# 常见算法

# A：5534 拆分一个背包为logn个

# B：3584 多维树状数组

# C：蔡勒（Zeller）公式（计算星期）

蔡勒（Zeller）公式：是一个计算星期的公式。

随便给一个日期，就能用这个公式推算出是星期几。蔡勒公式如下：

w=y+[y/4]+[c/4]-2c+[26(m+1)/10]+d-1

公式中的符号含义如下：

w：星期； w对7取模得：0-星期日，1-星期一，2-星期二，3-星期三，4-星期四，5-星期五，6-星期六

c：世纪（前两位数）

y：年（后两位数）

m：月（m大于等于3，小于等于14，即在蔡勒公式中，某年的1、2月要看作上一年的13、14月来计算，比如2003年1月1日要看作2002年的13月1日来计算）

d：日

[ ]代表取整，即只要整数部分。

# D：1986 LCA,读入优化

# E：Bit set

自己看懂下面的bitset类模版的操作表即可

表3-7  **bitset**操作

|  |  |
| --- | --- |
| b.any() | b中是否存在置为1的二进制位？ |
| b.none() | b中不存在置为1的二进制位吗？ |
| b.count() | b中置为1的二进制位的个数 |
| b.size() | b中二进制位的个数 |
| b[pos] | 访问b中在pos处的二进制位 |
| b.test(pos) | b中在pos处的二进制位是否为1？ |
| b.set() | 把b中所有二进制位都置为1 |
| b.set(pos) | 把b中在pos处的二进制位置为1 |
| b.reset() | 把b中所有二进制位都置为0 |
| b.reset(pos) | 把b中在pos处的二进制位置为0 |
| b.flip() | 把b中所有二进制位逐位取反 |
| b.flip(pos) | 把b中在pos处的二进制位取反 |
| b.to\_ulong() | 用b中同样的二进制位返回一个unsigned long值 |
| os << b | 把b中的位集输出到os流 |

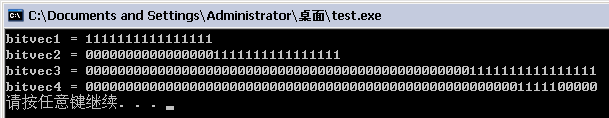
赋初值

bitset<16> bitvec1(0xFFFF);

bitset<32> bitvec2(0xFFFF);

bitset<64> bitvec3(0xFFFF);

bitset<64> bitvec4(string("1111100000"));



或者 scanf("%x“,&n) ;//%x 为16进制，%o为8进制

bitset<32>q(n);//初始化（只支持n为8，16进制数）

或者 string s="00011";

bitset<32>q(s);

# F:康托展开

如果已知{1,2,3,4}的一个排列{2,1,4,3}

想要计算排在他前面的排列数目，则我们只用将问题转换为求首位小于2的所有排列，首位为2第二位小于1的所有排列，前两位为21第3位小于4的所有排列，前3位为214第4位小于3的所有排列的和即可。

需要注意的是，在考虑后续的排列时前面已经确定的集合不能作为候选对象。

康托展开

把一个整数X展开成如下形式：  
X=a[n] x n!+a[n-1] x (n-1)!+...+a[2] x 2!+a[1] x 1!  
其中，a为整数，并且0<=a<i,i=1,2,..,n

例如{1,2,3,4,...,n}表示1,2,3,...,n的排列如{1,2,3} 按从小到大排列一共6个。123 132 213 231 312 321 。代表的数字 1 2 3 4 5 6 也就是把10进制数与一个排列对应起来。他们间的对应关系可由康托展开来找到。如我想知道321是{1,2,3}中第几个大的数可以这样考虑 ：

第一位是3，当第一位的数小于3时，那排列数小于321 如 123、 213 ，小于3的数有1、2 。所以有2 x 2!个。

再看小于第二位2的：小于2的数只有一个就是1 ，所以有1 x 1!=1 所以小于321的{1,2,3}排列数有2 x 2!+1 x 1!=5个。

所以321是第6个大的数。 2 x 2!+1 x 1!是康托展开。

再举个例子：1324是{1,2,3,4}排列数中第几个大的数：

第一位是1小于1的数没有，是0个 0 x 3!

第二位是3小于3的数有1和2，但1已经在第一位了，所以只有一个数2, 1 x 2! 。

第三位是2小于2的数是1，但1在第一位，所以有0个数 0 x 1!

所以比1324小的排列有0 x 3!+1 x 2!+0 x 1!=2个，1324是第三个大数

逆康托展开

{1,2,3,4,5}的全排列，并且已经从小到大排序完毕找出第96个数

首先用96-1得到95

用95去除4! 得到3余23

用23去除3! 得到3余5

用5去除2!得到2余1

用1去除1!得到1余0有3个数比它小的数是4

所以第一位是4,有3个数比它小的数是4但4已经在之前出现过了所以是5（因为4在之前出现过了所以实际比5小的数是3个）

有2个数比它小的数是3

有1个数比它小的数是2

# G读入优化

**inline char** nc**(){**

**static char** buf**[**100000**],\***p1**=**buf**,\***p2**=**buf**;**

**return** p1**==**p2**&&(**p2**=(**p1**=**buf**)+**fread**(**buf**,**1**,**100000**,**stdin**),**p1**==**p2**)?**'@'**:\***p1**++;**

**}**

**int** over**;**

**inline int** \_read**(){**

**char** ch**=**nc**();int** s**=**0**;**

**while(!(**ch**>=**'0'**&&**ch**<=**'9'**)){**

ch**=**nc**();**

**if(**ch**==**'@'**) {**over**=**1**;break;}**

**}**

**while(**ch**>=**'0'**&&**ch**<=**'9'**){**

s**=**s**\***10**+**ch**-**48**,**ch**=**nc**();**

**if(**ch**==**'@'**) {**over**=**1**;break;}**

**}**

**return** s**;**

**}**

# H:悬线法

黑白棋盘可以通过将所有i+j为奇数的格子反转，  
这样，问题就转换成了求一个最大的0/1子矩阵

那么怎么O（n^2）求一个最大全零矩阵呢，悬线法如下。  
先考虑一维的情况，h[i]表示以i为终点的最长连续0的长度，有h[i]=a[i]==0? h[i-1]+1:0,这样可以O（n）轻松求出。  
拓展到高维，首先同样按照一维的方法，h[i][j]表示第i行以j为终点的最长连续0的长度，预处理出h[]。。接下来考虑一列一列来更新答案，对于单独的一列i，若以h[i][j]为子矩阵的一个边长，则它能往上下扩展的最大长度len就是另一个边长，所谓扩展就是向一个方向扫描知道碰到h值比自己小位置。  
如果暴力扩展，复杂度就是O(n3)不能满足要求。  
实际上，我们想要扩展，只是想要得到这样一个信息：  
对于行i，其分别能够向上/向下扩展多少行？  
只要我们知道这个信息，结合h[i]就可以求出一个矩形用于更新答案了。  
我们考察单调栈：  
对于一个新元素，如果比栈顶元素h小，说明这个元素就是扩展的下界，马上弹出栈顶元素进行计算即可。  
当一个新元素插入栈时，容易知道，新元素一定比栈顶元素大，那么栈顶元素就是他的上界，这样我们就得到了上界。

Bzoj1057

# I:树状数组 1082 codeves

区间操作的树状数组

# J:string操作

string ss[M];

char sss[M];

cin>>ss[i];

L+=ss[i].size();

strcpy(sss, ss[i].c\_str());

# K:解亦或方程组1830