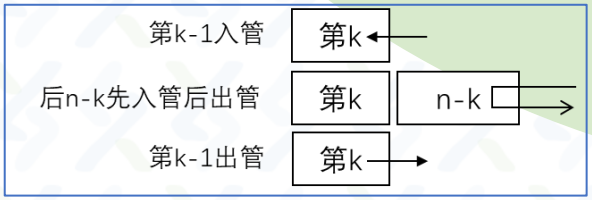
栈

分析：设 i 个元素一共有 h[i] 种出管方式，求 n 个元素的出管方式。其中每个元素都可能最后一个出管。假设第 k 个小球最后出管：

- 比 k 早入且早出有 k-1 个数，有 h[k-1] 种出管方式；
- 比 k 晚入且早出有 n-k 个数，有 h[n-k] 种出管方式，

一共有 h[k-1]×h[n-k] 种出管方式。





递推式为 h[n] = h[0]×h[n-1] + h[1]×h[n-2] + … + h[n-1]×h[0]

初始条件为 h[0]=h[1]=1

栈

像这种只有一个开口、元素先进后出的管子称为栈，在数据结构线性表一章我们会更详细地介绍栈的性质使用方式。

h 数组里面的数字就是卡特兰数，前几项是 1,1,2,5,14,42。

卡特兰数有很多奇妙的性质，会在《进阶篇》中仔细讨论。

```
#include<cstdio>
int main() {
    int n, h[20] = {1, 1};
    scanf("%d", &n);
    for (int i = 2; i <= n; i++)
        for (int j = 0; j < i; j++)
            h[i] += h[j] * h[i - j - 1];
    printf("%d", h[n]);
    return 0;
}
```

递归思想

构造函数，这个函数在运行过程中调用自己，从而解决问题的思路被称为递归思想。

请翻至课本 P158

数的计算

例 11.4 数的计算（洛谷 P1028，NOIP2001 普及组）

给出自然数 $n(n \leq 1000)$，最开始时数列中唯一的一项就是 n，可以对这个数列进行操作，生成新的数列。请问最后能生成几种不同的数列？

1. 原数列直接作为一种合法的数列；
2. 在原数列的末端加入一个自然数，但是它不能超过该数列最后一个数字的一半；
3. 加入自然数后的数列继续按此规则从第一条进行处理，直到不能再加新元素为止。

数的计算

输入数字 6，可以生成以下数列：

1：6 (原数字)

2：6 1 （在数列1后添加了1，无法再添加了）

3：6 2 （在数列1后添加了2）

4：6 2 1（在数列3后添加了1，无法再添加了）

5：6 3 （在数列1后添加了3）

6：6 3 1 （在数列5后添加了1，无法再添加了）

7：6 3 2 （在数列5后添加了2）

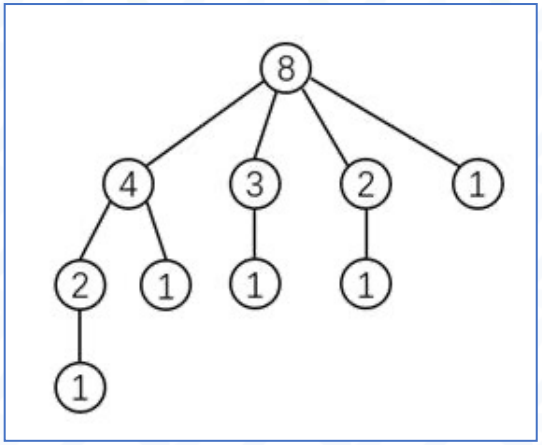
8：6 3 2 1（在数列7后添加了2，无法再添加了）

数的计算

分析：例如数列最开始只有一个元素 8，在末尾加入一个新元素，列表就变成 [8 4]、[8 3]、[8 2]、[8 1]，算上 [8] 一共有 5 种情况。

计算更长的数列的方案数怎么办呢？

分别计算 [4]、[3]、[2]、[1] 按照这样的操作能有几种情况，然后累加统计即可。以 4 开头（3、2、1 同理）的所有合法数列，都可以接续到 8 的后面。



数的计算

原来是要解决 n=8 的问题，现在分解成了四个规模更小但是本质上是同样的子问题。

直到 n=1 时，没法继续分解，可以直接返回唯一一种数列，即 [1]。

然后返回上一层（n=2）接收到所有小规模问题的答案，合并统计处理获得这个规模下的答案，再返回上一层……直到求得问题的解。

```
int sol(int x) {
    if (x == 1) return 1;
    int ans = 1;
    for (int i = 1; i <= x / 2; i++)
        ans += sol(i);
    return ans;
}
```

像这样构造函数，这个函数在运行过程中调用自己，从而解决问题的思路被称为递归思想。

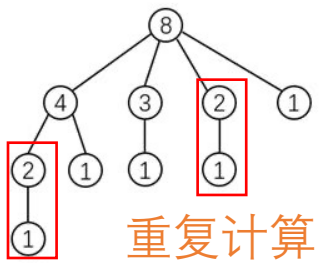
数的计算

答案正确，但是并不能通过本题：运行效率很低导致程序超时。

例如，sol(2) 可能由 sol(4) 调用，也有可能被 sol(8) 调用，但是 sol(2) 的值固定不变，这里却被重复运行了很多次。

开数组 f，f[i] 就是当问题规模为 i 的时候的答案。初始化为 -1，说明 f[i] 还没有被计算过。

使用同样的办法求解，如果发现已经计算过就直接返回 f[i]，否则仍递归计算，然后将结果存入数组中以便之后再次调用。




```
int n, f[1010];
int sol(int x) {
    int ans = 1;
    if (f[x] != -1)
        return f[x];
    for (int i=1; i<=x/2; i++)
        ans += sol(i);
    return f[x] = ans;
}
```

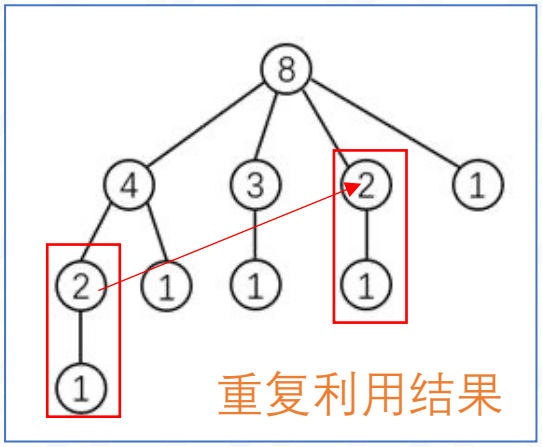
```
int main() {
    cin >> n;
    memset(f, -1, sizeof(f));
    f[1] = 1;
    cout << sol(n) << endl;
    return 0;
}
```

数的计算

这样的话每个数字最多只会只计算一次，因为一旦计算完成就会被存下来，便于日后使用。这样的思想被称为记忆化搜索。

有的同学发现本题可以写出递推式 f[i] = 1 + f[1] + f[2] .. + f[i/2]，f[1] = 1。请感兴趣的读者尝试使用递推求解本题。

有的情况下进行递推，需要求出初始条件，还需要确定递推顺序、所以这时使用递归思想会容易一些。



Function

例 11.5 Function （洛谷 P1464）

对于一个递归函数 w(a,b,c)：

- $a \le 0$ 或 $b \le 0$ 或 $c \le 0$ 返回值 1；
- $a > 20$ 或 $b > 20$ 或 $c > 20$ 返回 w(20,20,20)；
- $a < b$ 并且 $b < c$ 返回 w(a,b,c−1)+w(a,b−1,c−1)−w(a,b−1,c)；
- 其它情况返回 w(a−1,b,c)+w(a−1,b−1,c)+w(a−1,b,c−1)−w(a−1,b−1,c−1)；

给出 a,b,c 要求输出 w(a,b,c)。输入的数据在 long long 范围内。

Function

分析：如果输入数据不在 (0,20] 这个范围内，强制返回 1 或者 w(20,20,20)。根据题意写函数，建立数组记录 w 的值进行记忆化。

```
long long f[25][25][25];
long long w(long long a, long long b, long long c) {
    if (a <= 0 || b <= 0 || c <= 0) return 1;
    else if (a > 20 || b > 20 || c > 20) return w(20, 20, 20);
    else if (f[a][b][c] != 0) return f[a][b][c];
    else if (a < b && b < c)
        f[a][b][c] = w(a,b,c-1)+w(a,b-1,c-1)-w(a,b-1,c);
    else
        f[a][b][c] = w(a-1,b,c)+w(a-1,b-1,c)+w(a-1,b,c-1)-w(a-1,b-1,c-1);
    return f[a][b][c];
}
```

本题输出答案的大小并不好估计，可以先尝试使用 long long 类型，但是尝试几个输入后发现即使用 int 类型也可以通过。

本题也可以使用递推方法，只是边界问题和枚举顺序稍不好处理，感兴趣的读者可以自己尝试使用递推实现本题。

外星密码

例 11.6 外星密码 (洛谷 P1928)

有一种压缩字符串的方式：对于连续的 D($2 \le D \le 99$) 个相同的子串 X 会压缩为 "[DX]" 的形式，而 X 可能可以进行进一步的压缩。

比如说字符串 CBCBCBCB 可以压缩为[4CB]或者[2[2CB]]。

现给出压缩后的字符串，求压缩前的字符串原文。

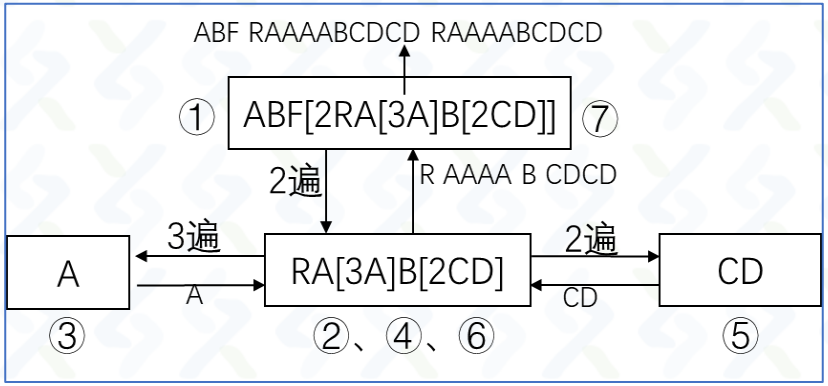
[2[2CB]] = [2CB][2CB] = CBCBCBCB

外星密码

分析：假设只有一层方括号，那只需要找到方括号，就可以将该部分还原。

如果方括号的“重复部分”里还有方括号呢？没关系，设法把里面的方括号继续展开即可。因此可以写成递归函数。

字符串 ABF[4RA[2A]B[3C]] 的展开如图：



外星密码

```
#include<iostream>
#include<string>
using namespace std;
string expand() {
    string s = "", X;
    char c; int D;
    while (cin >> c) {// 持续读入字符，直到全部读完
        if (c == '[') { // 发现一个压缩区
            cin >> D; // 读入D
            X = expand(); // 递归地读入X
            while (D--) s += X; // 重复D次X并进行拼接
            // 上面不能写成while (n--) s+=read();
        }
        else if (c == ']')
            return s; // 压缩区结束，返回已经处理好的X
        else s += c; // 如果不是'['和']'，那还是X的字符，加进去即可
    }
    return s;
}
int main() {
    cout << expand();
    return 0;
}
```

课后习题与实验

学而时习之，不亦说乎。学而不思则罔，思而不学则殆。——孔子

请翻至课本 P162

小结

递推

需要确定递推式、初始（边界）条件，从微观到宏观。

递归

将一个大的任务分解成若干个规模较小的任务，而且这些任务的形式与结构和原问题一致，然后将结果合并，直到达到边界。

有些问题使用递推策略和递归策略都能解决，但有些问题只能将大问题分割成小问题，但是却很难建立递推式，或者不好确定递推顺序，这种情况下应当使用递归策略。

但递归需要记录每一层的状态，因此可能会比较占用空间。

课后习题

思考题

递推与递归的例题中，哪些可以使用另外一种思路（比如递归的例题是否可以使用递推完成）？

如果可以的话，尝试使用另外一种方式完成这些例题。

课后习题

习题 11.3 选数（洛谷 P1036，NOIP2002 普及组）

已知 n 个整数 $x_1, x_2, \ldots, x_n (x_i \leq 5\times10^6)$，以及 1 个整数 $k(k < n \leq 20)$。从 n 个整数中任选 k 个整数相加，可分别得到一系列的和。要求你计算出和为素数的方案共有多少种。

习题 11.4 覆盖墙壁（洛谷 P1990）

有长为 N 宽为 2 的墙壁和两种砖头：一种是长 2 宽 1 的条形砖，另一种是 L 型覆盖 3 个单元的砖头。砖头可以旋转，且无限量提供。

请计算用这两种来覆盖整个墙壁的方案数，对 10000 取余。

课后习题

习题 11.5 秘密奶牛码（洛谷 P3612，USACO2017 January）

给定一个长度不超过 30 的字符串，不断对这个将这个字符串后面拼接自身的“旋转字符串”（旋转字符串是指把原字符串的最后一个字符移动到第一个之前）

比如 COW 拼接后变为 COWWCO，再变成 COWWCOOCOWWC，这样可以扩展成一个无限长度的字符串。

给定 $N(N \leq 10^{18})$，求这个字符串第 N 个字符是什么。第一个字符是 $N = 1$。

课后习题

习题 11.6 黑白棋子的移动（洛谷 P1259）

有 $2n(4 \le n \le 100)$ 个棋子排成一行，初始时先摆上 n 个白棋子，然后再摆上 n 个黑棋子，同时最右边还有 2 个空位。

移动棋子的规则：每次必须同时移动相邻的两个棋子，颜色不限，可以右空位上，但不能调换两个棋子的位置。每次移动必须跳过若干个棋子（不能平移），要求最后能移成黑白相间的一行棋子。要求。

请编程打印出移动过程。例如，当$n = 5$时，移动的过程如下：

```
step 0:ooooo*****--
step 1:oooo--****o*
step 2:oooo****--o*
step 3:ooo--***o*o*
step 4:ooo*o**--*o*
step 5:o--*o**oo*o*
step 6:o*o*o*--o*o*
step 7:--o*o*o*o*o*
```

课后习题

习题 11.7 幂次方（洛谷 P1010，NOIP1998 普及组）

任何一个正整数都可以用 2 的幂次方的和表示。例如$137 = 2^7 + 2^3 + 2^0$，同时约定方次用括号来表示，即a^b 可表示为a(b)。

由此可知， 137 可表示为：2(7)+2(3)+2(0)。进一步，$7 = 2^2 + 2 + 2^0$ (2^1用 2 表示)，$3 = 2 + 2^0$ ，所以最后可表示为 2(2(2)+2+2(0))+2(2+2(0))+2(0)。

给出 $n(n \le 20000)$，按照题目要求输出将 n 变为 2 和 0 组成的幂次方式子。

课后习题

习题 11.8 地毯填补问题（洛谷 P1228）

迷宫是一个边长为 $2^k, (0 < k \leq 10)$ 的正方形，公主站在迷宫的一个方格上。

要求你使用 L 形覆盖 3 格的小地毯不重不漏的覆盖整个迷宫（除了公主站立的位置）。请输出具体方案，方案可能不唯一。

例如右图：

当k=2，公主站在(3,3)时的一种方案
数字代表L形地毯的方向

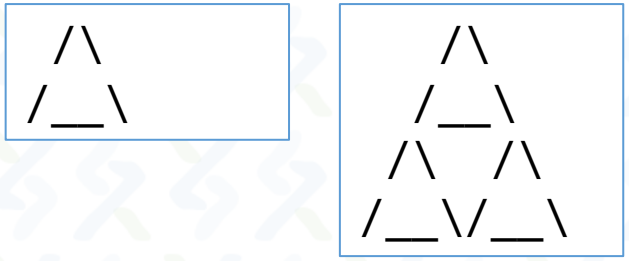
4	4	3	3
4	4	4	3
2	4	公	1
2	2	1	1

课后习题

习题 11.9 南蛮图腾（洛谷 P1498）

南蛮图腾是一种递归图形。当规模为 1 时，南蛮图腾是一个简单的三角形，如下图左。

规模每增加 1，图形就变得复杂了：把原来规模的图形复制三次，分别放置于上方，左下角和右下角，组成了一个更大的三角形。当规模为 2 的时候，图形如下图右：



给出规模 n(n≤10)，请画出对应规模的图形。