第五章 搜索与回溯算法

搜索与回溯是计算机解题中常用的算法，很多问题无法根据某种确定的计算法则来求解，可以利用搜索与回溯的技术求解。回溯是搜索算法中的一种控制策略。它的基本思想是：为了求得问题的解，先选择某一种可能情况向前探索，在探索过程中，一旦发现原来的选择是错误的，就退回一步重新选择，继续向前探索，如此反复进行，直至得到解或证明无解。

如迷宫问题：进入迷宫后，先随意选择一个前进方向，一步步向前试探前进,如果碰到死胡同，说明前进方向已无路可走，这时，首先看其它方向是否还有路可走，如果有路可走，则沿该方向再向前试探；如果已无路可走，则返回一步，再看其它方向是否还有路可走；如果有路可走，则沿该方向再向前试探。按此原则不断搜索回溯再搜索，直到找到新的出路或从原路返回入口处无解为止。

递归回溯法算法框架[一]

```
int Search(int k)
   {
   for (i=1;i<=算符种数;i++)
      if (满足条件)
         {
            保存结果
            if (到目的地) 输出解;
                  else Search(k+1);
            恢复：保存结果之前的状态{回溯一步}
         }
   }
```

递归回溯法算法框架[二]

```
int Search(int k)
   {
     if  (到目的地) 输出解;
         else
            for (i=1;i<=算符种数;i++)
               if  (满足条件)
                     {
                        保存结果;
                        Search(k+1);
                        恢复：保存结果之前的状态{回溯一步}
                     }
   }
```

【例1】素数环:从1到20这20个数摆成一个环，要求相邻的两个数的和是一个素数。

【算法分析】

非常明显，这是一道回溯的题目。从1开始，每个空位有20种可能，只要填进去的数合法：与前面的数不相同；与左边相邻的数的和是一个素数。第20个数还要判断和第1个数的和是否素数。

【算法流程】

1、数据初始化；　2、递归填数：判断第i个数填入是否合法；

A、如果合法：填数；判断是否到达目标（20个已填完）：是，打印结果；不是，递归填下一个；

B、如果不合法：选择下一种可能；

【参考程序】

```
#include<cstdio>
#include<iostream>
#include<cstdlib>
#include<cmath>
using namespace std;
bool b[21]={0};
int total=0,a[21]={0};
int search(int);                    //回溯过程
int print();                        //输出方案
bool pd(int,int);                   //判断素数
```

```
int main()
{
   search(1);
   cout<<total<<endl;                  //输出总方案数
}
int search(int t)
{
   int i;
   for (i=1;i<=20;i++)                 //有20个数可选
    if (pd(a[t-1],i)&&(!b[i]))         //判断与前一个数是否构成素数及该数是否可用
     {
        a[t]=i;
        b[i]=1;
        if (t==20) { if (pd(a[20],a[1])) print();}
           else search(t+1);
        b[i]=0;
     }
}
int print()
{
  total++;
  cout<<"<"<<total<<">";
  for (int j=1;j<=20;j++)
    cout<<a[j]<<" ";
  cout<<endl;
 }
bool pd(int x,int y)
{
   int k=2,i=x+y;
   while (k<=sqrt(i)&&i%k!=0) k++;
   if (k>sqrt(i)) return 1;
      else return 0;
}
```

【例2】设有n个整数的集合｛1,2,…,n｝，从中取出任意r个数进行排列（r<n），试列出所有的排列。

```
#include<cstdio>
#include<iostream>
#include<iomanip>
using namespace std;
int num=0,a[10001]={0},n,r;
bool b[10001]={0};
int search(int);                          //回溯过程
int print();                              //输出方案

int main()
{
  cout<<"input n,r:";
  cin>>n>>r;
  search(1);
  cout<<"number="<<num<<endl;     //输出方案总数
}
```

```
int search(int k)
{
    int i;
    for (i=1;i<=n;i++)
     if  (!b[i])                          //判断i是否可用
      {
         a[k]=i;                          //保存结果
         b[i]=1;
         if (k==r) print();
            else search(k+1);
         b[i]=0;
      }
}
int print()
{
  num++;
  for (int i=1;i<=r;i++)
    cout<<setw(3)<<a[i];
  cout<<endl;
}
```

【例3】任何一个大于1的自然数n，总可以拆分成若干个小于n的自然数之和。

当n=7共14种拆分方法：

7=1+1+1+1+1+1+1

7=1+1+1+1+1+2

7=1+1+1+1+3

7=1+1+1+2+2

7=1+1+1+4

7=1+1+2+3

7=1+1+5

7=1+2+2+2

7=1+2+4

7=1+3+3

7=1+6

7=2+2+3

7=2+5

7=3+4

total=14

【参考程序】

```
#include<cstdio>
#include<iostream>
#include<cstdlib>
using namespace std;
int a[10001]={1},n,total;
int search(int,int);
int print(int);
int main()
{
    cin>>n;
    search(n,1);
        //将要拆分的数n传递给s
    cout<<"total="<<total<<endl;
        //输出拆分的方案数
}
int search(int s,int t)
{
    int i;
    for (i=a[t-1];i<=s;i++)
     if (i<n)
//当前数i要大于等于前1位数，且不过n
     {
      a[t]=i;
//保存当前拆分的数i
      s-=i;
//s减去数i， s的值将继续拆分
      if (s==0) print(t);
//当s=0时，拆分结束输出结果
       else search(s,t+1);
//当s>0时，继续递归
      s+=i;
//回溯：加上拆分的数，以便产分所有可能的拆分
     }
}
int print(int t)
{
    cout<<n<<"=";
    for (int i=1;i<=t-1;i++)
//输出一种拆分方案
      cout<<a[i]<<"+";
    cout<<a[t]<<endl;
    total++;
//方案数累加1
}
```

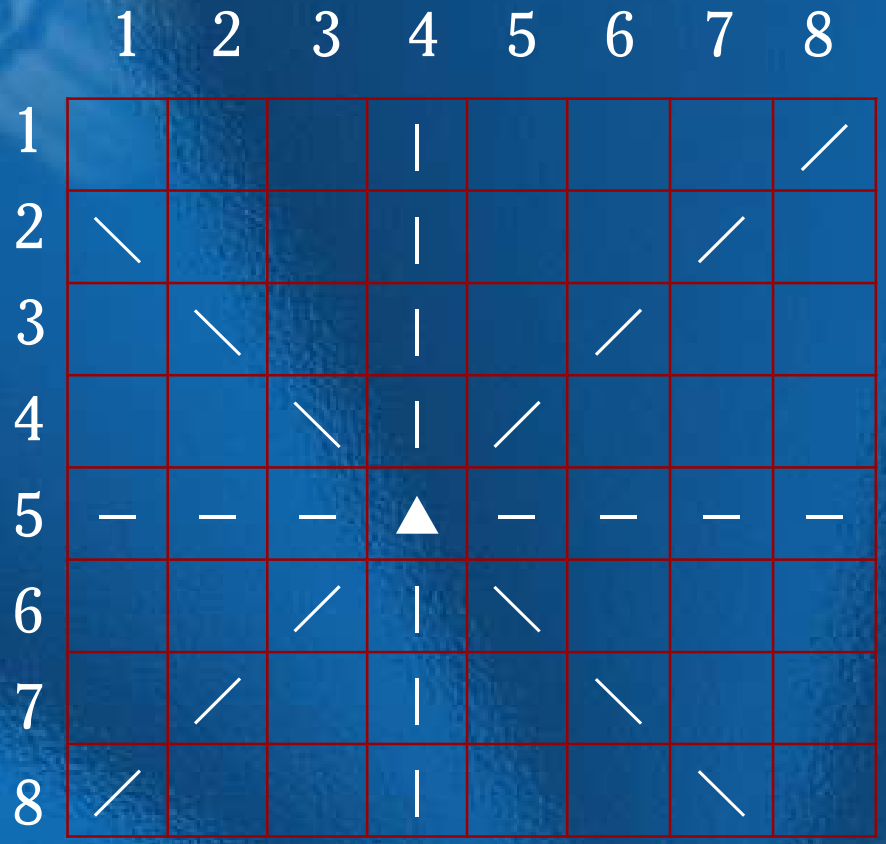
【例4】八皇后问题：要在国际象棋棋盘中放八个皇后，使任意两个皇后都不能互相吃。（提示：皇后能吃同一行、同一列、同一对角线的任意棋子。）

放置第 i 个(行)皇后的算法为：

```
int search(i);
  {
      int j;
       for (第i个皇后的位置j=1;j<=8;j++ )    //在本行的8列中去试
       if (本行本列允许放置皇后)
          {
             放置第i个皇后;
          对放置皇后的位置进行标记;
             if (i==8) 输出                          //已经放完个皇后
                  else search(i+1);                  //放置第i+1个皇后
             对放置皇后的位置释放标记，尝试下一个位置是否可行;
          }
  }
```

【算法分析】

显然问题的关键在于如何判定某个皇后所在的行、列、斜线上是否有别的皇后；可以从矩阵的特点上找到规律，如果在同一行，则行号相同；如果在同一列上，则列号相同；如果同在 / 斜线上的行列值之和相同；如果同在 \ 斜线上的行列值之差相同；从下图可验证：



考虑每行有且仅有一个皇后，设一维数组 A [1..8]表示皇后的放置：第 i 行皇后放在第 j 列，用 A [i]=j来表示，即下标是行数，内容是列数。例如：A[3]=5就表示第3个皇后在第3行第5列上。

判断皇后是否安全，即检查同一列、同一对角线是否已有皇后，建立标志数组 b [1..8]控制同一列只能有一个皇后，若两皇后在同一对角线上，则其行列坐标之和或行列坐标之差相等，故亦可建立标志数组 c [1..16]、 d [-7..7]控制同一对角线上只能有一个皇后。

如果斜线不分方向，则同一斜线上两皇后的行号之差的绝对值与列号之差的绝对值相同。在这种方式下，要表示两个皇后I和J不在同一列或斜线上的条件可以描述为：A[I]<>A[J] AND ABS(I-J)<>ABS(A[I]-A[J]){I和J分别表示两个皇后的行号}

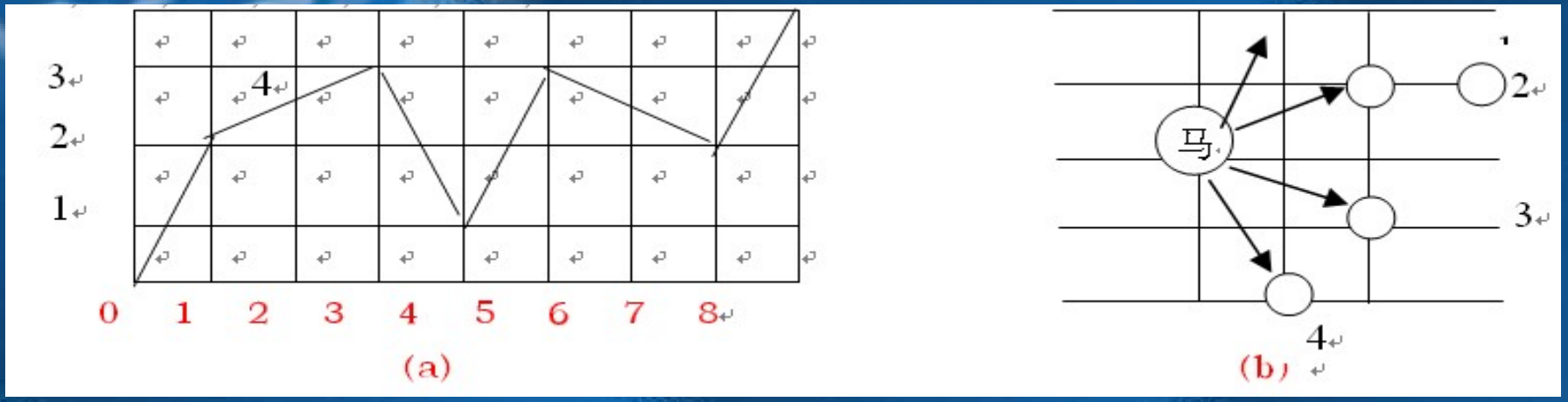
【参考程序】

```
#include<cstdio>
#include<iostream>
#include<cstdlib>
#include<iomanip>
using namespace std;
bool d[100]={0},b[100]={0},c[100]={0};
int sum=0,a[100];
int search(int);
int print();
int main()
{
  search(1);                                    //从第1个皇后开始放置
}
```

```
int search(int i)
{
  int j;
  for (j=1;j<=8;j++)                                  //每个皇后都有8位置(列)可以试放
       if ((!b[j])&&(!c[i+j])&&(!d[i-j+7]))           //寻找放置皇后的位置
      //由于C++不能操作负数组，因此考虑加7
      {                                               //放置皇后,建立相应标志值
      a[i]=j;                                        //摆放皇后
      b[j]=1;                                        //宣布占领第j列
      c[i+j]=1;                                      //占领两个对角线
      d[i-j+7]=1;
      if (i==8) print();                             // 8 个皇后都放置好,输出
        else search(i+1);                            //继续递归放置下一个皇后
      b[j]=0;                                        //递归返回即为回溯一步,当前皇后退出
      c[i+j]=0;
      d[i-j+7]=0;
      }
}
int print()
{
    int i;
    sum++;                                           //方案数累加1
    cout<<"sum="<<sum<<endl;
    for (i=1;i<=8;i++)                               //输出一种方案
      cout<<setw(4)<<a[i];
    cout<<endl;
}
```

【例5】马的遍历

中国象棋半张棋盘如图4（a）所示。马自左下角往右上角跳。今规定只许往右跳，不许往左跳。比如图4（a）中所示为一种跳行路线，并将所经路线打印出来。打印格式为：0,0->2,1->3,3->1,4->3,5->2,7->4,8…



【算法分析】

如图4（b）,马最多有四个方向，若原来的横坐标为j、纵坐标为i,则四个方向的移动可表示为：

1: （i,j）→（i+2,j+1）； (i<3,j<8)
2: （i,j）→（i+1,j+2）； (i<4,j<7)
3: （i,j）→（i-1,j+2）； (i>0,j<7)
4: （i,j）→（i-2,j+1）； (i>1,j<8)

搜索策略：

S1: A[1]:=(0,0);

S2:从A[1]出发，按移动规则依次选定某个方向，如果达到的是（4,8）则转向S3,否则继续搜索下一个到达的顶点；

S3:打印路径。

【参考程序】

```
#include<cstdio>
#include<iostream>
#include<cstdlib>
using namespace std;
int a[100][100],t=0;                    //路径总数和路径
int x[4]={2,1,-1,-2},                   //四种移动规则
    y[4]={1,2,2,1};
 int search(int);                       //搜索
 int print(int);                        //打印
 int main()                             //主程序
 {
   a[1][1]=0;a[1][2]=0;                 //从坐标(0,0)开始往右跳第二步
   search(2);
 };
```

```
int search(int i)
{
   for (int j=0;j<=3;j++)                                  //往4个方向跳
    if (a[i-1][1]+x[j]>=0&&a[i-1][1]+x[j]<=4
      &&a[i-1][2]+y[j]>=0&&a[i-1][2]+y[j]<=8) //判断马不越界
    {
       a[i][1]=a[i-1][1]+x[j];                             //保存当前马的位置
       a[i][2]=a[i-1][2]+y[j];
       if (a[i][1]==4&&a[i][2]==8) print(i);
          else search(i+1);                                //搜索下一步
    }
}
int print(int ii)
{
    {
       t++;
       cout<<t<<":  ";
       for (int i=1;i<=ii-1;i++)
          cout<<a[i][1]<<","<<a[i][2]<<"-->";
       cout<<"4,8"<<endl;
    }
}
```

【例6】设有A，B，C，D，E五人从事J1，J2，J3，J4，J5五项工作，每人只能从事一项，他们的效益如下。

	J1	J2	J3	J4	J5
A	13	11	10	4	7
B	13	10	10	8	5
C	5	9	7	7	4
D	15	12	10	11	5
F	10	11	8	8	4

每人选择五项工作中的一项，在各种选择的组合中，找到效益最高的的一种组合输出。

【算法分析】

1.用数组 f 储存工作选择的方案；数组 g 存放最优的工作选择方案；数组 p 用于表示某项工作有没有被选择了。

2.(1)选择 p (i)=0的第i项工作；

(2)判断效益是否高于max已记录的效益，若高于则更新 g 数组及max的值。

3.搜索策略: 回溯法（深度优先搜索dfs）。

【参考程序】

```
#include<cstdio>
#include<iostream>
#include<cstdlib>
#include<iomanip>
using namespace std;
int
data[6][6]={{0,0,0,0,0,0},{0,13,11,10,4,7},{0,13,10,10,8,5},{0,5,9,7,7,4},{0,15,12,10,11,5},{0,10,11,8,8,4}};
int max1=0,g[10],f[10];
bool p[6]={0};
int go(int step,int t)                    // step是第几个人，t是之前已得的效益
{
  for (int i=1;i<=5;i++)
   if (!p[i])                          //判断第i项工作没人选择
    {
      f[step]=i;                         //第step个人，就选第i项工作
      p[i]=1;                           //标记第i项工作被人安排了
      t+=data[step][i];                      //计算效益值
      if (step<5) go(step+1,t);
         else if (t>max1)                    //保存最佳效益值
          {
            max1=t;
            for (int j=1;j<=5;j++)
              g[j]=f[j];                 //保存最优效益下的工作选择方案
          }
      t-=data[step][i];                      //回溯
      p[i]=0;
    }
}
```

```
int main()
{
  go(1,0);                                  //从第1个人，总效益为0开始
  for (int i=1;i<=5;i++)
    cout<<char(64+i)<<":J"<<g[i]<<setw(3);    //输出各项工作安排情况
  cout<<endl;
  cout<<"supply:"<<max1<<endl;              //输出最佳效益值
}
```

【例7】选书

学校放寒假时，信息学竞赛辅导老师有A，B，C，D，E五本书，要分给参加培训的张、王、刘、孙、李五位同学，每人只能选一本书。老师事先让每个人将自己喜欢的书填写在如下的表格中。然后根据他们填写的表来分配书本，希望设计一个程序帮助老师求出所有可能的分配方案，使每个学生都满意。

学生	A	B	C	D	E
张同学			Y	Y	
王同学	Y	Y			Y
刘同学		Y	Y		
孙同学				Y	
李同学		Y			Y

【算法分析】

可用穷举法，先不考虑“每人都满意”这一条件，这样只剩“每人选一本且只能选一本”这一条件。在这个条件下，可行解就是五本书的所有全排列，一共有5!=120种。然后在120种可行解中一一删去不符合“每人都满意”的解，留下的就是本题的解答。

为了编程方便，设1，2，3，4，5分别表示这五本书。这五个数的一种全排列就是五本书的一种分发。例如54321就表示第5本书（即E）分给张，第4本书（即D）分给王，……，第1本书（即A）分给李。“喜爱书表”可以用二维数组来表示，1表示喜爱，0表示不喜爱。

算法设计：S1：产生5个数字的一个全排列；
S2：检查是否符合“喜爱书表”的条件，如果符合就打印出来；
S3：检查是否所有的排列都产生了，如果没有产生完，则返回S1；
S4：结束。

上述算法有可以改进的地方。比如产生了一个全排列12345，从表中可以看出，选第一本书即给张同学的书，1是不可能的，因为张只喜欢第3、4本书。这就是说，1×××××一类的分法都不符合条件。由此想到，如果选定第一本书后，就立即检查一下是否符合条件，发现1是不符合的，后面的四个数字就不必选了，这样就减少了运算量。换句话说，第一个数字只在3、4中选择，这样就可以减少3/5的运算量。同理，选定了第一个数字后，也不应该把其他4个数字一次选定，而是选择了第二个数字后，就立即检查是否符合条件。例如，第一个数选3，第二个数选4后，立即检查，发现不符合条件，就应另选第二个数。这样就把34×××一类的分法在产生前就删去了。又减少了一部分运算量。

综上所述，改进后的算法应该是：在产生排列时，每增加一个数，就检查该数是否符合条件，不符合，就立刻换一个，符合条件后，再产生下一个数。因为从第I本书到第I+1本书的寻找过程是相同的，所以可以用回溯算法。算法设计如下：

```
int Search(i)
{
  for (j=1;j<=5;j++)
   {
     if (第i个同学分给第j本书符合条件)
      {
        记录第i个数
        if (i==5) 打印一个解;
              else Search(i+1);
        删去第i 个数
      }
   }
}
```

【参考程序】

```
#include<cstdio>
#include<iostream>
#include<cstdlib>
using  namespace std;
int  book[6],c;
bool flag[6],like[6][6]={{0,0,0,0,0,0},{0,0,0,1,1,0},{0,1,1,0,0,1},
                         {0,0,1,1,0,0},{0,0,0,0,1,0},{0,0,1,0,0,1}};;
int  search(int);
int  print();
int  main()
{
  for (int i=1;i<=5;i++) flag[i]=1;
  search(1);                          //从第1个开始选书，递归。
}
int search(int i)                     //递归函数
{
  for (int j=1;j<=5; j++)             //每个人都有5本书可选
  if (flag[j]&&like[i][j])            //满足分书的条件
  {
    flag[j]=0;                        //把被选中的书放入集合flag中，避免重复被选
    book[i]=j;                        //保存第i个人选中的第j本书
    if (i==5) print();                //i=5时，所有的人都分到书，输出结果
       else search(i+1);              //i<5时，继续递归分书
    flag[j]=1;                        //回溯：把选中的书放回，产生其他分书的方案
    book[i]=0;
  }
}
```

```
int print()
{
  c++;                                                    //方案数累加1
  cout <<"answer " <<c <<":\n";
  for (int i=1;i<=5;i++)
    cout <<i <<": " <<char(64+book[i]) <<endl;  //输出分书的方案
}
```

输出结果：

```
zhang: C
wang: A
liu: B
sun: D
li: E
```

【例8】跳马问题。在5*5格的棋盘上，有一只中国象棋的马，从（1,1）点出发，按日字跳马，它可以朝8个方向跳，但不允许出界或跳到已跳过的格子上，要求在跳遍整个棋盘。

输出前5个方案及总方案数。

输出格式示例：

```
1    16   21   10   25
20   11   24   15    22
17   2     19   6     9
12   7     4     23   14
3    18    13   8     5
```

```
#include<cstdio>
#include<iostream>
#include<cstdlib>
#include<iomanip>
using namespace std;
int u[8]={1,2,2,1,-1,-2,-2,-1},          //8个方向上的x,y增量
    v[8]={-2,-1,1,2,2,1,-1,-2};
int a[100][100]={0},num=0;               //记每一步走在棋盘的哪一格和棋盘的每一格有
                                         //没有被走过
bool b[100][100]={0};
int search(int,int,int);                 //以每一格为阶段，在每一阶段中试遍8个方向
int print();                             //打印方案
```

```
int main()
{
   a[1][1]=1;b[1][1]=1;                          //从(1,1)第一步开始走
   search(1,1,2);                                //从(1,1)开始搜第2步该怎样走
   cout<<num<<endl;                              //输出总方案(304)
}
int search(int i,int j,int n)
{
  int k,x,y;                                     //这三个变量一定要定义局部变量
  if (n>25) {print();return 0;}                  //达到最大规模打印、统计方案
  for (k=0;k<=7;k++)                             //试遍8个方向
   {
     x=i+u[k];y=j+v[k];                          //走此方向，得到的新坐标
     if (x<=5&&x>=1&&y<=5&&y>=1&&(!b[x][y]))
      {                                          //如果新坐标在棋盘上，并且这一格可以走
         b[x][y]=1;
         a[x][y]=n;
         search(x,y,n+1);                        //从(x,y)去搜下一步该如何走
         b[x][y]=0;
         a[x][y]=0;
      }
   }
}
```

```
int print()
{
   num++;                                  //统计总方案
   if (num<=5)                             //打印出前5种方案
    {
       for (int k=1;k<=5;k++)              //打印本次方案
        {
          for (int kk=1;kk<=5;kk++)
            cout<<setw(5)<<a[k][kk];
          cout<<endl;
        }
    }
}
```

【例9】数的划分(NOIP2001)

【问题描述】

将整数n分成k份，且每份不能为空，任意两种分法不能相同(不考虑顺序)。例如:n=7,k=3,下面三种分法被认为是相同的。

- 1，1，5； 1，5，1； 5，1，1；
- 问有多少种不同的分法。
- 【输入格式】
- n,k (6<n≤200，2≤k≤6)
- 【输出格式】
- 一个整数，即不同的分法。
- 【输入样例】
- 7 3
- 【输出样例】
- 4

{4种分法为：1,1,5；1,2,4；1,3,3；2,2,3 说明部分不必输出 }

【算法分析】

- 方法1、回溯法，超时，参考程序如下。

```
#include<cstdio>
#include<iostream>
#include<cstdlib>
using namespace std;
int n,i,j,k,rest,sum,total;
int s[7];
int main()
{
    cout << "Input n k";
    cin >> n >> k;
    total = 0; s[1] = 0;
    i = 1;
    while (i)
    {
        s[i]++;
        if (s[i] > n) i--;
          else if (i == k)
              {
                  sum = 0;
                  for (j = 1; j <= k; j++) sum += s[j];
                  if (n == sum) total++;
              }
              else {
                      rest -= s[i];
                      i++;
                      s[i] = s[i-1] - 1;
                   }
    }
    cout << total;
   return 0;
}
```

- 方法2、递归，参考程序如下。

```
#include<cstdio>
#include<iostream>
#include<cstdlib>
using namespace std;
int n,k;
int f(int a,int b,int c)
{
    int g = 0,i;
    if (b == 1) g = 1;
      else for (i = c; i <= a/b; i++)
            g += f(a-i,b-1,i);
    return g;
}
int main()
{
    cout << "Input n,k:";
    cin >> n >> k;
    cout << f(n,k,1);
    return 0;
}
```

方法3、用动态循环穷举所有不同的分解，要注意剪枝，参考程序如下。

```
#include<cstdio>
#include<iostream>
#include<cstdlib>
using namespace std;
int n,k,total;
int min(int x,int y)
{
    if (x < y) return x;
        else return y;
}
void select(int dep,int rest,int last)
{
     int i;
     if (dep == 0) total++;
         else for (i = min(rest-dep+1,last); i >= rest/dep; i--)
             select(dep-1,rest-i,i);
}
int main()
{
    cout << "Input n,k:";
    cin >> n >> k;
    total = 0;
    select(k,n,n);
    cout << total;
    return 0;
}
```

- 方法4、递推法
- 首先将正整数n分解成k个正整数之和的不同分解方案总数等于将正整数n-k分解成任意个不大于k的正整数之和的不同分解方案总数(可用ferror图证明之),后者的递推公式不难得到,参考程序如下。

```
#include<cstdio>
#include<iostream>
#include<cstdlib>
#include<cstring>
using namespace std;
int i,j,k,n,x;
int p[201][7];
int main()
{
    cin >> n >> k;
    memset(p,0,sizeof(p));
    p[0][0] = 1;
    for (i = 1; i <= n; i++) p[i][1] = 1;
    for (i = 1; i <= n - k; i++)
        for (j = 2; j <= min(i,k); j++)
            for (x = 1; x <= min(i,j); x++)
                p[i][j] += p[i-x][min(i-x,x)];
    cout << p[n-k][k];
   return 0;
}
```

【课堂练习】

1、输出自然数1到n所有不重复的排列，即n的全排列。

【参考过程】

```
int Search(int i)
 {
   Int j;
   for (j=1;j<=n;j++)
     if (b[j])
      {
        a[i]=j;  b[j]=false;
        if (I<n) Search(i+1);
              else print();
        b[j]=true;
      }
 }
```

2、找出n个自然数(1,2,3,…,n)中r个数的组合。例如，当n= 5 ,r=3时，所有组合为：

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

total=10　　//组合的总数

【分析】：设在b[1],b[2],…,b[i-1]中已固定地取了某一组值且b[i-1]=k的前提下，过程Search(i,k)能够列出所有可能的组合。由于此时b[i]只能取k+1至n-r+i，对j=k+1,k+2,…,n-r+i，使b[i]:=j，再调用过程Search(i+1,j)，形成递归调用。直至i的值大于r时，就可以在b中构成一种组合并输出。

3、输出字母a、b、c、d，4个元素全排列的每一种排列。

4、显示从前m个大写英文字母中取n个不同字母的所有种排列。

5、有A、B、C、D、E五本书，要分给张、王、刘、赵、钱五位同学，每人只能选一本，事先让每人把自己喜爱的书填于下表，编程找出让每人都满意的所有方案。

	A	B	C	D	E
张			Y	Y	
王	Y	Y			Y
刘		Y	Y		
赵	Y	Y		Y	
钱		Y			Y

【答案】四种方案

张　王　刘　赵　钱

①C　A　B　D　E　　②D　A　C　B　E

③D　B　C　A　E　　④D　E　C　A　B

6、有红球4个，白球3个，黄球3个，将它们排成一排共有多少种排法？

【分析】：可以用回溯法来生成所有的排法。用数组b[1..3]表示尚未排列的这3种颜色球的个数。设共有I-1个球已参加排列，用子程序Search(i)生成由第I个位置开始的以后n-I+1位置上的各种排列。对于第I个位置，我们对3种颜色的球逐一试探，看每种颜色是否还有未加入排序的球。若有，则选取一个放在第I个位置上，且将这种球所剩的个数减1，然后调用Search(I+1),直至形成一种排列后出。对第I个位置上的所有颜色全部试探完后，则回溯至前一位置。

【上机练习】

1、全排列问题(Form.cpp)

【问题描述】

输出自然数1到n所有不重复的排列，即n的全排列，要求所产生的任一数字序列中不允许出现重复的数字。

【输入格式】

n(1≤n≤9)

【输出格式】

由1～n组成的所有不重复的数字序列，每行一个序列。

【输入样例】Form.in

3

【输出样例】Form.out

1 2 3
1 3 2
2 1 3
2 3 1
3 1 2
3 2 1

2、组合的输出(Compages.cpp)

【问题描述】

排列与组合是常用的数学方法，其中组合就是从n个元素中抽出r个元素(不分顺序且r<=n)，我们可以简单地将n个元素理解为自然数1，2，…，n，从中任取r个数。

现要求你用递归的方法输出所有组合。

例如n=5，r=3，所有组合为：

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

【输入】

一行两个自然数n、r(1<n<21，1<=r<=n)。

【输出】

所有的组合，每一个组合占一行且其中的元素按由小到大的顺序排列，每个元素占三个字符的位置，所有的组合也按字典顺序。

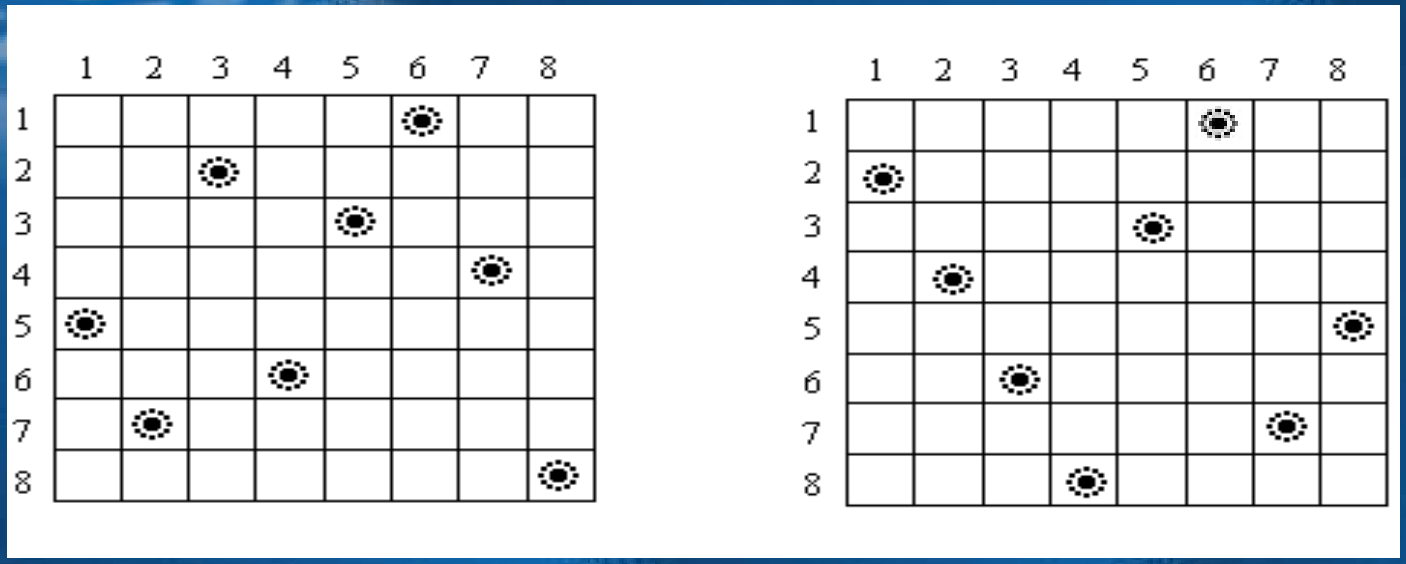
【样例】

compages.in	compages.out
5　3	1 2 3 1 2 4 1 2 5 1 3 4 1 3 5 1 4 5 2 3 4 2 3 5 2 4 5 3 4 5

3、N皇后问题(Queen.cpp)

【问题描述】

在N*N的棋盘上放置N个皇后（n<=10）而彼此不受攻击（即在棋盘的任一行，任一列和任一对角线上不能放置2个皇后），编程求解所有的摆放方法。



八皇后的两组解

【输入格式】

输入：n

【输出格式】

每行输出一种方案，每种方案顺序输出皇后所在的列号，各个数之间有空格隔开。若无方案，则输出no solute!

【输入样例】Queen.in

4

【输出样例】Queen.out

2 4 1 3

3 1 4 2

4、有重复元素的排列问题

【问题描述】

设R={ r1, r2 , …, rn}是要进行排列的n个元素。其中元素r1, r2 , …, rn可能相同。试设计一个算法，列出R的所有不同排列。

【编程任务】

给定n 以及待排列的n 个元素。计算出这n 个元素的所有不同排列。

【输入格式】

由perm.in输入数据。文件的第1 行是元素个数n，1≤n≤500。接下来的1 行是待排列的n个元素。

【输出格式】

计算出的n个元素的所有不同排列输出到文件perm.out中。文件最后1行中的数是排列总数。

【输入样例】

4
aacc

【输出样例】多解

aacc
acac
acca
caac
caca
ccaa
6

5、子集和问题

【问题描述】

子集和问题的一个实例为〈S,t〉。其中，S={ x1， x2，…， xn}是一个正整数的集合，c是一个正整数。子集和问题判定是否存在S的一个子集S1，使得子集S1和等于c。

【编程任务】

对于给定的正整数的集合S={ x1， x2，…， xn}和正整数c，编程计算S 的一个子集S1，使得子集S1和等于c。

【输入格式】

由文件subsum.in提供输入数据。文件第1行有2个正整数n和c，n表示S的个数，c是子集和的目标值。接下来的1 行中，有n个正整数，表示集合S中的元素。

【输出格式】

程序运行结束时，将子集和问题的解输出到文件subsum.out中。当问题无解时，输出“No solution!”。

【输入样例】

5 10
2 2 6 5 4

【输出样例】

2 2 6

6、工作分配问题

【问题描述】

设有n件工作分配给n个人。将工作i分配给第j个人所需的费用为cij。试设计一个算法，为每一个人都分配一件不同的工作，并使总费用达到最小。

【编程任务】

设计一个算法，对于给定的工作费用，计算最佳工作分配方案，使总费用达到最小。

【输入格式】

由文件job.in给出输入数据。第一行有1个正整数n (1≤n≤20)。接下来的n行，每行n个数，第i行表示第i个人各项工作费用。

【输出格式】

将计算出的最小总费用输出到文件job.out。

【输入样例】

```
3
4  2  5
2  3  6
3  4  5
```

【输出样例】

```
9
```

7、装载问题

【问题描述】

有一批共n个集装箱要装上艘载重量为c的轮船，其中集装箱i的重量为wi。找出一种最优装载方案，将轮船尽可能装满，即在装载体积不受限制的情况下，将尽可能重的集装箱装上轮船。

【输入格式】

由文件load.in给出输入数据。第一行有2个正整数n和c。n是集装箱数，c是轮船的载重量。接下来的1行中有n个正整数，表示集装箱的重量。

【输出格式】

将计算出的最大装载重量输出到文件load.out。

【输入样例】

5 10

7 2 6 5 4

【输出样例】

10

8、字符序列(characts)

【问题描述】

从三个元素的集合[A，B，C]中选取元素生成一个N个字符组成的序列，使得没有两个相邻字的子序列（子序列长度=2）相同。例：N = 5时ABCBA是合格的，而序列ABCBC与ABABC是不合格的，因为其中子序列BC，AB是相同的。

对于由键盘输入的N(1<=N<=12)，求出满足条件的N个字符的所有序列和其总数。

【输入样例】

4

【输出样例】

72

9、试卷批分(grade)

【问题描述】

某学校进行了一次英语考试，共有10道是非题，每题为10分，解答用1表示“是”，用0表示“非”的方式。但老师批完卷后，发现漏批了一张试卷，而且标准答案也丢失了，手头只剩下了3张标有分数的试卷。

试卷一：① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩

0 0 1 0 1 0 0 1 0 0 得分：70

试卷二：① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩

0 1 1 1 0 1 0 1 1 1 得分：50

试卷三：① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩

0 1 1 1 0 0 0 1 0 1 得分：30

待批试卷：① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩

0 0 1 1 1 0 0 1 1 1 得分：？

【问题求解】：

请编一程序依据这三张试卷，算出漏批的那张试卷的分数。

- **10、迷宫问题(migong)**
- 【问题描述】

- 设有一个N*N(2<=N<10)方格的迷宫，入口和出口分别在左上角和右上角。迷宫格子中分别放0和1，0表示可通，1表示不能，入口和出口处肯定是0。迷宫走的规则如下所示：即从某点开始，有八个方向可走，前进方格中数字为0时表示可通过，为1时表示不可通过，要另找路径。找出所有从入口（左上角）到出口（右上角）的路径(不能重复)，输出路径总数，如果无法到达，则输出0。

- 【输入样例】

- 3
- 0 0 0
- 0 1 1
- 1 0 0

- 【输出样例】

- 2　　//路径总数

11、部落卫队

【问题描述】

原始部落byteland中的居民们为了争夺有限的资源，经常发生冲突。几乎每个居民都有他的仇敌。部落酋长为了组织一支保卫部落的队伍，希望从部落的居民中选出最多的居民入伍，并保证队伍中任何2 个人都不是仇敌。

【编程任务】

给定byteland部落中居民间的仇敌关系，编程计算组成部落卫队的最佳方案。

【输入格式】

第1行有2个正整数n和m，表示byteland部落中有n个居民，居民间有m个仇敌关系。居民编号为1，2，…，n。接下来的m行中，每行有2个正整数u和v，表示居民u与居民v是仇敌。

【输出格式】

第1行是部落卫队的顶人数；文件的第2行是卫队组成x i，1≤i≤n，xi =0 表示居民i不在卫队中，xi=1表示居民i在卫队中。

【输入样例】

```
7  10
1  2
1  4
2  4
2  3
2  5
2  6
3  5
3  6
4  5
5  6
```

【输出样例】

```
3
1 0 1 0 0 0 1
```

12、最佳调度问题

【问题描述】

假设有n个任务由k个可并行工作的机器完成。完成任务i需要的时间为ti。试设计一个算法找出完成这n个任务的最佳调度，使得完成全部任务的时间最早。

【编程任务】

对任意给定的整数n和k，以及完成任务i需要的时间为ti，i=1~n。编程计算完成这n个任务的最佳调度。

【输入格式】

由文件machine.in给出输入数据。第一行有2 个正整数n和k。第2 行的n个正整数是完成n个任务需要的时间。

【输出格式】

将计算出的完成全部任务的最早时间输出到文件machine.out。

【输入样例】

```
7  3
2  14  4  16  6  5  3
```

【输出样例】

```
 17
```

13、图的m着色问题

【问题描述】

给定无向连通图G和m种不同的颜色。用这些颜色为图G的各顶点着色，每个顶点着一种颜色。如果有一种着色法使G中每条边的2个顶点着不同颜色，则称这个图是m可着色的。图的m着色问题是对于给定图G和m种颜色，找出所有不同的着色法。

【编程任务】

对于给定的无向连通图G和m种不同的颜色，编程计算图的所有不同的着色法。

【输入格式】

文件color.in输入数据。第1行有3个正整数n，k 和m，表示给定的图G有n个顶点和k条边，m种颜色。顶点编号为1，2，…，n。接下来的k行中，每行有2个正整数u,v，表示图G 的一条边(u,v)。

【输出格式】

程序运行结束时，将计算出的不同的着色方案数输出到文件color.out中。

【输入样例】

```
5 8 4
1 2
1 3
1 4
2 3
2 4
2 5
3 4
4 5
```

【输出样例】

```
48
```