# 组合数的各种方法总结详解

问题描述：找出从自然数1、2、... 、n中任取r个数的所有组合。例如n=5，r=3的所有组合为:

1,2,3  
1,2,4   
1,3,4   
2,3,4   
1,2,5   
1,3,5   
2,3,5   
1,4,5   
2,4,5   
3,4,5

**用程序实现有几种方法:**

## ****1)穷举法****

程序如下  
【程序】  
#include<stdio.h>  
const int n=5,r=3;  
int    i,j,k,counts=0;

int main()  
{  
     for(i=1;i<=r ;i++)  
        for(j=i+1;j<=r+1;j++)  
            for( k=j+1;k<=r+2;k++){  
               counts++;  
               printf("%4d%4d%4d/n",i,j,k);  
           }  
printf("%d",counts);  
return 0;  
}  
但是这个程序都有一个问题，当r变化时，循环重数改变，这就影响了这一问题的解，即没有一般性。

## 2)递归法

分析所列的10个组合，可以采用这样的递归思想来考虑求组合函数的算法。  
设函数为void    comb(int m, int k)为找出从自然数1、2、... 、m中任取k个数的所有组  
合。当组合的第一个数字选定时，其后的数字是从余下的m-1个数中取k-1数的组合。这  
就将求m个数中取k个数的组合问题转化成求m-1个数中取k-1个数的组合问题。设函数引入工作数组a[ ]存放求出的组合的数字，约定函数将确定的k个数字组合的第一个数字放  
在a[k]中，当一个组合求出后，才将a[ ]中的一个组合输出。第一个数可以是m、m-1、  
...、k，函数将确定组合的第一个数字放入数组后，有两种可能的选择，因还未确定组  
合的其余元素，继续递归去确定；或因已确定了组合的全部元素，输出这个组合。细节  
见以下程序中的函数comb。  
【程序】  
#include <time.h>  
#include <iostream>

using namespace std;

# define      MAXN      100  
int a[MAXN];  
int counts=0;

void printtime(void) //打印当前时间的函数   
{   
      char tmpbuf[128];   
      time\_t ltime;   
      struct tm \*today;

      time(<ime);   
      today = localtime(<ime );   
      strftime(tmpbuf,128,"%Y-%m-%d %H:%M:%S",today);   
      cout<<tmpbuf<<endl;   
}

void      comb(int m,int k)  
{     int i,j;  
      for (i=m; i>=k; i--) // 假定原来的序列是固定的，采用倒序的方式选取数据。  
      {   a[k]=i; // 第一个选取的数据是最后一个（即第K个）。  
          if (k>1) // 如果还没完成筛选，继续递归  
              comb(i-1,k-1);  
          else // 如果完成筛选，将选择好的组合存储（或者其他显示输出处理）。  
          {     
              counts++;  
              /\*  
              for (j=k; j>0 ;j--)  
                  printf("%4d",a[j]);  
              printf("/n");  
              \*/  
          }  
      }  
}

int main()  
{

      int m,r;  
      cout<<"m"<<endl;  
      cin>>m;  
      cout<<"r"<<endl;  
      cin>>r;  
      counts=0;  
      a[0]=r;  
      printtime();  
      comb(m,r);  
      cout<<counts<<endl;  
      printtime();  
      return 0;  
}

这是我在网上找到的程序，稍微修改了一下。程序写的很简洁，也具有通用性，解决了问题。

## ****3)回溯法****

采用回溯法找问题的解，将找到的组合以从小到大顺序存于a[0]，a[1]，…，a[r-1]  
中，组合的元素满足以下性质：

（1）     a[i+1]>a[i]，后一个数字比前一个大；  
（2）     a[i]-i<=n-r+1。  
按回溯法的思想，找解过程可以叙述如下：  
      首先放弃组合数个数为r的条件，候选组合从只有一个数字1开始。因该候选  
解满足除问题规模之外的全部条件，扩大其规模，并使其满足上述条件（1），候选组合  
改为1，2。继续这一过程，得到候选组合1，2，3。该候选解满足包括问题规模在内的全  
部条件，因而是一个解。在该解的基础上，选下一个候选解，因a[2]上的3调整为4，以  
及以后调整为5都满足问题的全部要求，得到解1，2，4和1，2，5。由于对5不能再作调  
整，就要从a[2]回溯到a[1]，这时，a[1]=2，可以调整为3，并向前试探，得到解1，3，  
4。重复上述向前试探和向后回溯，直至要从a[0]再回溯时，说明已经找完问题的全部  
解。

在网上我始终没有找到可以正常执行的完整程序，所以我只好花了一天的时间来自己来写这个程序，并且改变输出从0开始而不是从1开始，这样做的目的是 为了扩展程序的用途，适应c/c++语言的需要，这样输出就可以当作要选择的组合数组的地址序列，可以对长度为n任意类型数组找出r个组合。我对它进行了 优化，如果你认为还有可以优化的地方，请不惜赐教,。^\_^

#include <time.h>   
#include <iostream>  
#include <iomanip>  
using namespace std;

# define      MAXN      100  
int a[MAXN]; //定位数组,用于指示选取元素集合数组的位置，选取元素集合数组0 起始  
void comb(int m,int r)  
{     
      int cur;//指示定位数组中哪个成员正在移进

      unsigned int count=0;

      //初始化定位数组，0 起始的位置 ，开始的选择必是位置 0，1，2  
      for(int i=0;i<r;i++)  
          a[i]=i;

      cur=r-1;//当前是最后一个成员要移进

       do{  
          if (a[cur]-cur<=m-r ){

              count++;  
              /\*  
              for (int j=0;j<r;j++)  
                  cout<<setw(4)<<a[j];  
              cout<<endl;  
              \*/  
              a[cur]++;  
  
              continue;  
          }  
          else{  
              if (cur==0){  
                  cout<<count<<endl;  
                  break;  
              }

              a[--cur]++;  
              for(int i=1;i<r-cur;i++){  
                  a[cur+i]=a[cur]+i;  
              }

              if(a[cur]-cur<m-r)  
                  cur=r-1;                  
          }  
      }while (1);  
}

void printtime(void) //打印当前时间的函数   
{   
      char tmpbuf[128];   
      time\_t ltime;   
      struct tm \*today;

      time(<ime);   
      today = localtime(<ime );   
      strftime(tmpbuf,128,"%Y-%m-%d %H:%M:%S",today);   
      cout<<tmpbuf<<endl;   
}

int main (int argc, char \*argv[])  
{

      int m,r;  
      cout<<"m"<<endl;  
      cin>>m;  
      cout<<"r"<<endl;  
      cin>>r;  
      printtime();  
      comb(m,r);      
      printtime();  
      return(0);  
}

同上面的递归的程序进行比较，同样用g++ o2优化。当n=40，r=11，屏蔽掉输出，得到的结果都是2311801440项，递归程序用了23至24秒，回溯用了19至20秒。

## ****4)利用数组****

  定义：从n个数中取出m个数的组合。  
  实现机理：先创建一个字符串数组，其下标表示 1 到 n 个数，数组元素的值为1表示其下标代表的数被选中，为0则没选中。       
    然后初始化，将数组前 m 个元素置 1，表示第一个组合为前 m 个数。       
    然后从左到右扫描数组元素值的 10 组合，找到第一个 "10" 后交换 1 和 0 的位置,变为 01，而后将该10组合前的1和0重新组合（1放在前边，其个数为10组合前1的个数，0放在后边，其个数为10前0的个数，而后接10的倒转组合 01）。当m 个 1 全部移动到最右端时，就得到了最后一个组合。       
    例如求 5 中选 3 的组合：       
    1     1     1     0     0     //1,2,3       
    1     1     0     1     0     //1,2,4       
    1     0     1     1     0     //1,3,4       
    0     1     1     1     0     //2,3,4       
    1     1     0     0     1     //1,2,5       
    1     0     1     0     1     //1,3,5       
    0     1     1     0     1     //2,3,5       
    1     0     0     1     1     //1,4,5       
    0     1     0     1     1     //2,4,5       
    0     0     1     1     1     //3,4,5

# 全排列代码实现

using namespace std;

#include<vector>

#include<time.h>

class Solution {

public:

vector<vector<int>> permute(vector<int> &nums) {

// write your code here

vector<vector<int>> out;

//考虑使用递归方法，将递归函数独立出来

permut(nums, nums.size(), 0, out);

return out;

}

private:

/\* --------------------------------------------------------------------------------

正序倒排递归：

考虑从0开始进行全排列，我们依次选取N个元素（N为最大元素的个数），这样我们必须假设最开始

的数据顺序是固定的，我们必须基于这个固定的顺序上进行选取，否则会混乱。第一次选取第一个，

第二次选取第二个... ... 以此类推，我们能一直选取到最后一个元素：所以递归中退出的条件为：

当前选取元素的下标为最后一个元素，我们就退出，将当前序列记录下来，就是我们该次的全排列。

那么考虑中间的任何一个情况，既有：假设这个元素是当前第M个元素（M<=N），第M个元素必须要、

做的是事情分为：1）他需要和后面未构成全排列内容的元素进行位置互换，构成新的可能性。在每、

一种可能性里面，需要对M+1个元素进行全排列（递归）,针对第M个元素做完了交换和M+1的全排列

之后，我们需要将改元素和原来交换的元素恢复回来，也就是重新交换，这样能保证列表的顺序，

因为我们的排列的最基本的条件之一就是该序列表顺序是给定的。一次类推，全排列就出来了。

-------------------------------------------------------------------------------

将数组分成两个部分，0-start部分位已经选择好的区间，start-end为待选择的区间

每次从待选择的区间选择一个加入，然后递归下一个回合；

-----------------------------------------------------------------------------------\*/

int permut(vector<int> &in, int size, int start, vector<vector<int>> &out) {

if (size <= start) {

out.push\_back(in);

return 0;

}

//核心区块

for (int i = start; i < size; i++) {

swap(in[i], in[start]);// 这个交换是为了获取新数据

permut(in, size, start + 1, out);

// 这个交换是为了恢复到原来的数据序列，因为我们的前提是数据序列是固定的

// 在固定的顺序基础上得到不同的排列过。

swap(in[i], in[start]);

}

return 0;

}

void swap(int &a, int &b) {

int tmp = a;

a = b;

b = tmp;

return;

}

};

int main() {

Solution s = Solution();

vector<int> in = {0,1};

s.permute(in);

return 0;

}

# 部分排列实现