《计算概论A》课程 程序设计部分

函数的递归调用 (3)

李　戈

北京大学 信息科学技术学院 软件研究所

lige@sei.pku.edu.cn

北京大学

递归问题解法小结

- 面对一个问题时：
  - ① 从简单情况开始分析问题，找出解决问题的规律；
  - ② 总结并抽取出解决方案中**反复做的事情**；
  - ③ 用一个**函数（原型）**来描述这件**反复做的事情**；
  - ④ 假设反复做的事情已经由上述函数实现，写出如何利用上述函数解决整体问题的步骤；
  - ⑤ 分析并写出边界条件；

典型问题分析

- 问题
  - 从键盘读入一个英文单词（全部字母小写，且该单词中各个字母均不相同），输出该单词英文字母的所有全排列；
  - 例如，输入**abc**，则打印出：

abc
acb
bac
bca
cab
cba

史上最长的英文单词

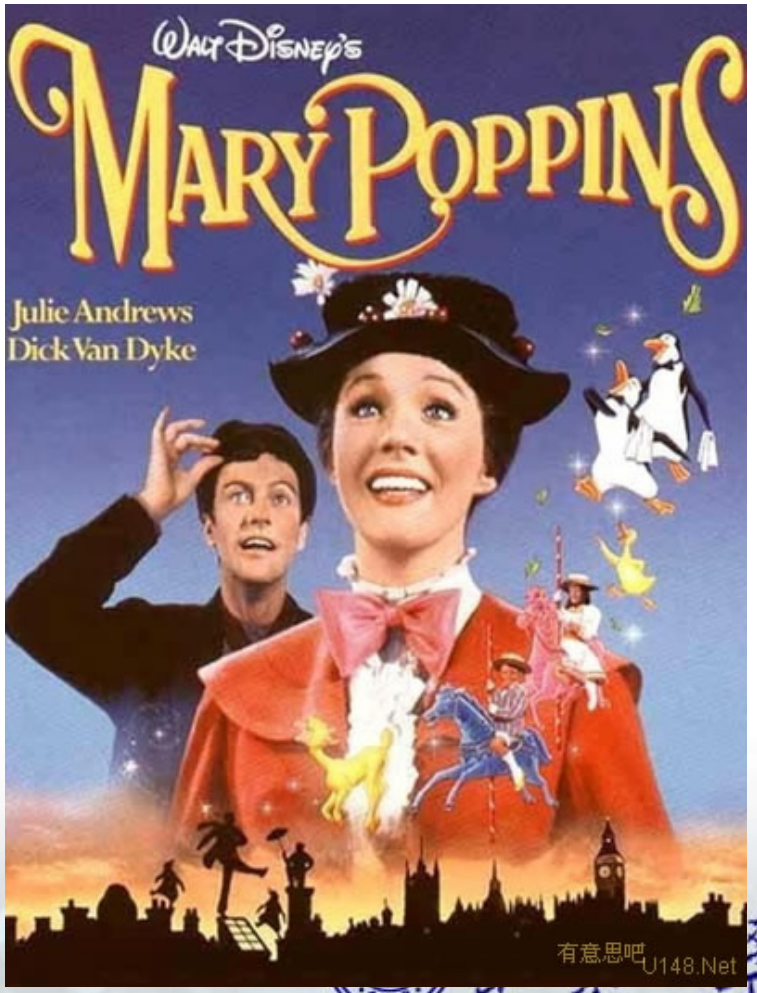
Honorificabilitudinitatibus

- 由27个字母组成
- 出现在莎士比亚的剧本“空爱一场”love's labour's lost里
- 意思是“不胜光荣”

史上最最长的英文单词

Supercalifragilisticexpiadocious

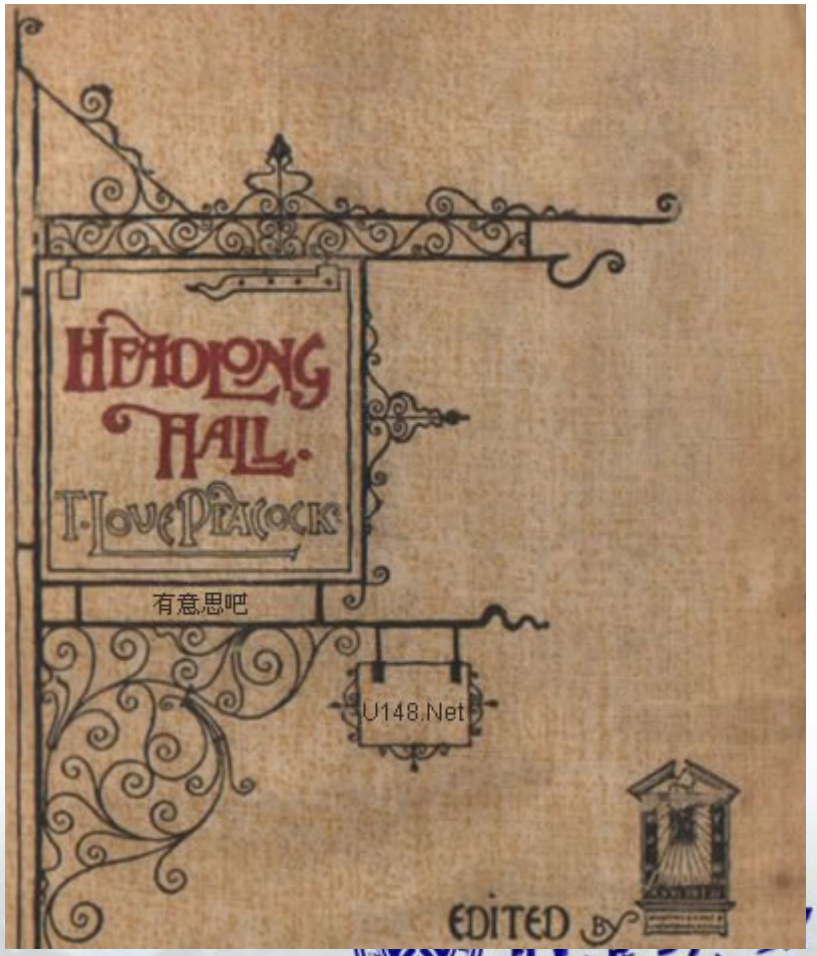
- 由34个字母组成
- 出现在一部名叫Mary Poppins的电影里
- 意思是“好”



史上最最最长的英文单词

Osseocaynisanguineoviscericartilaginine rvomedullary

- 由51个字母组成
- 曾出现在英国作家皮考克的小说**Headlong Hall**中
- 人体构造一术语



史上最最最最长的英文单词

Bababadalgharaghtakamminarronnkonnbronntonnerronntuonnthunntrovarrhounawnskawntoohoohoordenenthurnuk

- 由100个字母组成
- 出现在爱尔兰作家乔埃斯（James Joyce，1882-1942）作品Finnegans Wake的扉页
- 象征亚当和夏娃的堕落

北京大学

史上最……最长的英文单词

methionylglutaminylarginyltyrosylglutamylserylleucylphenylalanylalanylglutaminylleucyllysylglutamyl arginyllysylglutamylglycylalanylphenylalanylvalylprolyphenylalanylvalythreonylleucylglycylaspartylpr olylglycylisoleucylglutamylglutaminylserylleucyllysylisoleucylaspartylthreonylleucylisoleucylglutamylal anylglycylalanylasparthlalanylleucylglutamylleucylglycylisoleucylprolylphenylalanylserylaspartylprolyl leucylalanylaspartylglycylprolylthreonylisoleucylglutaminylaspfraginylalanylthreonylleucylarfinylalany lphenylalanylalanylalanylglycylvalythreonylprolylalanylglutaminylcysteinylphenylalanylglutamylmethi onylleucylalanylleuoylisoleucylarginylglutaminyllysyhistidylprolylthreonylisoleucylprolylisoleucylglycyl leucylmethionyltyrosylalanylasparaginylleucylvalylphenylalanylasparaginyllysyglycylisoleucylaspartylg lutamylphenylalanylthrosylalanylglutaminylcysteinylglutamyllysylvalylglycylvalylaspartylserylvalylleu cylvalylalnylaspartylvalylprolylvalylglutaminylglutamylserylalanylprolylphenylalanylarginylglutaminyl alanylalanylleucylarginylhistidylasparaginyvalylalanylprolylisoleucylprolylisoleucylphenylalanylisoleuc ylphenylalanylisoleucylcysteinylprolylprolylaspartylalanylaspartylaspartylaspartylleucylleucylarginylgl utaminylisoleucylalanylseryltyrosylglycylarginylglycyltyrosylthreonyltyrosylleucylleucylserylarginylala nylglycylvalylthreonylglycylalanylglutamylasparainylarginylalanylalanylleucylprolylleucylasparaginylh istidylleucylvalylalanyllysylleucyllysylglutamyltyrosylasparaginylalanylalanylprolylprolylleucylglutami nylglgycylphenylalanylglycylisoleucylserylalanylprolylaspartylglutaminylvalyllysylalanylalanylisoleucyl aspartylalanylglycylalanylalanylglycylalanylisoleucylserylglycylserylalanylisoleucylvalyllysylisoieucylis oleucylglutamylglutaminylhistidylasparaginylisoleucylglutamylprolylglutamyllysylmethionylleucylalan ylalanylleucyllysylvalylphenylalanylcalylglutaminylprolylmethionlysylalanylalanylthreonylarginylserin e

- **1913个字母“色氨酸合成酶a蛋白质”（一种含有267种氨基酸酶）的化学名**

北京大学

问题分析

- 反复做的事情：选择第n个位置的字母
  - **依次**检查 a, b, c三个字母

    如果某个字母未被选择过；

    1. 将该字母选入字符串；
    2. 标记该字母已经被选择；
    3. 如果全部位置都已选完，打印输出；

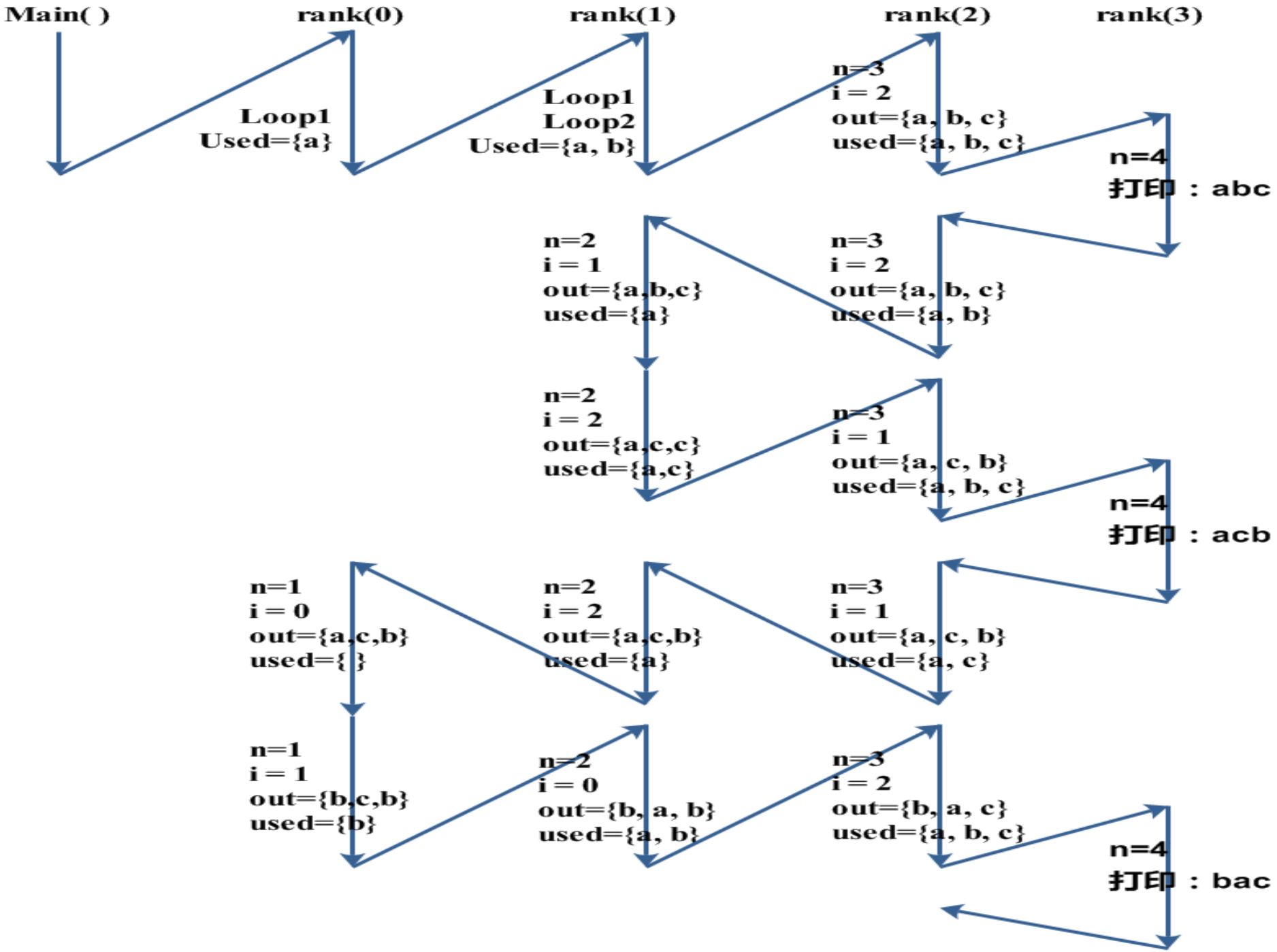
       否则，为下一个位置选择字母；
    4. 把刚刚标记过的字母重新标记为“未选择”；

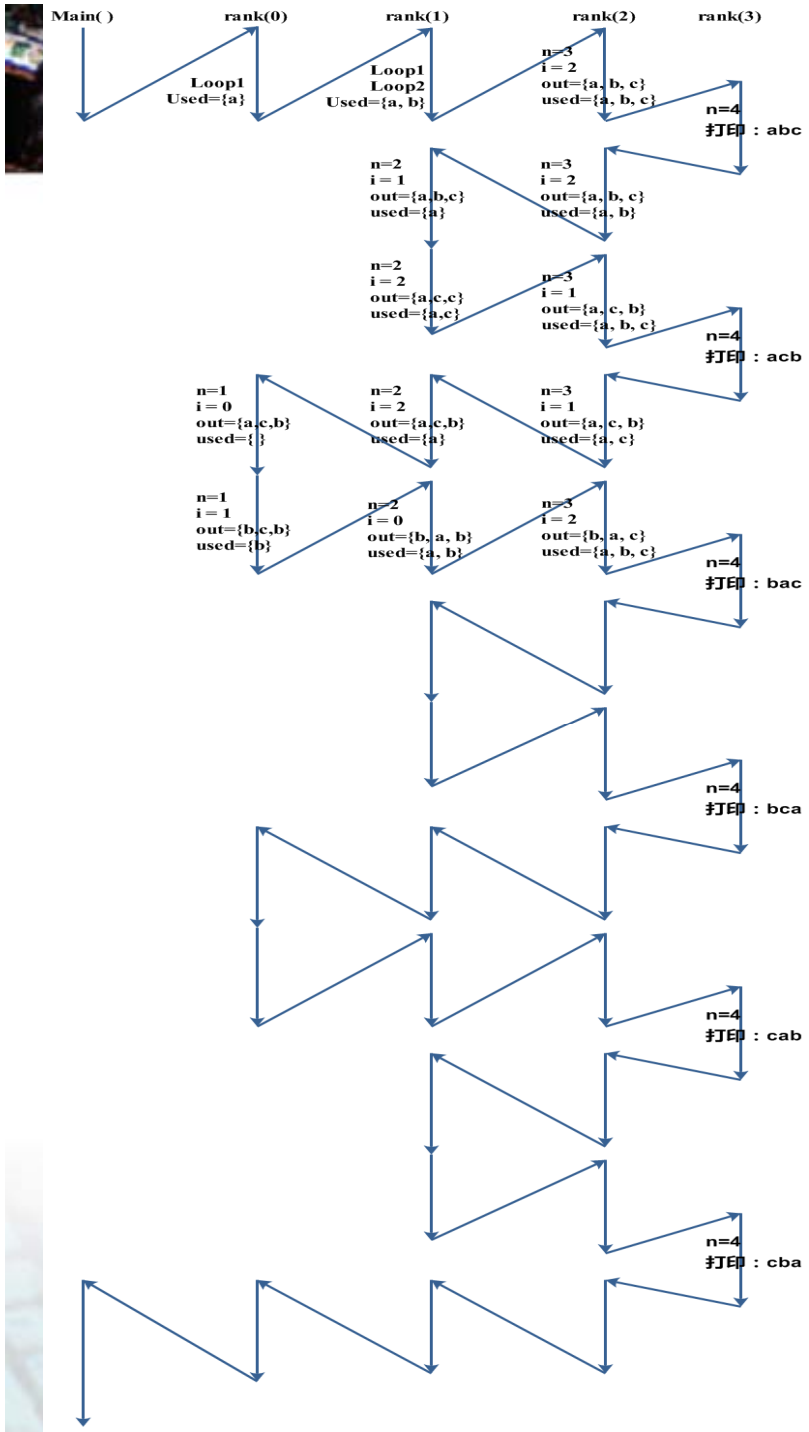
北京大学

问题分析

- 假设一个函数ranker( )能够完成上述事情；
- 每次调用之间的区别在于位置n
  - ranker(1)
  - ranker(2)
  - ranker(3)
  - ranker(4)
- ranker(n)
  - 为第n个位置选择字母

```
char in[3] = {0};                    //存放输入的单词
char out[3] = {0};                   //存放准备输出的字符串
int used[3] = {0};                   //记录哪个字母已经使用过
void ranker(int n)     //为第n个位置寻找字母
{
    for (int i = 0; i < 3; i++)  //依次查看每个字母
    {
        if (!used[i])              //如果某个字母尚未被选
        {
            out[n] = in[i];   //选入该字母
            used[i] = 1;       //标记该字母已经被选择
            if (n == 2)  {cout << out << endl;}
            //如果新字符串中已经有3个字母，打印输出
            else rank(n+1);
            //否则，为下一个位置寻找字母
            used[i] = 0;
            //标记该字母还未被选择，使其重新可被选
        }
}}
```


Main( )
rank(0)
rank(1)
rank(2)
rank(3)
Loop1
Used={a}
Loop1
Loop2
Used={a, b}
n=3
i = 2
out={a, b, c}
used={a, b, c}
n=4
打印：abc
n=2
i = 1
out={a,b,c}
used={a}
n=3
i = 2
out={a, b, c}
used={a, b}
n=2
i = 2
out={a,c,c}
used={a,c}
n=3
i = 1
out={a, c, b}
used={a, b, c}
n=4
打印：acb
n=1
i = 0
out={a,c,b}
used={ }
n=2
i = 2
out={a,c,b}
used={a}
n=3
i = 1
out={a, c, b}
used={a, c}
n=1
i = 1
out={b,c,b}
used={b}
n=2
i = 0
out={b, a, b}
used={a, b}
n=3
i = 2
out={b, a, c}
used={a, b, c}
n=4
打印：bac




```
#include<iostream>
using namespace std;
char in[3] = {0};
char out[3] = {0};
int used[3] = {0};
void ranker(int n)
{
	for (int i = 0; i < 3; i++)
	{
		if (!used[i])
		{
			out[n]=in[i];
			used[i] = 1;
			ranker(n+1);
			used[i] = 0;
		}

	}
	if (n==2) cout<<out<<endl;
}
int main()
{
    cin >> in;
    ranker(0);//从第一个字母开始
    system("pause");
    return 0;
}
```

回溯

描述反复做的事情时：

- 站在第n步的状态下进行分析
  - 递归
    - 在第n步的情况下，枚举出第n+1步的所有可能，向所有可能的方法形成递归；
  - 回溯
    - 恢复影响以后的选择“现场”（使重新的选择成为可能）

探索型递归问题的解法

- 第n步需要做什么？
  - 对于面前的每种选择
    - ① 把该做的事情做了！
    - ② 判定是否得到解！
    - ③ 递归（调用第n+1步）！
    - ④ 看是否需要回溯！

分书问题

- 问题
  - 有编号分别为1，2，3，4，5的五本书，准备分给A,B,C,D,E五个人，每个人阅读兴趣用一个二维数组加以描述：

$$Like[i][j] = \begin{cases} 1 & i\text{喜欢书}j \\ 0 & i\text{不喜欢书}j \end{cases}$$

人＼书	0	1	2	3	4
A	0	0	1	1	0
B	1	1	0	0	1
C	0	1	1	0	1
D	0	0	0	1	0
E	0	1	0	0	1

  - 请写一个程序，输出所有分书方案，让人人皆大欢喜。

分书问题

- 解决思路：

① 试着给第i个人分书，先试分0号书，再分1号书，分2号书…，分j号书，…,分4号书。

② 当“第i个人喜欢j书，且j书尚未被分走”时。第i个人能够得到第j本书。

③ 如果不满足上述条件，则什么也不做（循环返回条件）。

分书问题

④ 若满足条件，则做三件事情：

- 做事：将j书分给i，同时记录j书已被选用；
- 判断：查看是否将所有5个人所要的书分完，
  - 若分完，则输出每个人所得之书；
  - 若未分完，去寻找其他解决方案；
- 回溯：让第i人退回j书，恢复j书尚未被选的标志。

分书问题

1、使用二维数组定义阅读喜好用：

- **int like[5][5]**

={{0,0,1,1,0},{1,1,0,0,1,},{0,1,1,0,1},{0,0,0,1,0}{0,1,0,0,1}};

2、使用数组**book[5]**记录书是否已被选用。

int book[5]={0,0,0,0,0};

3、使用数组**take[5]**存放第几个人领到了第几本书；

```
void trybook(int i) {
    for (int j=0; j<=4; j=j+1)      //对于每本书，j为书号；
    {
      if ((like[i][j]>0)&&(book[j]==0))
                          //若第i个人喜欢第j本书，且这本书没有被分出；
      {
                take[i]=j;          //把第j号书分给第i个人
                book[j]=1;          //标记第j号书已被分出
                if (i==4)
                //若第5个人也已经拿到了书，则书已分完，输出分书方案
                {
                        n = n + 1;          //让方案数加1
                        cout <<"第"<<n<<"个方案"<<endl;
                        for (int k=0; k<=4; k=k+1)
                                cout<<take[k]<<"号书给"<<char(k+65);
                        cout <<endl;
                }
                else                //若书还没分完, 继续给下一个人找书；
                {
                        trybook(i+1);
                }
                book[j]=0;          //回溯，把书标记为未分，找其他解决方案；
      }
    }
}
```

分书问题

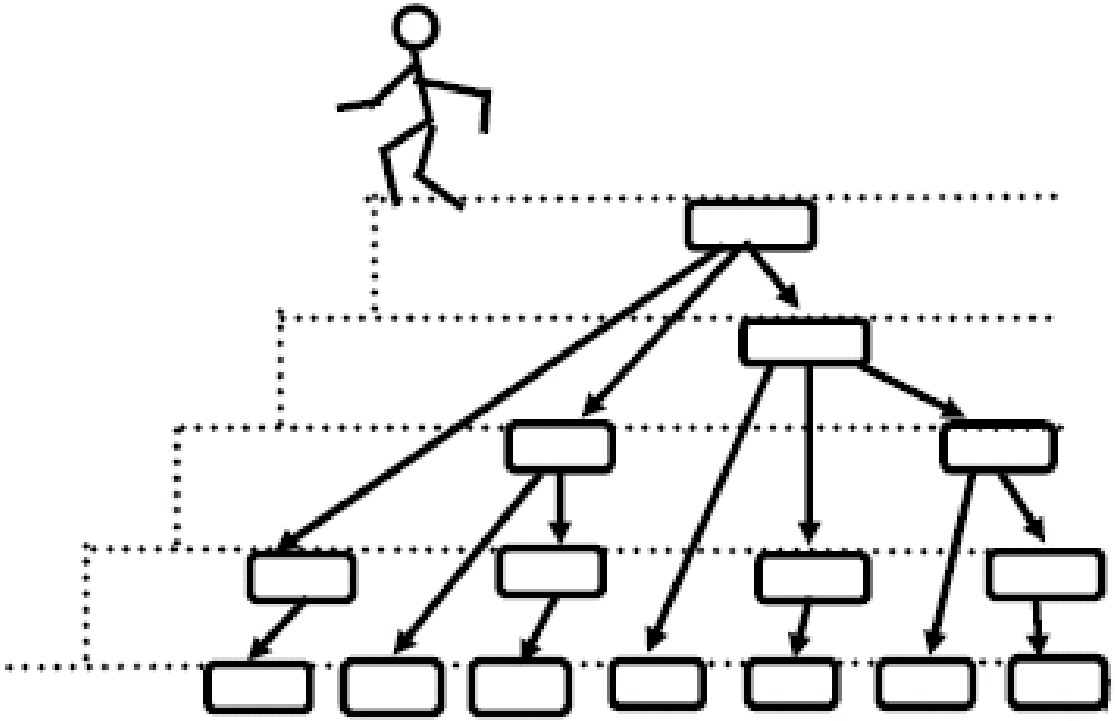
```
#include<iostream.h>
int
    like[5][5]={{0,0,1,1,0},{1,1,0,0,1,},{0,1,1,0,1},{0,0,0,1,0
    },{0,1,0,0,1}};
int book[5], take[5], n;      //n表示分书方案的总数
int main()
{
    int n=0;                  //分书方案数预置0
    trybook(0);               //从“为第0个人分书”开始执行
    return 0;
}
```

探索型递归问题的解法

- 第n步需要做什么？
  - 对于面前的每种选择
    - ① 把该做的事情做了！
    - ② 判定是否得到解！
    - ③ 递归（调用第n+1步）！
    - ④ 看是否需要回溯！

下楼问题

- 问题：
  - 从楼上走到楼下共有h个台阶，每一步有3种走法：走1个台阶；走2个台阶；走3个台阶。问可以走出多少种方案？将所有的方案输出。



直接考虑第s步时

- 有三种选择（走1、2、3步）
- 每个选择下有三种可能性：
  - 如果剩下的台阶小于想要走的步数
    - 返回
  - 如果剩下的台阶恰好等于要走的步数；
    - 打印输出；
  - 如果剩下的台阶大于想要走的步数
    - 走下去；

下楼问题

- 问题分析
  - 要枚举出所有的可能方案，所以是一个典型的递归回溯问题；
    - i 表示还剩几级台阶
    - s 表示到目前该走第几步
    - take[s] 表示第s步应该走几级台阶
  - 当走到底时，take[1]~take[s]，就是一路走来的过程，即一种成功的走法。

下楼问题

- 第s步有3种可能，用for循环枚举。
- 第s步走了j个台阶后，有三种结果：
  - ① for（j=1;j<=3;j++)
  - ② i<j.说明第s步走的台阶比剩下的阶梯数还多。j不可取。（递归函数的出口）
  - ③ i=j.说明第s步正好走完剩下的阶梯，得到一个解决方案。
  - ④ i>j.说明第s步走完后，还剩下i-j级阶梯没有走，可以走第s+1步。递归调用。

```
int take[99];
int num = 0;                    //num表示解决方案的总数
void Try(int i, int s) {    //i表示所剩台阶数
   for (int j = 3; j > 0; j--) //枚举第s步走的台阶数j
   {
     if (i<j)      //如果所剩台阶数i小于允许走的台阶数j
              continue;
     take[s] = j;              //记录第s步走j个台阶;
     if (i == j)               //如果已经走完全部台阶;
     {
              num++;        //方案数加1
              cout << "solution" << num << ": ";
              for (int k = 1; k <= s; k++)
                       cout << take[k];
              cout << endl;
     }
     else
              Try(i - j, s + 1); //尚未走到楼下
   }}
```

下楼问题

```
int main()
{
    int h = 0;
    cout << "how many stairs : ";
    cin >> h;
    Try(h,1);  //有h级台阶要走，从第一步开始走
    cout << "there are " << num << " solutions."
    << endl;
    return 0;
}
```

探索型递归问题的解法

- 第n步需要做什么？
  - 对于面前的每种选择
    - ① 把该做的事情做了！
    - ② 判定是否得到解！
    - ③ 递归（调用第n+1步）！
    - ④ 看是否需要回溯！

拼词问题

```
int len;                                    //单词中字母的个数
char in[26] = {0};                          //存放输入的单词
char out[26] = {0};                         //存放准备输出的字符串
int used[26] = {0};                         //记录哪个字母已经使用过
void rank(int n)          //n为新产生字符串中字母的个数
{
  if (n > len)                //如果新字符串中已经有len个字母
  {   cout << out << endl;   }
  else
  {
   for (int i = 0; i < len; i++)  //挨个查看输入单词中的字母
   {
          if (!used[i]) //如果某个字母尚未被选入字符串
          {
                 out[n-1] = in[i];   //将该字母加入字符串
                 used[i] = 1;        //标记该字母已经被选用
                 rank(n+1);          //寻找更长的字符串
                 used[i] = 0;        //回到为选择第i字母的状态
          }
}}}
```

分书问题

```
void trybook(int i) {
    for (int j=0; j<=4; j=j+1) //对于每本书，j为书号；
    {
        if ((like[i][j]>0)&&(book[j]==0)) {
                    //若第i个人喜欢第j本书，且这本书没有被分出；
            take[i]=j;      //把第j号书分给第i个人
            book[j]=1;      //标记第j号书已被分出
            if (i==4) {     //若第5个人拿到了书，则输出分书方案
                n = n + 1;      //让方案数加1
                cout <<"第"<<n<<"个方案"<<endl;
                for (int k=0; k<=4; k=k+1)
                    cout<<take[k]<<"号书给"<<char(k+65);
                cout <<endl;
            }
            else{           //若书还没分完, 继续给下一个人找书；
                trybook(i+1);
            }
            take[i]=-1;     //使i把书退还，寻找其他解决方案；
            book[j]=0;      //相应的也把书标记为未分；
        }
    }
}
```

例题：下楼问题

```
int take[99];
int num = 0;
void Try(int i, int s) {  //i: 剩余台阶数；s: 第s步；j: 要走的步数
    for (int j = 3; j > 0; j--)
    {
        if (i<j)     //如果所剩台阶数i小于允许走的台阶数j
                continue;
        take[s] = j;            //记录第s步走j个台阶；
        if (i == j)             //如果已经走完全部台阶；
        {
                num++;      //方案数加1
                cout << "solution" << num << ": ";
                for (int k = 1; k <= s; k++)
                        cout << take[k];
                cout << endl;
        }
        else
                Try(i - j, s + 1); //尚未走到楼下
        take[s] = 0;
    }}
```

好好想想,有没有问题?

谢　谢　！

北京大学