NOIP 复习资料

（C++版）

主 编 葫芦岛市一高中 李思洋

完成日期 2012 年 8 月 27 日

前 言

有一天，我整理了 NOIP 的笔记，并收集了一些经典算法。不过我感觉到笔记比较凌乱，并且有很多需要修改和补充的内容，于是我又搜集一些资料，包括一些经典习题，在几个月的时间内编写出了《NOIP 复习资料》。

由于急于在假期之前打印出来并分发给同校同学（**我们学校既没有竞赛班，又没有懂竞赛的老师。我们大家都是自学党**），《NOIP 复习资料》有很多的错误，还有一些想收录而未收录的内容。

在“减负”的背景下，暑期放了四十多天的假。于是我又有机会认真地修订《NOIP 复习资料》。

我编写资料的目的有两个：总结我学过（**包括没学会**）的算法、数据结构等知识；与同学共享 NOIP 知识，同时使我和大家的 RP++。

大家要清醒地认识到，《NOIP 复习资料》页数多，是因为程序代码占了很大篇幅。这里的内容只是信息学的皮毛。对于我们来说，未来学习的路还很漫长。

基本假设

作为自学党，大家应该具有以下知识和能力：

① 能够熟练地运用 C++语言编写程序（**或熟练地把 C++语言“翻译”成 Pascal 语言**）；

② 能够阅读代码，理解代码含义，并尝试运用；
③ 对各种算法和数据结构有一定了解，熟悉相关的概念；
④ 学习了高中数学的算法、数列、计数原理，对初等数论有一些了解；
⑤ 有较强的自学能力。

代码约定

N、M、MAX、INF 是事先定义好的常数（**不会在代码中再次定义，除非代码是完整的程序**）。N、M、MAX 针对数据规模而言，比实际最大数据规模大；INF 针对取值而言，是一个非常大，但又与 int 的最大值有一定差距的数，如 100000000。

对于不同程序，数组下标的下限也是不同的，有的程序是 0，有的程序是 1。阅读程序时要注意。

阅读顺序和方法

没听说过 NOIP，或对 NOIP 不甚了解的同学，应该先阅读附录 E，以加强对竞赛的了解。

如果不能顺利通过初赛，你就应该先补习初赛知识。这本《NOIP 复习资料》总结的是复赛知识。

如果没有学过 C++语言，应该先选择一本 C++语言教材。一般情况下，看到“面向对象编程”一章的前一页就足够了（NOIP **不用“面向对象编程”，更不用摆弄窗口对话框**）。

附录 G 介绍了一些书籍和网站。你应该选择一本书，认真地学习。再选择一个网站，作为练习的题库。

第一单元对竞赛中常用的操作和简单的算法分析进行了总结，算作对 C++语言的巩固。同时，阅读这一单元之后，你应该选择一个合适的 C++代码编辑器。

第二到第六单元介绍了竞赛常用的算法。阅读每一章时，应该先阅读“小结”——名曰“小结”，实际上是“导读”。

这五个单元除了经典习题，还有某些思想和算法的具体实现方法。这些信息可能在明处，也可能在暗处，阅读时要注意挖掘和体会。如果有时间，应该在不看解析和代码的前提下独立完成这些题。

第七单元是第六单元的一个部分，由于它的内容来自《背包九讲》，所以单独放在一个单元。

从第八单元开始，到第十三单元，基本上就没有习题了。换句话说，该“背课文”了。

第八单元介绍了常用的排序算法。你可以有选择地学习，但一定要掌握“STL 算法”和“快速排序”。

第九单元介绍了基本数据结构，你一定要掌握第九单元前五小节的内容（**本单元也有应该优先阅读的“小结”**）。有余力的话，第六小节的并查集也应该掌握。

第十单元介绍了与查找、检索有关的数据结构和算法。你也可以有选择地学习。

第十一单元与数学有关。数学对于信息学来说具有举足轻重的地位。标有“！”的应该背下来，至于其他内容，如果出题，你应该能把它解决。

第十二单元仍与数学有关。

第十三单元是图论。学习时要先阅读“小结”，把概念弄清楚。之后要掌握图的实现方法。接下来要掌握一些经典图论算法：Kruskal 算法、Dijkstra 算法、SPFA、Floyd 算法、拓扑排序。

附录 F 总结了 2004 年以来 NOIP 考察的知识点，可以作为选择性学习的参考。

在学习算法和数据结构的同时，应该阅读和学习附录 A。

如果你还有余力，你应该学习第十四单元。第十四单元的内容不是必须要掌握的，但是一旦学会，可以发挥 C++语言的优势，降低编程复杂度。

临近竞赛时，应该阅读附录 B 和附录 C，以增加经验，减少失误。

面临的问题

1. 这是复赛复习资料——需要有人能用心总结、整理初赛的知识，就像这份资料一样。
2. 潜在的问题还是相当多的，只是时间不够长，问题尚未暴露。
3. 部分代码缺少解说，或解说混乱。
4. 个人语文水平较差，《资料》也是如此。
5. 没有对应的 Pascal 语言版本。
   如果有人能为 P 党写一个 Pascal 版的 STL，他的 RP 一定会爆增！
6. 希望有人能用LaTeX整理《资料》，并以自由文档形式发布。

最后，欢迎大家以交流、分享和提高为目的修改、复制、分发本《资料》，同时欢迎大家将《资料》翻译成 Pascal 语言版供更多 OIer 阅读！

谢谢大家的支持！

葫芦岛市一高中　李思洋

2012 年 8 月 27 日

目 录

标题上的符号：

1. !：表示读者应该熟练掌握这些内容，并且在竞赛时能很快地写出来。换句话说就是应该背下来。
2. *：表示内容在 NOIP 中很少涉及，或者不完全适合 NOIP 的难度。
3. #：表示代码存在未更正的错误，或算法本身存在缺陷。

第一单元　C++语言基础

1.1　程序结构

(1) 程序框架

- 注释：注释有两种，一种是“//”，另一种是“/* … */”。“//”必须单独放置一行，或代码所在行的后面；而“/*”、“*/”成对存在，可以插入到代码的任意位置。
- 引用头文件：在代码开头写“#include <头文件名>”。如果想引用自己的头文件，需要把尖括号（**表示只从系统目录搜索头文件**）换成双引号（**表示先从 cpp 所在文件夹搜索，然后再到系统文件夹搜索**）。
- 命名空间：很多 C++的东西都要引用 std 命名空间，所以代码中会有“using namespace std;”。
- main()：所有程序都要从 main()开始。
  在所有的算法竞赛中，main()的返回值必须是 0，否则视为程序异常结束，得分为 0 分。
- 语句和语句块：
  1. 语句：一般情况下，一条语句要用一个分号“;”结束。为了美观和可读性，可以把一条语句扩展成几行，也可以把多个语句写到同一行上。
  2. 语句块：用“{”和“}”包围的代码是语句块。无论里面有多少代码，在原则上，语句块所在的整体都视为一条语句。

(2) 选择结构

1. if 语句：if 表示判断。如果条件为真，就执行接在 if 后的语句（**语句块**），否则执行 else 后的语句（**语句块**）。如果没有 else，就直接跳过。if 有以下几种格式：

```
if (条件)           // 如果条件成立，就执行if后面的语句或语句块。
    语句或语句块

if (条件)           // 如果条件成立，就执行if后面的A，否则执行B。
    语句或语句块A
else
    语句或语句块B

if (条件1)          // 实际上，这是if语句内的if语句，即if的嵌套。所以else和if中间要有空格。
    语句或语句块A
else if (条件2)
    语句或语句块B
……
else
    语句或语句块N
```

2. switch 语句：switch 表示选择。它根据条件的不同取值来执行不同的语句。格式如下：

```
switch (表达式)
{
case 值1:
    代码段A
```

```
    break;
case 值2:
    代码段B
    break;
……
default:
    代码段N
    break;
};
```

如果表达式的值是值 1，就执行代码段 A；如果表达式的值是值 2，就执行代码段 B……否则执行代码段 N。

注意：

- default 一部分可以省略。
- 如果不使用 break，那么紧随其后的 case 部分代码也会被执行，直到遇到 break 或 switch 语句结束为止！
- switch 结尾要有一个分号。

3. if、switch 都可以嵌套使用。

【问题描述】输入一个日期，判断它所在年份是否为闰年，并输出所在月份的天数。闰年的判断方法：四年一闰，百年不闰，四百年又闰。

```
int year,month,day;
bool b=false;
cin>>year>>month>>day;
// 判断是否为闰年
if (n%400==0)
    b=true;
else if (n%100!=0 && n%4==0)
    b=true;

if (b)
    cout<<y<<"是闰年。"<<endl;
else
    cout<<y<<"不是闰年。"<<endl;

// 判断所在月份的天数
switch (month)
{
case 1: case 3: case 5: case 7: case 8: case 10: case 12:
    cout<<"这个月有31天。"<<endl;
    break;
case 4: case 6: case 9: case 11:
    cout<<"这个月有30天。"<<endl;
    break;
case 2:
    cout<<"这个月有"<<(b ? 29 : 28)<<"天。"<<endl;
    break;
};
```

（3）循环结构

1．while 语句：如果条件成立，就继续循环，直到条件不成立为止。格式如下：

```
while（条件）
    循环体（语句或语句块）
```

2．do…while 语句：如果条件成立，就继续循环，直到条件不成立为止。它与 while 的最大区别在于，do…while 循环中的语句会被执行至少一次，而 while 中的语句可能一次都没有被执行。格式如下：

```
do
{
    循环体
}
while（条件）;                    // 注意分号
```

4．for 语句：for 分四部分，有初始条件、继续循环的条件、状态转移的条件和循环体。格式如下：

```
for（初始条件；继续循环的条件；状态转移的条件）
    循环体
```

转换成 while 循环，即：

```
初始条件
while（继续循环的条件）
{
    循环体
    状态转移
}
```

for 后三个条件不是必需的，可以省略不写，但分号必须保留。

5．在循环语句内部使用 break，可以跳出循环；使用 continue，可以忽略它后面的代码，马上进入下一轮循环。

注意，这两个语句只对它所在的一层循环有效。

6．写 for 循环时，一定要注意：

- 不要把计数器的字母打错，尤其是在复制/粘贴一段代码的时候。
- 根据算法要明确不等号是“<”、“>”，还是“<=”、“>=”。
- 逆序循环时，不要把自减“--”写成自增“++”！

【问题描述】输入 n，输出 $n!$（$n!=1\times2\times3\times4\times\cdots\cdots\times n$）。结果保证小于 long long 的范围。当输入值为负数时结束程序。

```
int n;
long long r=1;
cin>>n;
while (n>-1)
{
    r=1;
    for (int i=1; i<=n; i++)
        r*=i;
    cout<<n<<"! = "<<r<<endl;
    cin>>n;
}
```

(4) goto 语句

goto 语句用于无条件跳转。要想跳转，首先要定义标签（**在代码开头的一段标识符，后面紧跟冒号**），然后才能 goto 那个标签。

很多教程不提倡使用无条件跳转，因为它破坏了程序结构，还容易给代码阅读者带来麻烦。不过，这不代表 goto 没有使用价值。goto 的一个用途是跳出多层循环：

```
for (int i=0; i<9; i++)
    for (int j=0; j<9; j++)
        for (int k=0; k<9; k++)
        {
            if (满足某种条件) goto __exited;
            ……
        }
__exited:
```

(5) C 与 C++的区别

C++语言是以 C 语言为基础开发出来的，C 语言的大多数内容被保留了下来。在信息学竞赛领域，很多情况下 C 和 C++可以互相转化，甚至不用对代码进行任何修改。

下面是信息学竞赛领域中 C 和 C++的重要区别：

- C++支持用流输入输出，而 C 只能用 scanf 和 printf——再见了，%d!
- C++非常支持面向对象编程，而 C 已经“out”了。
  《资料》中的“高精度算法”就只能用 C++完成，因为在 struct 内定义了成员函数。
  C++可以用更强大的 string 类处理字符串，而不必担心发生某些低级错误。
- C++有强大的 STL，而 C 没有（**有一个小小的 qsort 和 bsearch 算是补偿了**）。
  STL 是很多人从 C 转到 C++的重要原因。
- C 的头文件名仍然可以用在 C++中，不过可能会收到警报——应该去掉“.h”，前面再加一个“c”。
  如<stdio.h>应该改成<cstdio>。
- C 程序运行速度稍优于 C++。不过也没快多少。

总之，C 能做的一切事情，C++也能做；C++能做的一切事情，C 不一定能做。

1.2　数据类型

(1) 基本数据类型

名称	占用空间	别名	数据范围
int	4	signed, signed int, long, long int	-2,147,483,648～2,147,483,647
unsigned int[①]	4	unsigned, unsigned long,unsigned long int	0～4,294,967,295
char	1	char	-128～127
unsigned char	1	unsigned char	0～255
short[②]	2	short int,	-32,768～32,767

① 一般都使用有符号整数，除非范围不够大，或者你确定你的减法运算不会出现“小数减大数”。
② 一般来说，使用 int、long long 更保险一些，除非内存不够用。

		signed short int	
unsigned short	2	unsigned short int	0～65,535
long long[①]	8	signed long long	-9,223,372,036,854,775,808～9,223,372,036,854,775,807[②]
unsigned long long	8		0～18,446,744,073,709,551,615
bool	1		true 或 false
char	1		-128～127
signed char	1		-128～127
unsigned char	1		0～255
float	4		3.4E +/- 38 (7 位有效数字)
double	8	long double	1.7E +/- 308 (15 位有效数字)

(2) 变量与常量

1. 定义变量："变量类型 标识符"，如"int i;"定义了一个名字为 i 的整型变量。
   注意，此时 i 并未初始化，所以 i 的值是不确定的。
2. 定义常量："const 变量类型 标识符=初始值"，如：const int N=90;
3. 合法的标识符：
   - 标识符不能和关键字（**在 IDE 中会变色的词语**）相同。
   - 标识符只能包括字母、数字和下划线"_"，并且开头只能是字母或下划线。
   - 标识符必须先定义后使用。
   - 在同一作用域内，标识符不能重复定义（**即使在不同作用域内也不应重复，否则容易产生歧义**）。
   - C++区分大小写。所以 A 和 a 是两个不同的标识符。

(3) 数组

1. 定义一个一维数组：int a[10];
   这个数组一共 10 个元素，下标分别为 0～9。访问某个元素时，直接用 a 加方括号，如 a[5]。
2. 定义一个二维数组：int b[5][3];
   这个数组一共 5×3＝15 个元素，分别是 b[0][0]、b[0][1]、b[0][2]、b[1][0]……b[4][2]。
   访问某个元素时要用两个方括号，如 b[2][1]。
   多维数组的定义和使用方法与此类似。
3. 数组名和元素的寻址：以上面的 a、b 为例
   - 数组名是一个指针，指向整个数组第一个元素所在的地址。如 a 就是&a[0]、b 就是&b[0][0]。
   - 多维数组的本质是数组的数组，所以 b[0]实际上是 b[0][0]、b[0][1]……的数组名，b[0]就是&b[0][0]。
   - 在内存中，数组中每个元素都是紧挨着的，所以可以直接进行指针的运算。如 a+3 就是&a[3]，**(b+1)就是 b[1][0]，*(*(b+3)+2)就是 b[3][2]。
   - 在竞赛中要尽可能回避这些功能。
4. 字符串：
   - 字符串实际上是 char 的数组。
   - 字符串最后一位必须是'\0'，否则会在进行输出、使用字符串函数时发生意外。

---

① **不要使用"__int64"！**它是 Visual C++特有的关键字。

② 假如 a 是 long long 类型，把超过 2^{31} 的值赋给它时要使用字面值 LL（ULL）：a=123456789012345**LL**。

- 数组，包括字符串，不可以整体地赋值和比较。如果需要，应使用 memcpy 和 memcmp（字符串是 strcpy 和 strcmp）。

5．C++中数组的下标只能从 0 开始（当然可以闲置不用），并且 **int a[10]中 a 的最后一个元素是 a[9]，不是 a[10]！**

6．**C++不检查数组下标是否越界**！如果下标越界，程序很有可能会崩溃！

（4）指针

1．取地址运算符和取值运算符：

- 取地址运算符“&”：返回变量所在的地址。一般用于变量。（而数组名本身就是指针，无需“&”）
- 取值运算符“*”：返回地址对应的值，或用于改变指针所指内存空间的值。只能用于指针。

2．指针的意义：保存另一个变量的内存地址。

3．定义指针：int *p;

定义多个指针时，每个字母的前面都要有“*”。

注意，如果 p 没有被初始化，它就会指向一个未知的内存空间，而错误地操作内存会导致程序崩溃！

4．指针使用实例：

```
int a = 0, b = 1; int c[] = {1,2,3,4,5,6,7,8,9,10};
int *p;                      // 定义一个指针
p=&a;                        // 让p指向a
(*p)=3;                      // 相当于a=3
(*p)=b;                      // 相当于a=b，此时a等于1
// p=b;                      // 非法操作，左边是int *，右边是int，类型不匹配。
p=&b;                        // 让p指向b，从此p和a没关系了
p=c+6;                       // 让p指向c[6]，p和b又没关系了
cout<<*p;                    // 输出p指向的变量的值，即c[6]
p++;                         // 现在p指向了c[7];
p=NULL;                      // 表示p没有指向任何变量
cout<<*p;                    // 由于NULL（0）是一段无意义的地址，所以程序极有可能崩溃。
```

为了不在竞赛中把自己搞晕，请回避指针，对其敬而远之。

（5）引用

通俗地讲，引用是某个变量的别名。下面定义了一个叫 p 的引用，它实际上是 f[0][2]。无论是改变 p 的值，还是改变 f[0][2]的值，结果都是一样的。

```
int &p = f[0][2];
```

使用引用的好处是，在函数的形参中定义引用类型，可以直接修改变量的值，而不用考虑“&”和“*”运算符。像上面一行代码一样，如果频繁调用 f[0][2]，也可以用引用节省篇幅，提高代码可读性。

引用与指针不同。引用被创建的同时也必须被初始化，并且必须与合法的存储单元关联。一旦引用被初始化，就不能改变引用的关系。而指针可以不立刻初始化，也可以改变所指向的内存空间。

（6）结构体

- 结构体用 struct 定义。例如下面代码定义了一个叫 pack 的结构体，它有两个成员，一个叫 value，另一个叫 weight。

```
struct pack
{
    int value, weight;
};
```

- 变量可以定义成上面的 pack 类型：pack p;　　// 不必写成 struct pack p;
- 访问 pack 的成员时，用“.”运算符（指针变量用“->”）：p.value、(&p)->value
- C++中结构体可以像类一样建立自己的构造函数、成员函数，也可以重载运算符。
- 对于 pack 这个结构体，它的内部不允许再有 pack 类型的成员，但是可以有 pack 类型的指针。

1.3　运算符

(1) 运算符的优先级

运算符	结合方式
::	无
.（对象成员）　->（指针）　[]（数组下标）　()（函数调用）	从左向右
++　--　(*typecast*)（强制类型转换）　sizeof　~　!　+（一元）　-（一元） *（取值运算符）　&（取地址运算符）　new　delete	从右向左
.*　->*	从左向右
*　/　%（取余数）	从左向右
+　-	从左向右
<<（左移）　>>（右移）	从左向右
<　<=　>　>=	从左向右
==（判断相等）　!=（判断不等）	从左向右
&（按位与）	从左向右
^（异或）	从左向右
|（按位或）	从左向右
&&（条件与）	从左向右
||（条件或）	从左向右
?:（条件运算符）	从右向左
=　*=　/=　%=　+=　-=　&=　^=　>>=　<<=	从右向左
,	从左向右

(2) 常用运算符的作用

1. 算术运算符：+、-、*、/、%分别表示加、减、乘、除、取余。

   两个整数做除法时，如果除不开，程序将把小数部分直接截断（**不是四舍五入**）。即：整数/整数＝整数，浮点数/浮点数＝浮点数。

   学习过其他语言的同学要注意，C++中没有乘方运算符，“^”也不是乘方运算符。

2. 比较运算符：

   - >、>=、<、<=、==（**相等**）、!=（**不等**）用来判断不等关系，返回值是 true 或 false。

     小心，千万不要把“==”写成“=”！
   - **永远不要对浮点数使用==和!=运算符**！正确的做法是：

     ```
     const double eps = 0.000001;       // 自己控制精度
     ……
     if (d>=2-eps && d<=2+eps) ……       // 相当于 if (d==2)
     ```
   - 不应该判断一个变量的值是否等于 true。安全的做法是判断它是否不等于 false。

3. 位运算：
   - &、|、^分别表示按位与、按位或、按位异或，即两个操作数上每一个二进制位都要进行运算。
   - ~表示按位求反。
   - <<、>>表示二进制位的移动。当某一位数字移动到二进制位的外面之后，它就直接“消失”了。a<<n 相当于 $a\times 2^n$，a>>n 相当于 $a\div 2^n$。

4. 逻辑运算符：
   - &&、||、^分别表示与、或、异或。!表示非。
   - ?:是条件运算，它是 C++唯一一个三目运算符。它的格式如下：A ? B : C。

     如果 A 不为 false，那么返回 B，否则返回 C。

     可以将这个运算符理解为一个简化版的 if。
   - ||、&&、?:是短路运算符[①]。不要在这三种运算符内调用函数或赋值，以免发生难以查出的错误！

5. 比较运算符、位移运算符、逻辑运算符、条件运算符的优先级较低，所以使用时要多加括号，以防出错。

6. 自增（++）、自减（--）运算符：
   - 增量分别是 1 和-1。
   - 这两种运算符只能用于数值类型的变量，不能用于非数值类型变量、常量、表达式和函数调用上。
   - 前缀++、--和后缀++、--的作用效果不同：

     ```
     int i=0, j=8, k=5;
     j = j + (++i);       // i 先自增，变成 1，然后再和 j 相加。执行之后 i=1，j=9。
     k = k + (i++);       // i 先和 k 相加，使 k=6。然后 i 再自增。执行之后 i=2，k=6。
     ```
   - 前缀运算符返回引用类型，后缀运算符返回数值类型。
   - 为了避免错误，不要让++、--和其他能够改变变量的操作在同一行出现！

7. 赋值运算符：
   - 在 C++中赋值是一种运算符。所以你会看到 i=j=0、d[x=y]、return c=i+j 之类的代码。
   - +=、-=、*=、……可以简化书写。例如 a*=2+3 相当于 a=a*(2+3)。

(3) 真值表

p	q	p && q (p & q)	p || q (p | q)	p ^ q	!p (~p)
true (1)	true (1)	true (1)	true (1)	false (0)	false (0)

[①] 例如计算“a && b”，如果 a 为 false，那么实际上结果就是 false——不管 b 是什么，程序都不再计算了。

true (1)	false (0)	false (0)	true (1)	true (1)	false (0)
false (0)	true (1)	false (0)	true (1)	true (1)	true (1)
false (0)	false (0)	false (0)	false (0)	false (0)	true (1)

(4) 类型强制转换

用一对小括号把数据类型包上，则它右侧的变量或常量的**值**（变量本身不变）就变成了对应的类型。如：

```
int i=2;
float c=6.4/(float)i;                    // 把i的值变成float类型。
```

两个操作数进行四则运算，如果想保留小数位，那么两个操作数应该都是浮点数。上面的代码就是这样。

1.4　函数

(1) 定义和使用函数

1. 定义和调用函数：下面定义了一个函数，返回值是 double 类型的，其中有两个参数 i、j，分别是 int 和 float 类型的。

```
double foo(int j, float j)
{
    ……
}
```

   - 如果函数不需要返回任何值，可定义为 void 类型。
   - 函数的定义必须在函数调用的前面。只有在前面添加了函数定义，才能把具体实现放到调用的后面：
     `double foo(int, float);          // 放到调用之前`

2. 返回值：return 值;
   - 函数是 void 类型的，那么 return 后面除了分号，什么都不跟。
   - 调用之后，函数立刻结束。
   - 不可以直接对函数名赋值（**学过 Pascal 或 Basic 语言的同学要特别注意**）。
3. 如果你的函数需要返回指针或引用，你必须注意：不要引用函数内部的变量！因为函数一结束，函数内部的变量就烟消云散，不复存在了。正确做法是引用静态变量（static）或全局变量。
4. 内联函数（inline）：当一个函数内部只有寥寥几句时，如“华氏度变摄氏度”，可以考虑将其定义成内联函数，通知编译器省略函数入栈等操作，直接展开函数内容，以加快运行速度。

```
inline int FtoC(int f) { return (f-32)/9*5; }
```

(2) 传递实参

1. 按值传递：例如 int foo(int n)，在调用 foo 时，程序会把参数复制一份给 n。这样，对 n 的任何修改都不会反映到调用 foo 的参数上面。
   对于按值传递数组，一定要慎重。因为复制数组的元素要浪费很多时间。
2. 传递指针：例如 int foo(int *n)。对 n 的修改会反映到调用 foo 的参数上面。
   - 修改 n 的值时要注意，必须用取值运算符，否则改变的是 n 指向的内存空间[①]。
   - 此外，这种方法可以用于传递数组——调用时只需把数组名作为参数。这时不需要取值运算符。
3. 传递引用：例如 int foo(int &n)。

优点是既可以直接修改调用 foo 的参数，又不会带来指针的麻烦（**无需取值运算符**）。缺点是不能传入常

[①] 使用 const 可防止 n 指向的内存空间发生改变：int foo(**const** int *n)。这时再写 n=5 之类的语句会被报错。

数或表达式。

1.5　输入和输出！

(1) 使用标准输入/输出

头文件：<cstdio>

变量约定：FILE *fin, *fout;——fin、fout 分别代表输入文件和输出文件。把它们换成 stdin 和 stdout，就是从屏幕输入和从屏幕输出。“1.5 字符串操作”也使用了同样的变量。

1. **输出字符串或变量的值**：printf("格式字符串", ……);

   或 fprintf(fout, "格式字符串", ……);

   格式字符：“%”后连接一个字母。

字符	含义	字符	含义
d	整数[①]	e, E	用科学记数法表示的浮点数
u	无符号整数	f	浮点数
o	八进制整数	c	字符
x, X	十六进制整数（**小写、大写**）	s	字符串（**字符数组**）

   常见的修饰符：

   - %5d：5 位数，右对齐。不足 5 位用空格补齐，超过 5 位按实际位数输出。
   - %-5d：5 位数，左对齐。不足 5 位用空格补齐，超过 5 位按实际位数输出。
   - %05d：5 位数，右对齐。不足 5 位用'0'补齐，超过 5 位按实际位数输出。
   - %+d：无论是正数还是负数，都要把符号输出。
   - %.2f：保留 2 位小数。如果小数部分超过 2 位就四舍五入，否则用 0 补全。

1. **输入到变量**

   - **读取不含空白的内容**：scanf("格式字符串", &……);

     或 fscanf(fin, "格式字符串", &……);

     ① 格式字符和 printf 基本一致。

     ② **不要忘记“&”！printf 传的是值，scanf 传的是地址！**

     ③ scanf 和 fscanf 的返回值是：成功输入的变量个数。fscanf 返回 EOF，表示文件结束。

     ④ scanf 和 fscanf 忽略 TAB、空格、回车。遇到这些字符它们就停止读取。

   - **读取单个字符**：fgetc(fin);

     首先要判断它是否为 EOF（文件结束）。如果不是，就可以用强制类型转换变成 char。

     读取到行末时，要注意对换行符的处理。

   - Windows、Linux、Mac 的回车字符是不同的。Linux 是'\n'，Mac 是'\r'，Windows 下是两个字符——'\r'和'\n'。

(2) 使用流输入/输出

头文件：<iostream>

1. **输入到变量**：cin>>n;
2. **输出到屏幕上**：cout<<a;

   可以连续输入、输出，如 cin>>n>>m;　cout<<a<<','<<b<<endl;
3. **换行**：cout<<endl;
4. **格式化输出**：

[①] 在 Windows 下调试时，用“%I64d”输出 long long 类型的值。交卷时，**由于用 Linux 测试，要改成“%lld”**。

头文件：<iomanip>

- **右对齐，长度为 n，不足的部分用空格补齐：**

  ```
  cout.width(n);
  cout.fill(' ');              // 如果想用“0”补齐，就可以把空格换成“0”
  cout<<a;
  ```

  前两行代码，每次输出之前都要调用。

- **输出成其他进位制数：**

  ```
  8:  cout<<oct<<a;
  16:  cout<<hex<<a;
  ```

  其他进位制需要自己转换。

5. **注意，数据规模很大时，流的输入输出速度会变得很慢，甚至数据还没读完就已经超时了。**

   在进行输入输出之前加入这样一条语句：`ios::sync_with_stdio(false);`

   调用之后，用 cin、cout 输入输出的速度就和 scanf、printf 的速度一样了。

1.6　其他常用操作！

本资料常用的头文件：<iostream>、<cstdlib>、<cstring>、<fstream>以及<algorithm>、<stack>、<queue>、<vector>等。

C++的流、容器、算法等都需要引用 std 命名空间。所以需要在#include 后面、你的代码前面加上一句：`using namespace std;`

(1) 库函数

1. **数组的整体操作：**

   头文件：<cstring>

   - 将 a[]初始化：`memset(a, 0, sizeof(a));`

     第二个参数应该传入 0、-1 或 0x7F。传入 0 或-1 时，a[]中每个元素的值都是 0 或-1；如果传入 0x7F 时，那么 a[]中每个元素的值都是 0x7F7F7F7F（不是 0x7F！），可认为是“无穷大”。

   - 将 a[]整体复制到 b[]中：`memcpy(b, a, sizeof(a));`
   - 判断 a[]和 b[]是否等价：`memcmp(a, b, sizeof(a));`　　// 返回 0 表示等价

2. **字符操作：**

   头文件：<cctype>

   - tolower(c)、toupper(c)：将 c 转化为小写或大写。
   - isdight(c)、isalpha(c)、isupper(c)、islower(c)、isgraph(c)、isalnum(c)：分别判断 c 是否为十进制数字、英文字母、大写英文字母、小写英文字母、非空格、字母或数字。

3. **最大值/最小值：**

   头文件：<algorithm>

   max(a,b)返回 a 和 b 中的最小值，min(a,b)返回 a 和 b 中的最大值。

   其实我们可以自己写：

4. **交换变量的值：**swap(a,b)

   头文件：<algorithm>

   其实我们可以自己写：`inline void swap(int &a, int &b) { int t=a; a=b; b=t; }`

5. **执行 DOS 命令或其他程序：**`system("命令行");`

   - 头文件：<cstdlib>
   - 暂停屏幕：`system("pause");`
   - 竞赛交卷或 OJ 提交代码之前必须删除 system，否则会被视为作弊（**如果是 tyvj 甚至都无法提交**）。
   - 如果使用输入重定向，那么命令提示符不会接受任何键盘输入——直接用文件内容代替键盘了。

6. **立刻退出程序：**`exit(0);`

这种方法常用于深度优先搜索。执行后，程序立刻停止并返回 0，所以在调用前应该输出计算结果。

头文件：<cstdlib>

7. **计时**：double a = (double)clock() / (double)CLOCKS_PER_SEC;

上面的 a 对应一个时刻。而将两个时刻相减，就是时间间隔。可用这种方法卡时。

头文件：<ctime>

8. **断言**：assert(条件)

- 条件为假时，程序立刻崩溃。
- 头文件：<cassert>
- 如果定义了 NDEBUG 符号，那么它将不会起任何作用。
- 断言和错误处理不同：例如出现“人数为负数”的情况，如果责任在于用户，那么应该提示错误并重新输入，而不是用断言；如果发生在计算过程，应该用断言来使程序崩溃，以帮助改正代码中的错误。换句话说，错误处理防的是用户的错误，断言防的是代码的错误。

9. **快速排序**：qsort(首项的指针，待排序元素个数，每个元素所占字节，比较函数)

- 头文件：<cstdlib>
- 这是留给 C 党的快速排序，它比 STL 的排序算法啰嗦一些。
- 比较函数返回 int 类型，用于对两个元素的比较。原型如下：

  int compare(const void *i, const void *j);

  如果*i＜*j，则应返回一个小于 0 的数；如果*i==*j 则应返回 0，否则返回一个大于 0 的数。

10. **随机数发生器**：

- 头文件：<cstdlib>
- 产生随机数：

  ① 0~32767 的随机数：rand()

  ② 粗略地控制范围：rand()%范围

  注意，这种方法产生的随机数的分布是不均匀的。

  ③ 精确地控制范围：(double)rand()/RAND_MAX*范围

  ④ 控制在[a, b)之间：a + (int) ((double)rand()/RAND_MAX*(b-a))
- 初始化随机数种子：

  ① srand(数字)：初始化随机数种子。

  ② 注意，这条语句在程序开头使用，并且最多用一次。同一程序、同一平台，srand 中的参数相等，用 rand()产生的随机数序列相同。

  ③ 使随机数更加随机：引用<ctime>，然后这样初始化随机数种子，srand(time(NULL))。

  不要在循环中使用这条语句（例如批量产生随机数据），因为 time 只精确到秒。

11. **数学函数**：

- 头文件：<cmath>
- abs(x)：求 x 的绝对值（该函数同时包含于<cstdlib>）。
- sin、cos、tan、asin、acos、atan：三角函数，角的单位为弧度。

  可用 atan(1)*4 表示 π。
- sinh、cosh、tanh、asinh、acosh、atanh：双曲函数
- sqrt：求平方根
- ceil(x)、floor(x)：分别返回大于等于 x 的最小整数、小于等于 x 的最大整数。注意，参数和返回值都是浮点数类型。
- exp(x)、log(x)、log10：分别求 e^x、$\ln x$、$\lg x$

（顺便提一句，指数可以把加法问题转化为乘法问题，对数可以把乘法问题转化为加法问题。）

- pow(a,b)：计算 a^b。由于精度问题，你仍然需要学会快速幂。
- fmod(a,b)：计算 a 除以 b 的余数。当然，这是浮点数的版本。

(2) 宏定义

宏定义是 C 语言的产物。在 C++中，它真的 out 了。

1. **第一种用法——配合条件编译**：#define DEBUG

定义一个叫 DEBUG 的标识符。它应该与#ifdef 或#ifndef 配合使用。举例如下：

```
#define DEBUG
#ifdef DEBUG
   void print(int v) { cout << v << endl;}
#else
   void print(int) {}
#endif
```

如果符号 DEBUG 存在，那么编译器会编译上面的、能输出数值的 print，否则编译器编译下面的、什么事情都不做的 print。

把上面的#ifdef 换成#ifndef，那么编译的代码正好上面所说的相反。

2. **第二种用法——表达式**：

```
#define N     5000
```

编译时，编译器会用类似于“查找和替换”的方法，把代码中的 N 换成 5000。如果需要换成表达式，应该用括号把它们包围。例如：

```
#define a         1+2
#define b         (1+2)
c=a*2; d=b*2;
```

编译时上面一行会变成“c=1+2*2; d=(1+2)*1;”，显然它们的值是不同的。

所以，用 enum 和 const 代替它是明智之举。

此外，要注意表达式末尾不能有分号（**除非你需要**）。

3. **第三种用法——简易“函数”**：

```
#define FtoC(a) (((a)-32)/9*5)
```

这类似于一个函数。不过，由于编译器只是进行简单替换，所以为了安全，a、b 应该用括号包围，整个表达式也应该用括号包围。

这种“函数”用法和普通函数一样，且速度更快。然而，它很容易出现难以查出的错误。所以，请用内联函数（inline）代替宏定义。

注意，**不要在“参数”中改变变量的值**！

4. **第四种用法——简化一段代码**：

```
#define move(dx, dy)    if (isfull(dir)) return;                    \
                        if (map(x+dx, y+dy)=='0')                   \
                        {                                           \
                           push(dir,x+dx,y+dy,head[dir], dep);      \
                           check(dir);                              \
                        }
```

不要忘记每行后面的“\”，它相当于换行符。这次 move 简化了一大段代码。同样，内联函数也可以。

1.7　字符串操作！

头文件：<cstring>。printf 和 scanf 在<cstdio>中，cin 和 cout 在头文件<iostream>中且位于 std 命名空间内。

下面假设待处理的字符串为 str 和 str2，即：char str[MAX], str2[MAX];

牢记，**字符串的最后一个字符一定是'\0'**。如果字符串内没有'\0'，进行以下操作（**输入除外**）时可能

会造成意外事故。

1. **输出字符串 str**：
   - cout<<str;
   - printf("%s",str); // 输出到文件：fprintf(fout, "%s", str);
2. **输入字符串 str**：
   - scanf("%s", str); // 输出到文件：fscanf(fin, "%s", str);
   - cin>>str;
     以上两种方法在输入时会忽略空格、回车、TAB 等字符，并且在一个或多个非空格字符后面输入空格时，会终止输入。
   - fgets(str, MAX, fin);

     每调用一次，就会读取一行的内容（**即不断读取，直到遇到回车停止**）。
3. **求字符串 str 的长度**：strlen(str) // 这个长度不包括末尾的'\0'。
4. **把字符串 str2 连接到字符串 str 的末尾**：strcat(str, str2)
   - str 的空间必须足够大，能够容纳连接之后的结果。
   - 连接的结果直接保存到 str 里。函数返回值为&str[0]。
   - strncat(str, str2, n)是把 str2 的前 n 个字符连接到 str 的末尾。
5. **把字符串 str2 复制到字符串 str 中**：strcpy(str, str2)
6. **比较 str 和 str2 的大小**：strcmp(str, str2)
   如果 str＞str2，返回 1；如果 str==str2，返回 0；如果 str＜str2，返回-1。
7. **在 str 中寻找一个字符 c**：strchr(str, c)
   返回值是一个指针，表示 c 在 str 中的位置。用 strchr 的返回值减 str，就是具体的索引位置。
8. **在 str 中寻找 str2**：strstr(str, str2)
   - 返回值是一个指针，表示 str2 在 str 中的位置。用 strstr 的返回值减 str，就是具体的索引位置。
   - 此问题可以用 KMP 算法解决。KMP 算法很复杂，在 NOIP 范围内用途不大。
9. **从 str 中获取数据**：sscanf(str, "%d", &i);
   格式化字符串：sprintf(str, "%d", i);
   它们和 fscanf、fprintf 非常像，用法也类似。可以通过这两个函数进行数值与字符串之间的转换。

1.8 文件操作!

正式竞赛时，数据都从扩展名为“in”的文件读入，并且需要你把结果输出到扩展名为“out”的文件中；在 OJ（Online Judge，在线测评器）中则不需要文件操作。具体情况要仔细查看题目说明，以免发生悲剧。

(1) 输入/输出重定向

头文件：<fstream>或<cstdio>

- 方法：只需在操作文件之前添加以下两行代码。

```
freopen("XXXXX.in","r",stdin);
freopen("XXXXX.out","w",stdout);
```

- 调用两次 freopen 后，scanf、printf、cin、cout 的用法完全不变，只是操作对象由屏幕变成了指定的文件。
- 使用输入重定向之后，“命令提示符”窗口将不再接受任何键盘输入（**调用 system 时也是如此**），直到程序退出。这时不能再用 system("pause")暂停屏幕。

(2) 文件流

头文件：<fstream>

流的速度比较慢，在输入/输出大量数据的时候，要使用其他文件操作方法。

- 方法：定义两个全局变量。

```
ifstream fin("XXXXX.in");
ofstream fout("XXXXX.out");
```

- fin、fout 和 cin、cout 一样，也用“<<”、“>>”运算符输入/输出，如：fin>>a; fout<<b;

当然，也可以通过#define 把 fin、fout 变成 cin、cout：

```
#define cin fin
#define cout fout
```

(3) FILE 指针

头文件：<cstdio>或<fstream>

- 方法：定义两个指针。

```
FILE *fin, *fout;
……
int main()
{
    fin = fopen("XXXXX.in", "r");
    fout = fopen("XXXXX.out", "w");
    ......
    fclose(fin); fclose(fout);      // 在某些情况下，忘记关闭文件会被认为是没有产生文件。
    return 0;
}
```

- 进行输入/输出操作时，要注意**函数名的前面有 f**，即 fprintf、fscanf、fgets，并且这些函数的第一个参数不是格式字符串，而是 fin 或 fout，如 fprintf(**fout,** "%d", ans)。
- 想改成从屏幕上输入/输出时，不用对代码动手术，只需把含 fopen 和 fclose 的代码注释掉，并改成：fin=stdin; fout=stdout;

1.9　简单的算法分析和优化

(1) 复杂度

为了描述一个算法的优劣，我们引入算法时间复杂度和空间复杂度的概念。

时间复杂度：一个算法主要运算的次数，用大 O 表示。通常表示时间复杂度时，我们只保留数量级最大的项，并忽略该项的系数。

例如某算法，赋值做了 $3n^3+n^2+8$ 次，则认为它的时间复杂度为 $O(n^3)$；另一算法的主要运算是比较，做了 $4\times2^n+2n^4+700$ 次，则认为它的时间复杂度为 $O(2^n)$。

当然，如果有多个字母对算法的运行时间产生很大影响，就把它们都写进表达式。如对 $m\times n$ 的数组遍历的时间复杂度可以写作 $O(mn)$。

空间复杂度：一个算法主要占用的内存空间，也用大 O 表示。

在实际应用时，空间的占用是需要特别注意的问题。太大的数组经常是开不出来的，即使开出来了，遍历的时间消耗也是惊人的。

(2) 常用算法的时空复杂度

1s 运算次数约为 5,000,000[①]。也就是说，如果把 n 代入复杂度的表达式，得数接近或大于 5,000,000，那么会有超时的危险。

常见的数量级大小：$O(1)<O(\log n)<O(n)<O(n\log n)<O(n^2)<O(n^3)<O(2^n)<O(n!)$

数量级	能承受的大致规模	常见算法
$O(1)$	任意	直接输出结果
$O(\log n)$	任意	二分查找、快速幂
$O(n)$	以百万计（**五六百万**）	贪心算法、扫描和遍历
$O(n\log n)$	以十万计（**三四十万**）	带有分治思想的算法，如二分法
$O(n^2)$	以千计数（**两千**）	枚举、动态规划
$O(n^3)$	不到两百	动态规划
$O(2^n)$	24	搜索
$O(n!)$	10	产生全排列
$O(n^n)$	8	暴力法破解密码

$O(1)$叫常数时间；$O(n)$、$O(n^2)$、$O(n^3)$、$O(n^4)$……叫做多项式时间；$O(2^n)$、$O(3^n)$……叫做指数时间。

(3) 简单的优化方法

1．时间的简单优化方法

时间上的优化在于少做运算、做耗时短的运算等。有几个规律需要注意：

- 整型运算耗时远低于实型运算耗时。
- 位运算速度极快。
- 逻辑运算比四则运算快。
- +、-、*运算都比较快（-、*比+慢一点点，可以忽略不计）。
- /运算比+、-、*慢得多（甚至慢几十倍）。
- 取余%和除法运算速度相当。
- 调用函数要比直接计算慢（因为要入栈和出栈）。

这些规律我们可以从宏观上把握。事实上，究竟做了几步运算、几次赋值、变量在内存还是缓存等多数由编译器、系统决定。但是，少做运算（尤其在循环体、递归体中）一定能很大程度节省时间。

2．空间的简单优化方法

空间上的优化主要在于减小数组大小、降低数组维数等。常用的节省内存的方法有：

压缩储存——见 192 页“A.7　状态压缩*”。

覆盖旧数据——例如滚动数组（见 66 页“(5) 使用滚动数组”）。

要注意的是，对空间的优化即使不改变复杂度，只是改变 n 的系数也是极有意义的。空间复杂度有时对时间也有影响，要想方设法进行优化。

---

① 不同资料给出的数值是不同的，不过这不要紧。在 NOIP 中，只要你的算法正确，就不会在运行时间上“打擦边球”。

3．优化的原则

- 尽量不让程序做已做过的事。
- 不让程序做显然没有必要的事。
- 不要解决无用的子问题。
- 不要对结果进行无意义的引用。

1.10　代码编辑器

为了学习信息学，你需要一台带有 C++编译器的计算机。

为了方便编程，你还需要 IDE（集成开发环境）或具有编程特性的代码编辑器，否则你需要自己从命令行编译程序。

常见的 IDE 和编辑器如下：

名称	适用操作系统	代码编辑功能	编译器	调试功能	单文件编译
DEV-C++	Windows	一般	自备	差	√
Code::Blocks	Linux、Windows	好	×	好	√（调试除外）
Anjuta C/C++ IDE	Linux	好	×	好	×
GUIDE	Windows、Linux	一般	×	一般	√
Emacs（Linux 自带）	Linux、Windows	好	×	好	√
Vim（Linux 自带）	Linux、Windows	好	×	gdb**	g++***
Eclipse****	Windows、Linux	好	×	好	×
记事本＋命令提示符	Windows	×	×	gdb	g++
gedit＋终端	Linux	差	×	gdb	g++
Visual C++	Windows	好	自备*	好	×
Qt Creator	Windows、Linux	好	自备	好	×

* 不是 GCC，而是微软自己的编译器。

** gdb 调试功能的好坏取决于你使用的熟练程度。

*** 会用就能编译。

**** 需要安装 CDT 插件才能编写 C++的程序。

Windows 不预备 C++编译器，你需要自己下载并安装，下载地址 www.mingw.org。Linux 自备 GCC，无需再安装编译器。

第二单元　基础算法

2.1　经典枚举问题

(1) 韩信点兵

【问题描述】正整数 x 除以 3 余 1，除以 5 余 2，除以 7 余 3。问符合条件的最小 x。

```
// 当x比较大时，更合适的做法是解同余式组，具体做法见145页。
int x;
for (x=1; ;x++)
    if (x%3==1  &&  x%5==2  &&  x%7==3) break;
cout<<x;
```

(2) 百鸡问题

【问题描述】1 只公鸡 5 文钱，1 只母鸡 3 文钱，3 只小鸡 1 文钱。如果正好花 100 文钱，可以买几只公鸡、几只母鸡、几只小鸡？

```
for (int x=0; x<=20; x++)                    // 公鸡
    for (int y=0; y<=33; y++)                // 母鸡
    {
        int z=3*(100-5*x-3*y);               // 小鸡
        if (z>=0) cout<<x<<" "<<y<<" "<<z<<endl;
    }
```

(3) 求水仙花数

【问题描述】请输出所有的水仙花数。水仙花数是一个三位数，每一位数字的立方相加，得到的总和是它本身，如 $143=1^3+4^3+3^3$。

```
for (int i=1; i<=9; i++)
    for (int j=0; j<=9; j++)
        for (int k=0; k<=9; k++)
        {
            int p=i*100+j*10+k, q=i*i*i + j*j*j + k*k*k;
            if (p==q) cout<<p<<endl;
        }
```

(4) 砝码称重

【问题描述】给你若干个 1g、2g、5g、10g、20g、50g 的砝码，数量分别为 a[1]、a[2]……a[6]，问用这些砝码可以称量出多少种重量（**重量大于** 0，**小于等于** 1000）。

```
int count=0, weight[1001], a[7];
memset(weight, 0, sizeof(weight));
```

```
for (int x1=0; x1<=a[1]; x1++)
    for (int x2=0; x2<=a[2]; x2++)
        for (int x3=0; x3<=a[3]; x3++)
            for (int x4=0; x4<=a[4]; x4++)
                for (int x5=0; x5<=a[5]; x5++)
                    for (int x6=0; x6<=a[6]; x6++)
                    {
                        int w=1*x1+2*x2+5*x3+10*x4+20*x5+50*x6;
                        weight[w]++;
                    }

for (int i=1; i<=1000; i++)
    if (weight[i]>0) count++;
cout<<count;
```

上面的代码看起来似乎很“笨拙”，但实际上它已经能顺利地解决问题了。

2.2　火柴棒等式[①]

【问题简述】给你 n（$n \leqslant 24$）根火柴棍，你可以拼出多少个形如“$A+B=C$”的等式？等式中的 A、B、C 是用火柴棍拼出的整数（**若该数非零，则最高位不能是** 0），数字的形状和电子表上的一样。

注意：

1．加号与等号各自需要两根火柴棍。

2．如果 $A \neq B$，则 $A+B=C$ 与 $B+A=C$ 视为不同的等式（A，B，$C \geqslant 0$）。

3．n 根火柴棍必须全部用上。

【分析】

直接枚举 A 和 B（**事实证明只到** 3 **位数**）。为了加快速度，事先把 222 以内各个数所用的火柴数算出来。

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

int matches[223], n;
void getmatches();          // 把火柴棍事先算好
int count=0;

int main()
{
    getmatches();

    cin>>n;
    for (int i=0;i<=111;i++)
        for (int j=0;j<=111;j++)
        {
            int k=i+j;
            if(matches[i]+matches[j]+matches[k]+4==n)
```

[①] 题目来源：NOIP2008 第二题

```
            count++;
        }
    cout<<count;
    return 0;
}

void getmatches()
{
    matches[0]=6;
    matches[1]=2;
    matches[2]=5;
    ……                          // 事先编一个程序来产生这一部分代码。
    matches[222]=15;
}
```

2.3　梵塔问题

【问题描述】已知有三根针分别用 1、2、3 表示。在一号针中从小到大放 n 个盘子，现要求把所有的盘子从 1 针全部移到 3 针。移动规则是：使用 2 针作为过渡针，每次只移动一块盘子，且每根针上不能出现大盘压小盘，找出移动次数最小的方案。

【分析】

这是一个经典的递归问题。

递归的思路：如果想把 n 个盘子放到 3 针上，就应该先把 n-1 个盘子放到 2 针上。然后把 1 针最底部的盘子移动到 3 针，再把 2 针上的 n-1 个盘子移动到 3 针上。

```
void move(int n, int a, int b, int c)          // a是盘子来源，b是暂存区、c是目标
{
    if (n==1)
        cout<<a<<"->"<<c<<endl;                // 只有一根针时直接移动
    else
    {
        move(n-1,a,c,b);                       // 先把n-1个盘子移动到#2上
        cout<<a<<"->"<<c<<endl;
        move(n-1,b,a,c);                       // 把n-1个盘子移动到#3上
    }
}
```

调用：move(n,1,2,3);

时间复杂度：O(2^n)。n 层梵塔至少要移动 2^n-1 步。

2.4　斐波那契数列

斐波那契数列为 1，1，2，3，5，8，……

很明显，斐波那契数的递推公式是 $f(n)=\begin{cases}1 & (n\leqslant 2)\\ f(n-2)+f(n-1) & (n>2)\end{cases}$

(1) 递归

```
int f(int n)
{
```

```
    if (n<=2)
        return 1;
    else
        return f(n-2)+f(n-1);
}
```

(2) 记忆化搜索

当 n 很大时，递归的运行速度会明显变慢。根本原因在于递归算法进行了大量的重复运算。所以可以开一个数组，记录每个被计算过的函数值。每次调用 f 时，如果函数值被计算过，就直接调用计算好的结果。

```
int F[MAX] = {0};                          // 初始化为0表明没有计算过
int f(int n)
{
    if (F[n]!=0) return F[n];              // 直接调用记录的值
    if (n<=2)
        return F[n]=1;
    else
        return F[n]=f(n-2)+f(n-1);         // 计算，同时记录计算结果
}
```

(3) for 循环

尽管记忆化快了很多，但仍然不是最好的。本问题递推顺序明显，用一个 for 循环就可以解决问题了。

```
int t1, t2, r;                  // int f[MAX] = {0, 1, 1};
t1=t2=r=1;
for (int i=3; i<=n; i++)
{
    r=t1+t2;                    // f[i]=f[i-2]+f[i-1];
    t1=t2, t2=r;                // 使用f后这一行代码就没有意义了。
}
// cout<<r;                     // cout<<f[n];
```

2.5　常见的递推关系！

1. **前序和**：设 S_i 表示数列中首项[①]到第 i 项的和。
   - S_i 的递推公式：$S_i=S_{i-1}+a_i$
   - 第 i 项到第 j 项中所有元素的和：$S=S_j-S_{i-1}$
2. **等差数列**
   - 递推公式：设首项为 a_0，公差为 d，则 $a_n=a_{n-1}+d$
   - 通项公式[②]：$a_n=a_0+nd$
   - 前 n 项和：$S_n=\frac{1}{2}(a_0+a_n)(n+1)=(n+1)a_0+\frac{1}{2}dn(n+1)$
3. **等比数列**
   - 递推公式：设首项为 a_0，公比为 q，则 $a_n=qa_{n-1}$
   - 通项公式：$a_n=a_0q^n$

[①] 在本节若无特殊说明，n 都是从 0 开始的，当然“n 条直线”、“n 个物品”是例外。
[②] 这里的通项公式、求和公式和高中数学的略有不同，原因是首项的下标不同。

- 前 n 项和：$S_n=\frac{a_0(1-q^{n+1})}{1-q}$

4．**斐波那契（Fibonacci）数列**

- 递推公式：$f_n=\begin{cases}1 & n=0\text{或}n=1\\ f_{n-1}+f_{n-2} & n\geqslant 2\end{cases}$
- 前 n 项和：$S_n=f_{n+2}-1$
- 通项公式（**不能用于编程！**）：$f_n=\frac{\sqrt{5}}{5}\left[\left(\frac{1+\sqrt{5}}{2}\right)^{n+1}-\left(\frac{1-\sqrt{5}}{2}\right)^{n+1}\right]$
- 实例：
  ① 楼梯有 n 阶台阶，上楼可以一步上 1 阶，也可以一步上 2 阶，计算共有多少种不同的走法。
  ② 有一对雌雄兔，每两个月就繁殖雌雄各一对兔子.问 n 个月后共有多少对兔子？
  ③ 有 $n*2$ 的一个长方形方格，用一个 $1*2$ 的骨牌铺满方格。求铺法总数。

5．**第二类 Stirling 数**

- $s(n,k)$ 表示含 n 个元素的集合划分为 k 个集合的情况数。
- **递推公式**：$s(n,k)=s(n-1,k-1)+k\cdot s(n-1,k)$，$1\leqslant k<n$

6．**错位排列**

- 示例：在书架上放有编号为 1，2，…，n 的 n 本书。现将 n 本书全部取下然后再放回去，当放回去时要求每本书都不能放在原来的位置上。求满足以上条件的放法共有多少种？
- 错位排列数列为 0，1，2，9，44，265，…
- 第一种递推公式：$d_n=\begin{cases}0 & n=1\\ 1 & n=2\\ (n-1)(d_{n-2}+d_{n-1}) & n\geqslant 3\end{cases}$
- 第二种递推公式：$d_n=\begin{cases}0 & n=1\\ nd_{n-1}+(-1)^{n-2} & n\geqslant 2\end{cases}$
- 通项公式：$d_n=n!\left[\frac{1}{0!}-\frac{1}{1!}+\frac{1}{2!}-\frac{1}{3!}+\frac{1}{4!}-\cdots+(-1)^n\frac{1}{n!}\right]$

7．**分平面的最大区域数**

- n 条直线分平面的最大区域数的序列为：2，4，7，11，…
  递推公式：$f_n=f_{n-1}+n$
  通项公式：$f_n=n(n+1)/2+1$
- n 条折线分平面的最大区域数的序列为：2，7，16，29，…
  递推公式：$f_n=f_{n-1}+4n-3$
  通项公式：$f_n=(n-1)(2n-1)+2n$
- n 条封闭曲线（**如一般位置上的圆**）分平面的最大区域数的序列为：2，4，8，14，…

  递推公式：$f_n=f_{n-1}+2(n-1)$
  通项公式：$f_n=n^2-n+2$

8．**组合数**

- 通项公式：$C_n^m=\frac{A_n^m}{m!}=\frac{n(n-1)(n-2)\cdots\cdots(n-m+1)}{m!}=\frac{n!}{m!(n-m)!}$
- 第一种递推公式：$C_n^m=C_{n-1}^{m-1}+C_{n-1}^m$　（边界条件$C_n^0=C_n^n=1$）
- 第二种递推公式：$C_n^m=\frac{n-m+1}{m}C_n^{m-1}$　（边界条件$C_n^0=1$，必须先乘后除）

  优化：$m>\frac{n}{2}$时，可利用$C_n^m=C_n^{n-m}$来减少计算次数。

9．**卡塔兰（Catalan）数列**

- 序列：1，2，5，14，42，132，429，1430，4862，16796…

- 通项公式：$h_n=\frac{C_{2n}^{n}}{n+1}$
- 第一种递推公式（**基于公式意义**）：$h_n=\begin{cases}1 & n=1\\ h_1h_{n-1}+h_2h_{n-2}+\ldots+h_{n-1}h_1 & n\geqslant 2\end{cases}$
- 第二种递推公式（**基于组合数性质**）：$h_n=\begin{cases}1 & n=1\\ \frac{2(2n-1)}{n+1}h_{n-1} & n\geqslant 2\end{cases}$
- 实例：

① 有 $2n$ 个人排成一行进入剧场。入场费 5 元。其中只有 n 个人有一张 5 元钞票，另外 n 人只有 10 元钞票，剧院无其它钞票，问有多少中方法使得只要有 10 元的人买票，售票处就有 5 元的钞票找零？

② 一位大城市的律师在她住所以北 n 个街区和以东 n 个街区处工作。每天她走 $2n$ 个街区去上班。如果他从不穿越（**但可以碰到**）从家到办公室的对角线，那么有多少条可能的道路？

③ 在圆上选择 $2n$ 个点，将这些点成对连接起来使得所得到的 n 条线段不相交的方法数？

④ n 个结点可构造多少个不同的二叉树？

⑤ 一个栈（**无穷大**）的进栈序列为 1，2，3，…n，有多少个不同的出栈序列？

⑥ 将一个凸多边形区域分成三角形区域的方法数？

⑦ 一个乘法算式 $P=a_1a_2a_3\cdots a_n$，在保证表达式合法的前提下（**某个数不会被括号括两次，如“((a))”是错误的**），有多少种添加括号的方法？

2.6　选择客栈[①]

【问题描述】

丽江河边有 n 家（$2\leqslant n\leqslant 200,000$）很有特色的客栈，客栈按照其位置顺序从 1 到 n 编号。每家客栈都按照某一种色调进行装饰（总共 k 种，用整数 0~k-1 表示，且 $0<k\leqslant 50$），且每家客栈都设有一家咖啡店，每家咖啡店均有各自的最低消费。

两位游客一起去丽江旅游，他们喜欢相同的色调，又想尝试两个不同的客栈，因此决定分别住在色调相同的两家客栈中。晚上，他们打算选择一家咖啡店喝咖啡，要求咖啡店位于两人住的两家客栈之间（**包括他们住的客栈**），且咖啡店的最低消费不超过 p（$0\leqslant p$，最低消费$\leqslant 100$）。

他们想知道总共有多少种选择住宿的方案，保证晚上可以找到一家最低消费不超过 p 元的咖啡店小聚。

【分析】

首先考虑 30%（$n\leqslant 100$）的数据。很明显，本题需要用枚举法解决。

枚举法的思路很明显：枚举区间 $[i, j]$（$i\neq j$，且 i、j 色调相同），再判断区间内是否有最低消费不超过 p 的咖啡店。时间复杂度为 $O(n^3)$。

现在开始优化算法。可以看到，判断区间内咖啡店的时间为 $O(n)$，有减少的余地。令 $c[i]$ 表示 $[1, i]$ 之间最低消费不超过 p 的咖啡店个数[②]，那么可以看出，如果 $c[j]-c[i]>0$，说明 i、j 之间有符合要求的咖啡店。这样判断区间的时间就降到了 $O(1)$。

有这一步做基础，为了使思路清晰，就可以把不同颜色的客栈分开了。

在处理 j 时，我们要统计 $1\sim j-1$ 与 j 的解的个数。处理 $j+1$ 时又要处理 $1\sim j-1$，造成了重复计算。

[①] 题目来源：NOIP2011 day1 第二题

[②] 事实上不是咖啡店个数。如果 i、j 并列出现“√×”的情况，那么 $c[j]-c[i]$ 也应该大于 0。

设 sum[j]为[1，j]到[i，j]中解的个数。如果j+1 无法消费，那么由于j 的存在，有 sum[j+1]＝sum[j]；如果j+1 可以消费，那么j+1 可以和 1 到j 中任何一个客栈组合，那么 sum[j+1]＝j。

最后将每种颜色的所有 sum 相加，就可以在 O(n)的时间内得出答案了。

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

int n,k,p;
int c[51][200001], top[51];
long long total=0;

int main()
{
   memset(top,-1,sizeof(top));
   memset(c,0,sizeof(c));

   ios::sync_with_stdio(false);
   cin>>n>>k>>p;

   int cnt=0;
   bool P=false;
   for (int i=0; i<n; i++)
   {
      int x,y;
      cin>>x>>y;

      int t=++top[x];
      if (y<=p || P)
         cnt++;
      c[x][t]=cnt;

      P = y<=p;
   }

   for (int color=0; color<k; color++)
   {
      int prev=c[color][0];
      long long sum=0;

      for (int i=0; i<=top[color]; i++)
      {
         if (c[color][i]-prev>0) sum=i;
         total+=sum;
         prev=c[color][i];
      }
   }

   cout<<total<<endl;
```

```
    return 0;
}
```

2.7　2^k进制数[①]

【问题描述】

设 r 是个 2^k 进制数（$1 \leqslant k \leqslant 9$），并满足以下条件：

（1）r 至少是个 2 位的 2^k 进制数。

（2）作为 2^k 进制数，除最后一位外，r 的每一位严格小于它右边相邻的一位。

（3）将 r 转换为 2 进制数 q 后，则 q 的总位数不超过 w（$k < w \leqslant 30000$）。

输入 k 和 W，请满足上述条件的不同 r 的数量。

【分析】

本题不可能直接枚举——答案不超过 200 位，说明用到了高精度算法！

先不考虑 w。对于一个 m 位数来讲，m 个数字的顺序是确定的。所以只需直接从 n（$n=2^k-1$）个数中取 m 个数，组成 m 位数。所以符合条件的 m 位数有C_m^n个。

现在考虑 w 存在的情况。这里有一个明显的事实：2^k 进制数，除首位外，每一位转化为二进制后都是 k 个二进制位。现在最大的问题就是 2^k 进制数的首位。

二进制不超过 w 位的数，在 2^k 进制下就是 $a=[w/k]+1$ 位（w 能被 k 整除时，认为首位有 0 个数字）。此时首位剩下 $b=w\%k$ 个二进制位，即最大可以取 2^b-1。接下来只需针对 2^k 进制下位数不足 a 和恰好为 a 两种情况分别进行枚举。

所以，符合条件的 2^k 进制数个数为 $f(k,w)=\begin{cases}\sum_{i=2}^{n} C_n^i & w \geqslant kn \\ \sum_{i=2}^{a-1} C_n^i + \sum_{i=1}^{2^b-1} C_{n-i}^a & k < w < kn\end{cases}$

其中 $n=2^k-1$，$a=[w/k]+1$，$b=w \bmod k$。

通过排列组合，我们间接地枚举了所有符合条件的解。

2.8　Healthy Holsteins[②]

【问题描述】

农民约翰以他有世界上最健康的奶牛为骄傲。他知道每种饲料的各种维生素的含量，以及一头牛每天需要最少的维生素的量。请你帮助约翰喂养这些奶牛，使得它们能够保持健康，并且消耗的饲料种类最少。

【输入格式】

第一行：一个数 V（$1 \leqslant V \leqslant 25$），表示维生素的种类数。

第二行：V 个整数，表示一头牛一天需要的维生素量。一个整数对应一种维生素。

第三行：一个数 G（$1 \leqslant G \leqslant 15$），表示饲料的种类数。

第四行：G 个整数，表示每种饲料的各种维生素的含量。

【输出格式】

输出共一行。第一个数是需要的最少的饲料种类数。从第二个数开始，输出所需的饲料种类序号。请输出字典序最小的解。

【分析】

① 题目来源：NOIP2006 第四题

② 题目来源：USACO Training Gateway

本题最多只有 15 种饲料。如果尝试所有取法，也只有 $2^{15}=32768$ 种状态，因此完全可以枚举所有方案。

考虑一种饲料，它要么用来喂牛，要么扔到一边去。用来喂牛可以用“1”表示，放到一边用“0”表示。于是，G 种饲料的状态可以转化成一个 G 位的二进制数。这样，枚举 0 到 2^G-1 的数，就枚举了所有状态。

当然这只是一种思路。如果不能熟练使用位运算，这种做法的编程复杂度会比 DFS 高很多。

【代码】

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

int v, req[26];
int g, vitamin[16][26];
int Min=100, minstat=0;        // Min表示需要最少饲料数，minstat表示目前的最优状态

int count(int stat)            // 统计stat中“1”的个数
{
    int r=0;
    while (stat)
    {
        r+=stat & 1;
        stat>>=1;
    }
    return r;
}

int main()
{
    cin>>v;
    for (int i=1; i<=v; i++) cin>>req[i];

    cin>>g;
    for (int i=1; i<=g; i++)
        for (int j=1; j<=v; j++)
            cin>>vitamin[i][j];

    for (int stat=0; stat < (1<<g); stat++)                // 尝试所有组合
    {
        int r[26] = {0};

        for (int i=1; i<=g; i++)                           // 计算组合中每种维生素的量
            if (stat & (1<<(i-1)))
                for (int j=1; j<=v; j++) r[j]+=vitamin[i][j];

        bool no=false;
        for (int i=1; i<=v; i++) if (r[i]<req[i]) no=true; // 判断维生素是否达标
        if (no) continue;

        int c=count(stat);
        if (c<Min && c>0) Min=c, minstat=stat;
```

```
    }

    cout<<Min<<" ";
    for (int i=1; i<=g; i++) if (minstat & (1<<(i-1))) cout<<i<<" ";
    return 0;
}
```

2.9　小结

本单元总结了三种基本算法：枚举、递归和递推。

1．枚举

所谓枚举，就是将所有的可能一一遍历出来，在遍历中逐个检查答案的正确性，从而确定最终结果。因此，枚举法适合解决“是否有解”和“有多少个解”的问题。搜索的本质就是枚举。

应用枚举法搜索答案之前，要确定枚举的对象和状态的表示方式。

常见的枚举方法有三种：第一种是使用 for 循环，第二种是利用排列组合，第三种是顺序化（比如把状态与二进制数对应）。

由于枚举法产生了大量的无用解，所以规模稍大的时候要进行优化。优化时要从两方面入手，一是减少状态量，二是降低单个状态的考察代价。优化过程要从以下几个方面考虑：枚举对象的选取、枚举方法的确定、采用局部枚举或引进其他算法。

（在这里附上次短路的求法：先求出最短路，枚举最短路上的每一条边，对于每一条边都删除然后重新求最短路。几个新的“最短路”的最小值就是次短路。）

2．递归

递归，即自己调用自己。递归法将较复杂的大问题分解成较简单的小问题，然后一层一层地得出最终答案。

递归是实现许多重要算法的基础。深度优先搜索、记忆化搜索、分治算法和输出动态规划的中间过程等都需要用递归实现。

有一些数据结构也是递归定义的，如二叉树。使用此类数据结构时，也要递归地思考问题。

写递归要从两方面入手：分解问题的方式、边界条件。

注意，没有边界条件会引发无限递归，从而导致程序崩溃！

3．递推

递推指利用数列中前面一项或几项，通过递推公式来求出当前项的解。

递推与递归不同。一般情况下，递推用循环语句完成，而递归必须定义函数。递推也可以转化为递归，即记忆化搜索，这常用于递推顺序不明显的情况。

解决递推问题要从以下几方面入手：状态表示、递推关系、边界条件。

按照计算顺序，递推可以分为顺退和逆推。对于某些问题，递推顺序甚至直接影响编码的复杂度。

动态规划必须使用递推或记忆化搜索来解决。动态规划的状态转移方程一定是递推式。但动态规划与纯粹的递推不同，前者有明显的阶段，而后者的数学性更强。

我们已经在数学课上学习了数列的通项公式和递推公式。高中数学喜欢通项公式，而信息学更喜欢递推公式。下次遇到类似问题，就去寻找递推式吧。

4．递归和递推的比较

	递归	递推

适合解决的问题	1．问题本身是递归定义的，或者问题所涉及到的数据结构是递归定义的，或者是问题的解决方法是递归形式的。 2．需要利用分治思想解决的问题。 3．回溯、深度优先搜索 4．输出动态规划的中间过程。	1．能够用递推式计算的数学题。 2．动态规划（**必须使用递推或记忆化搜索**）
特点	结构清晰、可读性强 目的性强	速度较快、比较灵活 思路不易想到
编码方式	在函数中调用自己	迭代（**使用** `for` **循环等**）
替代方法	问题的性质不同，改写的方法也不同。 ① 有的问题可以根据程序本身的特点，使用栈来模拟递归调用。 ② 有的问题可以改写成递推或迭代算法。	在拓扑关系不明显时，可以采用记忆化搜索。