前记：昨天在图书馆中阅读《挑战程序设计大赛》中的一个问题：从一堆数据中选择三个数据使它们能够组成一个三角形。作者给了一个穷举法，但是算法的效率不高。这个问题明显就是一个生成组合数的问题，所以我开始研究如何生成组合数，我在网上搜索了不少资料，下面这篇博文虽然内容不是关于如何生成组合数，而是关于如何计算组合数的，但是其中使用到的算法是非常有借鉴和学习意义的，所以特地将它编辑一番。

求组合数可能不是在真正编程中经常用到的东西，不过ACM啊，纯数学运算以及HARD-CORE PROGRAMMING还是经常会碰到的。我们用

C(n,r)

来表示组合数，代表从n个不同小球里取出r个小球的取法。

计算组合数有几个办法，说说我知道的吧。

# 方法一：公式法

第一种根据公式而来。在数学中，

C(n,r) = n\*(n-1)\*(n-2)\*...\*(n-r+1)/r!

这是直接运算的公式，写成代码的话循环r次就可以得到分子分母，然后相除。这种方法的毛病就是求分子分母的数可能太大了，计算时浪费了很多内存。当然也可先除，再乘起来，不过如果恰好对应的两个数不能整除就麻烦了（在python有fraction模块不用担心浮点数的问题），只能约分HCF，然后用分数表示。总之这种办法在大数上不灵的，小的计算还是不错。

# 方法二：递归法

第二种是最常见的，Google一下很多人还是用这种方法来做，也就是用递归了，

C(n,r)=C(n-1,r)+C(n-1,r-1)

在n==r或r==0返回1就行了。这种方法在处理大数方面也有很大问题，递归层数多了也是特别浪费内存的东西，如果我要计算C(2000,1000)可以看看需要多久。

# 方法三：动态规划算法

第三种办法，也就是我刚刚学到的一种方法。其实之前在Project Euler中有类似的问题（76)用的近乎一致的解法，虽然当时我根本看不懂（现在也不太懂里面的数学部分），不过看过这种求组合数的办法后，好像有点感觉了。

（很显然，上面的递归公式是非常适合使用动态规划算法来实现的，在这篇博客中所使用的这两种实现方式非常好地揭示了动态规划算法和递归算法之间的关联，也提示了我，在对递归算法进行优化的时候，使用动态规划算法是一个非常好的思路）

这种办法是从第二种演变而来的。

下面以求C(5,2)为例，

C(0,0)-C(1,1)-C(2,2)

C(1,0)-C(2,1)-C(3,2)

C(2,0)-C(3,1)-C(4,2)-C(5，3)

C(3,0)-C(4,1)-C(5,2)-C(6，3)

在这个图中，每个位置上组合数的值都等于其左边组合数和其上方组合数值之和。根据这一点，只要第一排和第一列的值知道，就可以求出剩下所有的值。恰好他们都等等于1。

于是就有

#include <stdio.h>

/\*

 Compute C(n,r)

 \*/

unsigned long cnr(int,int);

int main(void)

{

  printf("%d",cnr(2000,1000));

  return 0;

}

unsigned long cnr(int n, int r)

{

  unsigned long c[r+1];

  int i,j;

  for(i=0;i<=r;i++) c[r] = 1UL; /\*Initialize\*/

  for(i=1;i<=n-r;i++)

    for(j=1;j<=r;j++)

      c[j] = c[j]+c[j-1];

  return c[r];

}

在这个过程中，始终只用了c[r]这么大的一个数组，反复在其中进行运算，避免了递归浪费内存。其实这种想法在一些求Fibonacci数列的介绍中经常出现。只定义两个变量 unsigned long m, unsigned long n 反复对他们求和（m+=n; n+=m)，求得的值保存在m,n中这样就可以打印这个数列，而不是用递归的方式去做。

这种编程方法应该叫做动态编程(Dynamic Programming)吧(我是小白，其实印象在哪看过，所以猜测就是这个意思）。在求组合数的过程中，是纯加法实现，而且只要要求的数不是大到必须溢出，都可以求。

我自己也“照虎画猫”写了一个判断两个数是否互质的函数。一般大家都知道用Euclid公式，求余数，最后看最大公约数是否大于1可以判断（这也可以用DP来做，比其递归，我跟喜欢DP)，但是还有种不求最大公约数的办法。

#include <stdio.h>

#define TRUE 1

#define FALSE 0

int bprime(int, int);

int main(void)

{

  int a=bprime(84,28);

  int b=bprime(10,5);

  printf ("%d %d\n",a,b);

}

int bprime(int a, int b)

{

  if(a<b)

    {

      printf("the first number must be larger than the second.\n");

      return 2;

    }

  int c;

  while(b>0)

    {

      c=a-b;

      c>=b ? a=c : (a=b,b=c);

      if(b == 1) return TRUE;

    }

  return FALSE;

}

我的代码写的丑 T \_ T.   里面的数学就不说了(没有一开始就判断两个是不是都是偶数，因为如果都是偶数请用眼睛判断），没用递归，也没求公约数，且减法实现，不过对错...我觉得是对的。（刚看了百度百科，这个方法应该就是里面说的更相减损法的变化版本，好吧，我弱爆了）

对于DP的学习我还要继续，如果有好心人愿意收留我，请告诉我一声。

Here is a good article talking about DP: http://hawstein.com/posts/dp-novice-to-advanced.html