##### 位运算法

我们使用位运算进行优化，即如果取第i个数以第i位赋值为1表示，否则为0表示，则输入5 3的结果对应于二进制的位表示将如图2.26所示。

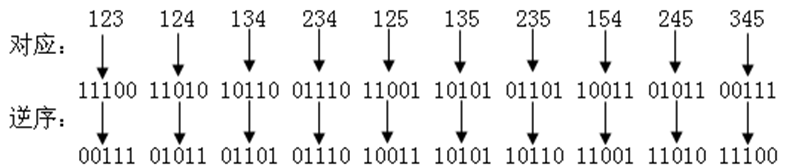


图2.26

可以看出，最后逆序产生的二进制数，每个数均有3个1，且数值依次增大。因此问题转化成了当获得一个组合数K，如何获得一个比K大，并且二进制下有相同个数的“1”，并且是其中最小的数的问题。例如当K＝00111时，如何获得01011呢？

事实上，只要先找出K右起第一次出现1的位置，对这个位置加1，然后再右边补上少了的1就可以了，例如00111，右起第一位出现1的位置是1，则在此位置上加1变为01000，再在右边补充少了的两个1变为01011即可，同理，01110也可以此法找到下一个数10011，如图2.27所示。



图2.27

找到右起第一次出现1的位置的算法很简单，设整型变量first1保存组合数combin右起第一次出现1的位置，则first1＝(combin & -combin)。在此位置上加1，即combin＋first1。

那么右边补充1的个数，实际等于原数右起第一个1向左数，连续的1的个数减1，可以使用((combin＋first1)^combin)>>2即右移两位来获得，例如combin＝00111，对它＋1的结果为01000，则01000^00111＝1111，1111>>2＝11，即应补上的值为11。

不过，在当combin最右边不是1，有若干个0的时候，前一个步骤得到的数的最右边，就会有同样多的0，去掉多余0的方法即是除以(combin & －combin)。完整的计算过程如图2.28所示。

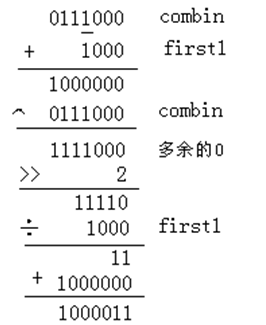


图2.28

即下一个组合数为((((combin＋first1)^combin)>>2)/first1)＋(combin＋first1)。

参考程序如下所示。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | //组合问题 — 位运算  #include <bits/stdc++.h>  using namespace std;  int Next\_combination(int combin,int n) //根据前一组合枚举下一组合  {  int next; //下一个组合  int first1 = combin & -combin; //找到右起第一个出现1的位置  int t=(combin+first1); //在first1的位置上加1  next=(((t^combin)>>2)/first1) | t;  if ((1 << n) < next) //如果越界，则退出  return 0;  return next;  }  int main()  {  int M,N;  scanf("%d%d",&M,&N);  if(N==0)  {  printf("\n");  exit(0); //直接退出程序，返回0值  }  int combin=(1<<N)-1; //初始化为100...000-1=111...1111，即N个1  do  {  for (int i = 0; i<M; ++i) //输出组合  if ((combin & (1 << i))>0) //如果第i位为1,则表示组合中有第i个数  cout<<i+1; //因为下标从0开始，所以输出i+1  cout<<endl;  }while(combin=Next\_combination(combin,M));  return 0;  } |