《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

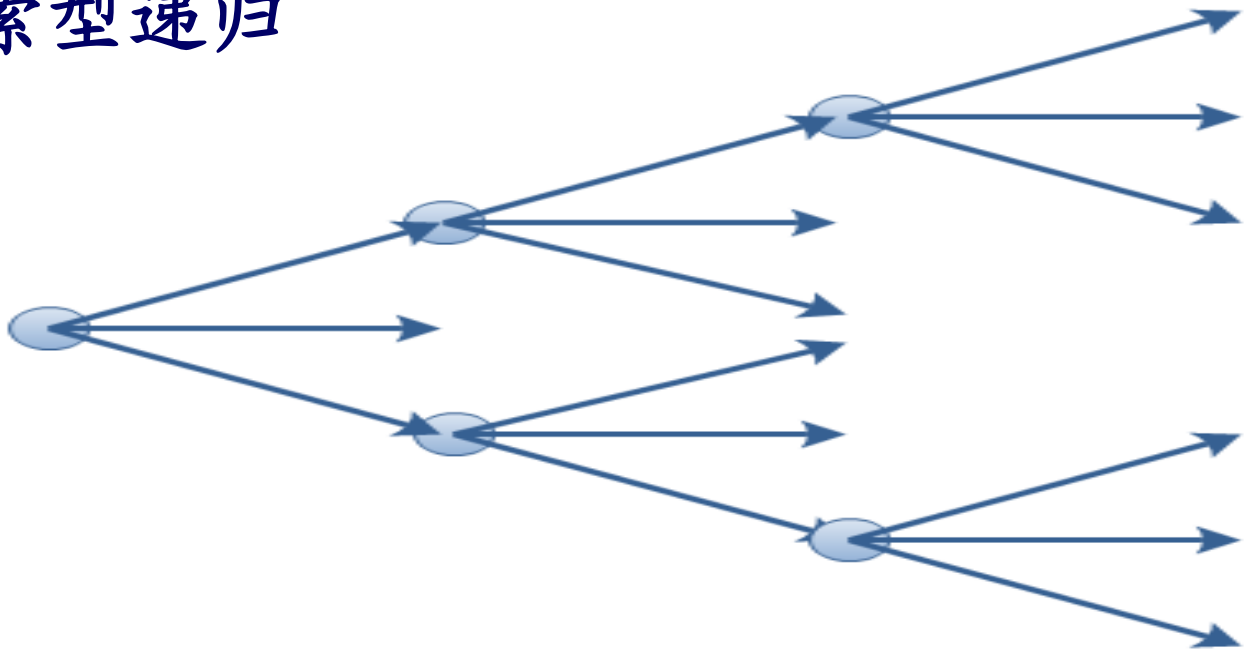
2010年12月8日

北京大学

递归问题

- 递推型递归

- 探索型递归



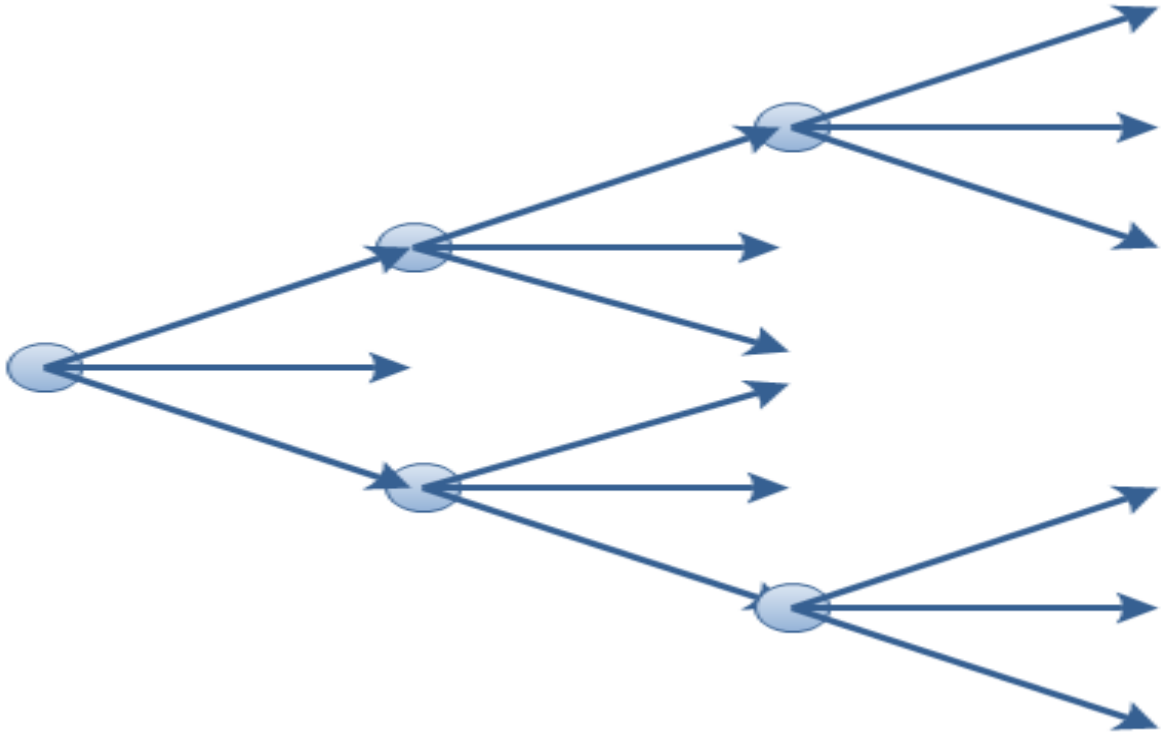
递推型递归的解法

- 发现递归
  - 对解决方案进行分析，若发现总是需要反复执行相同的操作；
  - 且后一步操作需要以前一步操作的结果为前提；

  （如：进制转换、Fibonacci、逆波兰表达式...）
- 编写递归
  - 分析每一步需要做哪些操作
    - 重点：前一步操作与后一步操作之间的关系；
  - 对边界/极限情况进行分析，作为递归终止条件；

探索型递归问题

- 问题特点
  - “下一步有多种选择”！



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get，则打印出：

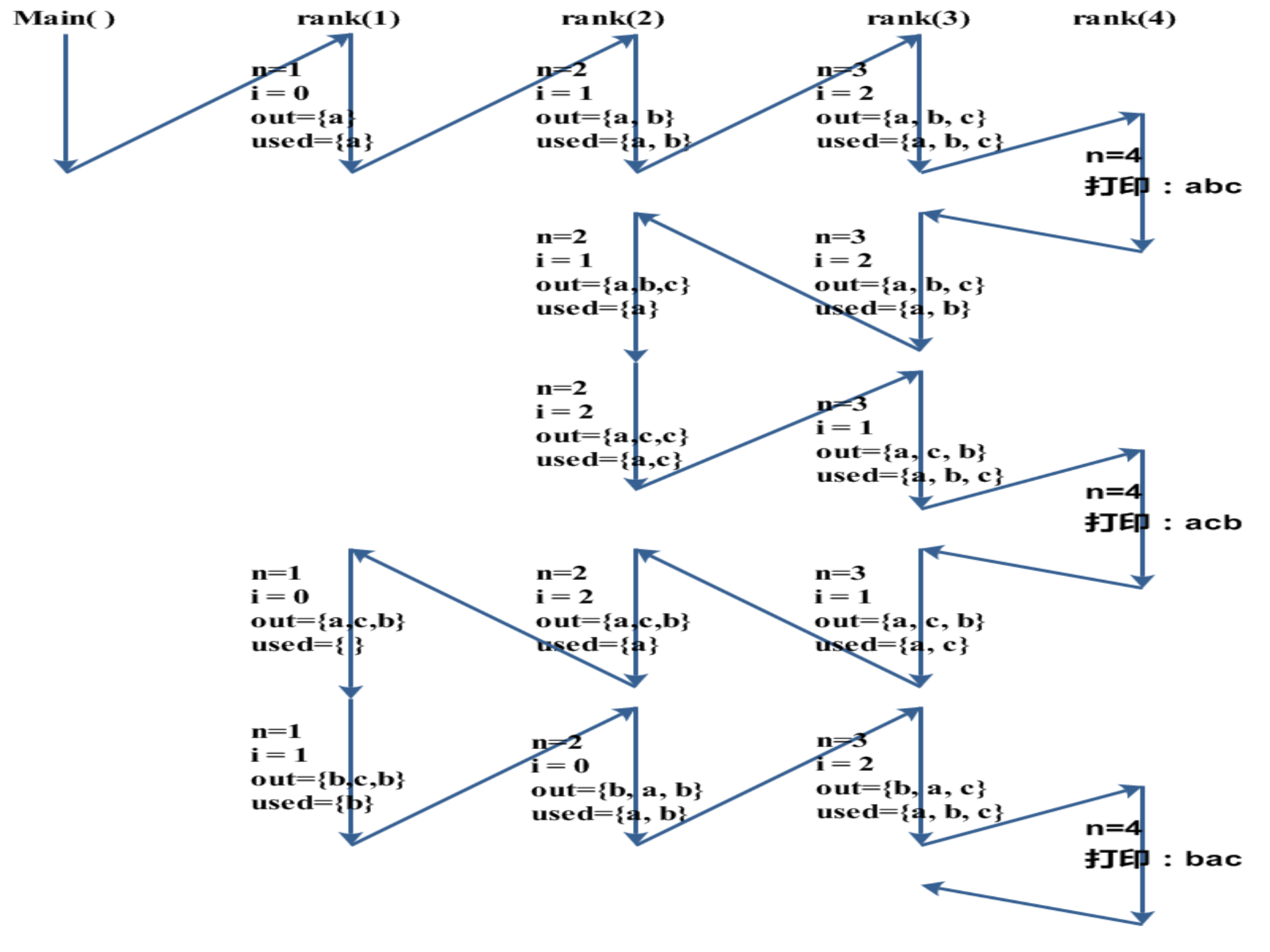
    get
    gte
    egt
    etg
    tge
    teg

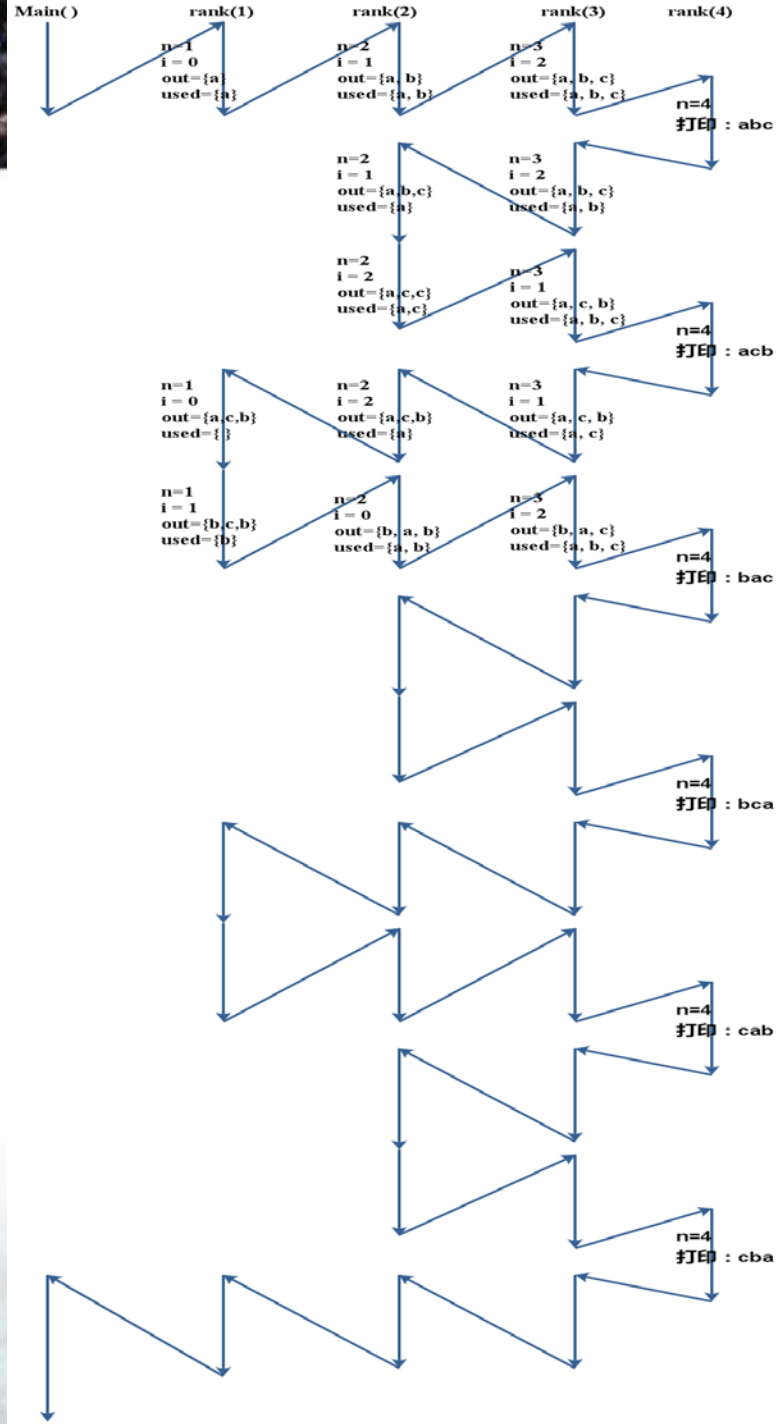
```
int len;                                    //单词中字母的个数
char in[26] = {0};                          //存放输入的单词
char out[26] = {0};                         //存放准备输出的字符串
int used[26] = {0};                         //记录哪个字母已经使用过
void rank(int n)          //n为新产生字符串中字母的个数
{
   if (n > len)                  //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
            }
}}}
```

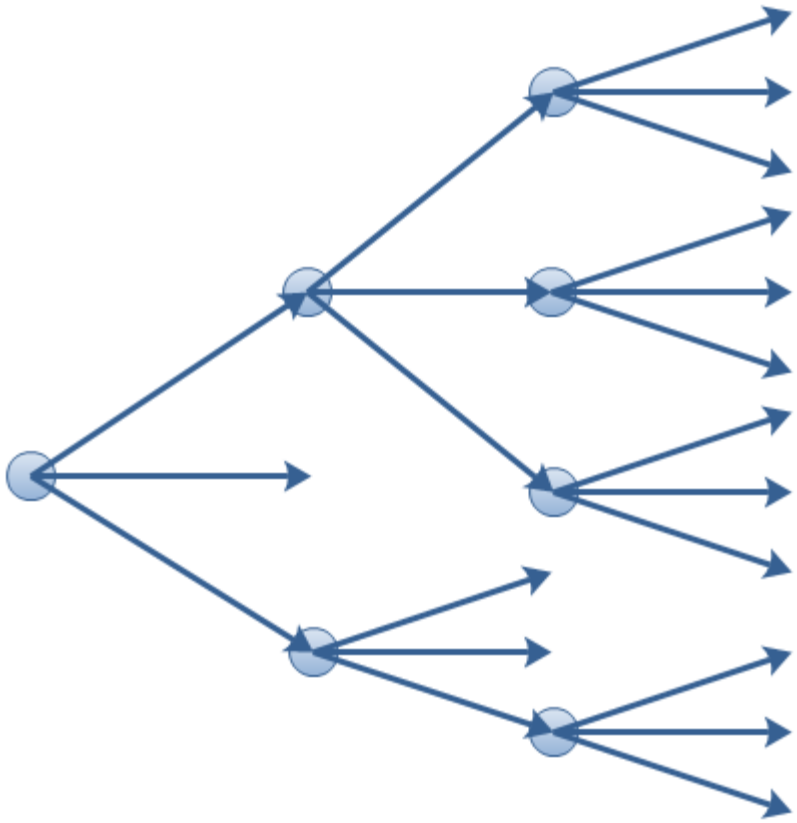
```
#include<iostream>
using namespace std;

char in[26] = {0};
char out[26] = {0};
int used[26] = {0};
int len;
void rank(int n)
{
    if (n > len)
    {   cout << out << endl;    }
    else
    {
       for (int i = 0; i < len; i++)
       {
                    if (!used[i])
                    {
                                 out[n-1] = in[i];
                                 used[i] = 1;
                                 rank(n+1);
                                 used[i] = 0;
                    }
}}}

int main()
{
    cin >> in;
    len = strlen(in);
    rank(1);          //从新产生字符串第一个字母开始
    return 0;
}
```


Main( )
rank(1)
rank(2)
rank(3)
rank(4)
n=1
i = 0
out={a}
used={a}
n=2
i = 1
out={a, b}
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


Main()
rank(1)
rank(2)
rank(3)
rank(4)
n=1
i = 0
out={a}
used={a}
n=2
i = 1
out={a, b}
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
n=4
打印：bca
n=4
打印：cab
n=4
打印：cba



北京大学

探索型递归的解法——回溯

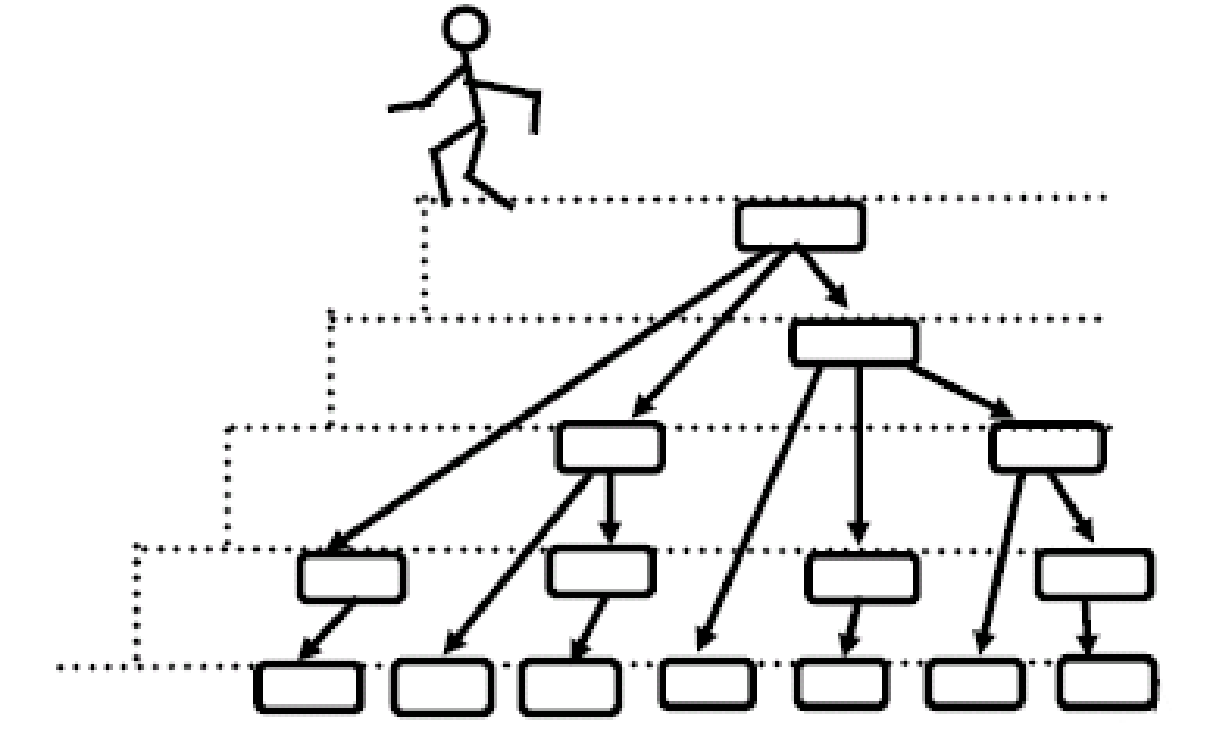
- 回溯
  - 按照深度优先的策略，在包含所有解的树中，从根结点出发搜索。搜索至树的任一结点时，先判断该结点是否包含问题的解。若包含则进入该子树，继续按深度优先的策略进行搜索。若不包含解，则逐层退回到其祖先结点的状态，再向其他节点搜索。
  - 关键点：在第n步的情况下，枚举第n+1步的所有可能，向所有可能的方法形成递归；

回溯算法的特点

- 递归函数中通常包含以下元素
  - 判定当前状况是否是问题的解?
    - 若解决return，否则进行如下步骤:
  - 构造循环探测每种分支情况
    - 完成需要重复完成的动作;
    - 给出发起递归的语句;
  - 递归回溯时需要完成动作;

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。



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;
}
```

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：

- 第一件事：将j书分给i，同时记录j书已被选用；
- 第二件事：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。
- 第三件事：回溯，去寻找其他解决方案：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

- 特点：
  - 每一步所做动作相同；
- 重点：
  - 考虑第n步时做什么！

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！

好好想想,有没有问题?

谢　谢　！

北京大学