

# 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

## 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 isprime(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 (isprime(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 (isprime(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 isprime(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 $\{1, 2, \dots, n\}$ , 从中任意取出 $r$ 个数进行排列( $r < n$ ), 试着列出所有的排列

**【解题思路】**题目含义可以简化从 $n$ 个整数中取出 $r$ 个数的全排列。通过循环递归调用来试探可能解, 使用回溯来剔除已经找到并输出的解。使用数组 $a$ 存储待排列的数, 通过布尔数组 $b$ 来实现选用标记和回溯操作。选用是某个整数作为备用解的意思是, 将该整数对应的布尔数组单元的值置为1, 表示选用, 置为0表示没有被选用, 回溯是通过 $b[i]=0$  (置0) 来实现的, 置0的目的是供下一轮递归调用求解选用。每次执行完回溯操作 $b[i]=0$ 后, 该次递归调用 $i$ 的值++, 确保下次试探新的排列解的时候, 不会选用刚通过回溯操作释放出来的整数 (确保在本轮求解中不会在该位置再次选用该数作为解)。

每一层的递归调用 (第 $k$ 层调用) 都使用循环来遍历本层可以选用的整数 (尚未被标注选用的, 对应标识单元置为0)。  $k=1$ , 搜索排列中的第一个数;  $k=2$ 的时候, 搜索排列中的第2个数, 发生第一次递归调用 (search第2次被调用,  $k=1$ 以及发生search( $k+1$ )调用前的 $i$ 值被压栈保存, 该 $i$ 值表示整数 $i$ 已经被选用。下层 ( $k+1$ ) 递归调用返回时, 回溯该值 $b[i]=0$ , 表示下轮求解中该数又可以被选用);  $k=3$ 的时候 (search第3次被调用), 搜索排列中的第3个数, ..., 以此类推。

$k=r$ 时 (search被第 $r$ 次调用, 实际上是第 $r-1$ 次递归调用), 由于满足 $k=r$ , 不会再发生递归调用, 执行完 $b[i]=0$ , 将第 $r$ 个位置选用的数被释放后, 该回溯语句执行后直接返回到上层调用 $k=r-1$ 压栈前的状态 (断点), 返回到 $k=r-1$ 层的断点后, 再次执行 $b[i]=0$ 的回溯语句, 将该层 (第 $r-1$ 个位置) 之前选用的数释放, 然后执行该层的 $i++$ 语句, 选择下个可选的数, 如果 $i > r$ , 表示没有可选, 退出第 $k=r-1$ 层的递归调用, 返回 $k=r-2$ 层, 继续回溯和释放。

# 参考程序

```
#include<cstdio> //兼容C标准输入输出C++头文件
#include<iostream> //标准输入输出
#include<iomanip> //io代表输入输出, manip是manipulator (操纵器) 的缩写, 使用
//ssetfill, setw, setbase, setprecision等添加
using namespace std;
int num = 0, a[10001] = { 0 }, n, r; //n为整数集合, r为取出待全排列整数数目 (r<n),
//num用于存储方案数, 数组a存储满足条件的r个数排列
bool b[10001] = { 0 }; //数组b
int search(int); //递归回溯过程函数
int print(); //输出可行方案函数
int main()
{
    cout << "input n,r:"; //输入n,r的提示
    cin >> n >> r; //输入n,r
    search(1);
    cout << "number=" << num << endl; //输出方案总数
}
int print()
{
    num++; //累加符合条件排列的个数
    for (int i = 1; i <= r; i++)
        cout << setw(3) << a[i]; //setw(int n)用来控制输出间隔
    cout << endl;
    return 0;
}
```



```

int search(int k)
{
    int i;
    for (i = 1; i <= n; i++)          //通过i++实现往前走（选下个没有被选的数），
                                      //通过b[i]=0，实现回溯。
    {
        if (!b[i])                   //判断第i个数是否被选用
        {
            a[k] = i;                //第k个位置选用数i,前提是没有被选用
            b[i] = 1;                 //第k个位置选用数i,后置0，表示被占用
            if (k == r) print();      //k=r，表示已经选到满足条件的r个数，输出并且通过
                                      //b[i]回溯，不再递归调用，返回r-1层调用。
            else search(k + 1);       //递归调用，选取第k+1个位置的数
            b[i] = 0;
            /*从k+1层递归返回时，b[i]中i就为第k层函数search(k)执行时调用第k+1层递归函数前的i
            值，b[i]=0，回溯，表示可以再次选用，但是随后执行的i++，保证整数i不会在该层再次被
            选用，完成往前走的搜索。最后一层调用，也就是k==r时，b[i]就是第r层调用时满足
            b[i]!=0,也就是还有没有选的那个数的值，选用该值后，找到满足条件的一组解后输出，输
            出收这个数被释放，然后执行i++，i>n时退出k==r层的调用，返回r-1层，再次执行
            b[i]=0,释放之前r-1层选用的数。*/
        }
    }
    return 0;
}

```



**【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：

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

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

$$7=1+1+1+1+3$$

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

$$7=1+1+1+4$$

$$7=1+1+2+3$$

$$7=1+1+5$$

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

$$7=1+2+4$$

$$7=1+3+3$$

$$7=1+6$$

$$7=2+2+3$$

$$7=2+5$$

$$7=3+4$$

$$\text{total}=14$$

## 【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, 以便产生所有可能
        //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：

	1	2	3	4	5	6	7	8
1								/
2	\						/	
3		\				/		
4			\		/			
5	—	—	—	▲	—	—	—	—
6			/		\			
7		/				\		
8	/						\	

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： $A[I] \neq A[J] \text{ AND } \text{ABS}(I-J) \neq \text{ABS}(A[I]-A[J])$  ( $I$  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

```

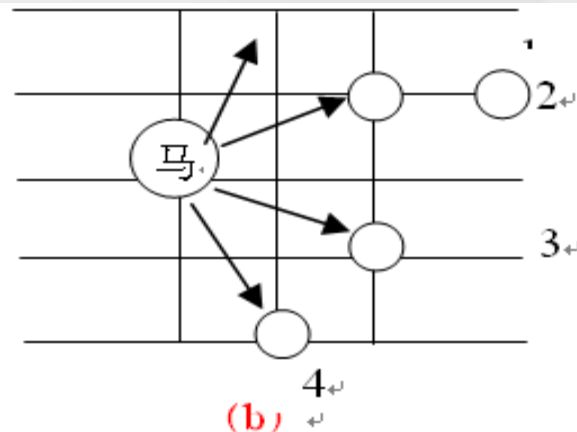
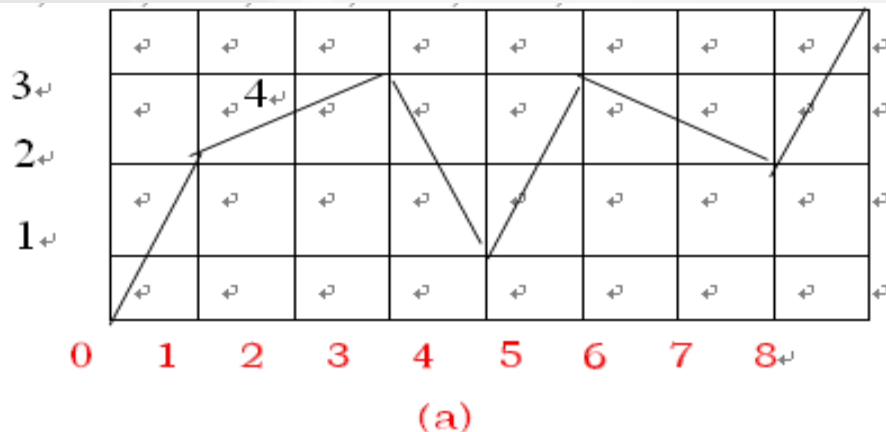
int search(int i)
{
    int j;
    for (j=1;j<=8;j++)
        if ((!b[j])&&(!c[i+j])&&(!d[i-j+7]))
            //每个皇后都有8位置(列)可以试放
            //寻找放置皇后的位置
            //由于C++不能操作负数组，因此考虑加7
            {
                //放置皇后,建立相应标志值
                //摆放皇后
                //宣布占领第j列
                //占领两个对角线
                a[i]=j;
                b[j]=1;
                c[i+j]=1;
                d[i-j+7]=1;
                if (i==8) print();
                // 8 个皇后都放置好,输出
                //继续递归放置下一个皇后
                //递归返回即为回溯一步,当前皇后退出现
                else search(i+1);
                b[j]=0;
                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:  $(i,j) \rightarrow (i+2,j+1)$  ;  $(i<3,j<8)$
- 2:  $(i,j) \rightarrow (i+1,j+2)$  ;  $(i<4,j<7)$
- 3:  $(i,j) \rightarrow (i-1,j+2)$  ;  $(i>0,j<7)$
- 4:  $(i,j) \rightarrow (i-2,j+1)$  ;  $(i>1,j<8)$

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\书	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<stdio>
#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},
```

```
    v[8]={-2,-1,1,2,2,1,-1,-2};
```

```
int a[100][100]={0},num=0;
```

```
bool b[100][100]={0};
```

```
int search(int,int,int);
```

```
int print();
```

//8个方向上的x,y增量

//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

//以每一格为阶段，在每一阶段中试遍8个方向  
//打印方案

```
int main()
```

```
{
```

```
    a[1][1]=1;b[1][1]=1;
```

```
    search(1,1,2);
```

```
    cout<<num<<endl;
```

```
}
```

```
int search(int i,int j,int n)
```

```
{
```

```
    int k,x,y;
```

```
    if (n>25) {print();return 0;}
```

```
    for (k=0;k<=7;k++)
```

```
    {
```

```
        x=i+u[k];y=j+v[k];
```

```
        if (x<=5&&x>=1&&y<=5&&y>=1&&(!b[x][y]))
```

```
        {
```

```
            b[x][y]=1;
```

```
            a[x][y]=n;
```

```
            search(x,y,n+1);
```

```
            b[x][y]=0;
```

```
            a[x][y]=0;
```

```
        }
```

```
    }
```

```
}
```

//从(1,1)第一步开始走

//从(1,1)开始搜第2步该怎样走

//输出总方案(304)

//这三个变量一定要定义局部变量

//达到最大规模打印、统计方案

//试遍8个方向

//走此方向，得到的新坐标

//如果新坐标在棋盘上，并且这一格可以走

//从(x,y)去搜下一步该如何走

```
int print()
{
    num++;
    if (num<=5)
    {
        for (int k=1;k<=5;k++)
        {
            for (int kk=1;kk<=5;kk++)
                cout<<setw(5)<<a[k][kk];
            cout<<endl;
        }
    }
}
```

//统计总方案  
//打印出前**5**种方案

//打印本次方案

## 【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】

$n, k$     ( $6 < n \leq 200, 2 \leq k \leq 6$ )

### 【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Ferrers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 $(1,2,3,\dots,n)$ 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 $b[1], b[2], \dots, b[i-1]$ 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 $j=k+1, k+2, \dots, n-r+i$ ，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

① C A B D E

② D A C B E

③ D B C A E

④ D 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 **$l-1$** 个球已参加排列，用子程序**Search(i)**生成由第 **$l$** 个位置开始的以后 **$n-l+1$** 位置上的各种排列。对于第 **$l$** 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 **$l$** 个位置上，且将这种球所剩的个数减1，然后调用**Search(l+1)**，直至形成一种排列后出。对第 **$l$** 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( $1 \leq n \leq 9$ )

### 【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 $r \leq n$ )，我们可以简单地将 $n$ 个元素理解为自然数 $1, 2, \dots, n$ ，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 $n$ 、 $r$ ( $1 < n < 21$ ， $1 \leq r \leq n$ )。

### 【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

5 3

compages.out

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 $N \times N$ 的棋盘上放置 $N$ 个皇后（ $n \leq 10$ ）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。

	1	2	3	4	5	6	7	8
1						♚		
2			♚					
3					♚			
4							♚	
5	♚							
6				♚				
7		♚						
8								♚

	1	2	3	4	5	6	7	8
1						♚		
2	♚							
3					♚			
4		♚						
5								♚
6			♚					
7							♚	
8				♚				

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 $R=\{r_1, r_2, \dots, r_n\}$ 是要进行排列的 $n$ 个元素。其中元素 $r_1, r_2, \dots, r_n$ 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 $n$ ， $1 \leq n \leq 500$ 。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  $\langle S, t \rangle$ 。其中， $S = \{x_1, x_2, \dots, x_n\}$  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  $S_1$ ，使得子集  $S_1$  和等于  $c$ 。

### 【编程任务】

对于给定的正整数的集合  $S = \{x_1, x_2, \dots, x_n\}$  和正整数  $c$ ，编程计算  $S$  的一个子集  $S_1$ ，使得子集  $S_1$  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 $c_{ij}$ 。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 $n$  ( $1 \leq n \leq 20$ )。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 $w_i$ 。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( $1 \leq N \leq 12$ )，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 $N \times N$  ( $2 \leq N < 10$ ) 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行是部落卫队的最多人数；文件的第2行是卫队组成 $x_i, 1 \leq i \leq n$ ,  $x_i = 0$  表示居民i不在卫队中， $x_i = 1$ 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 $t_i$ 。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 $t_i$ ， $i=1\sim n$ 。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】

#### 【输出样例】

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

```
48
```