1.5 回溯算法

主讲老师：党东

回溯算法

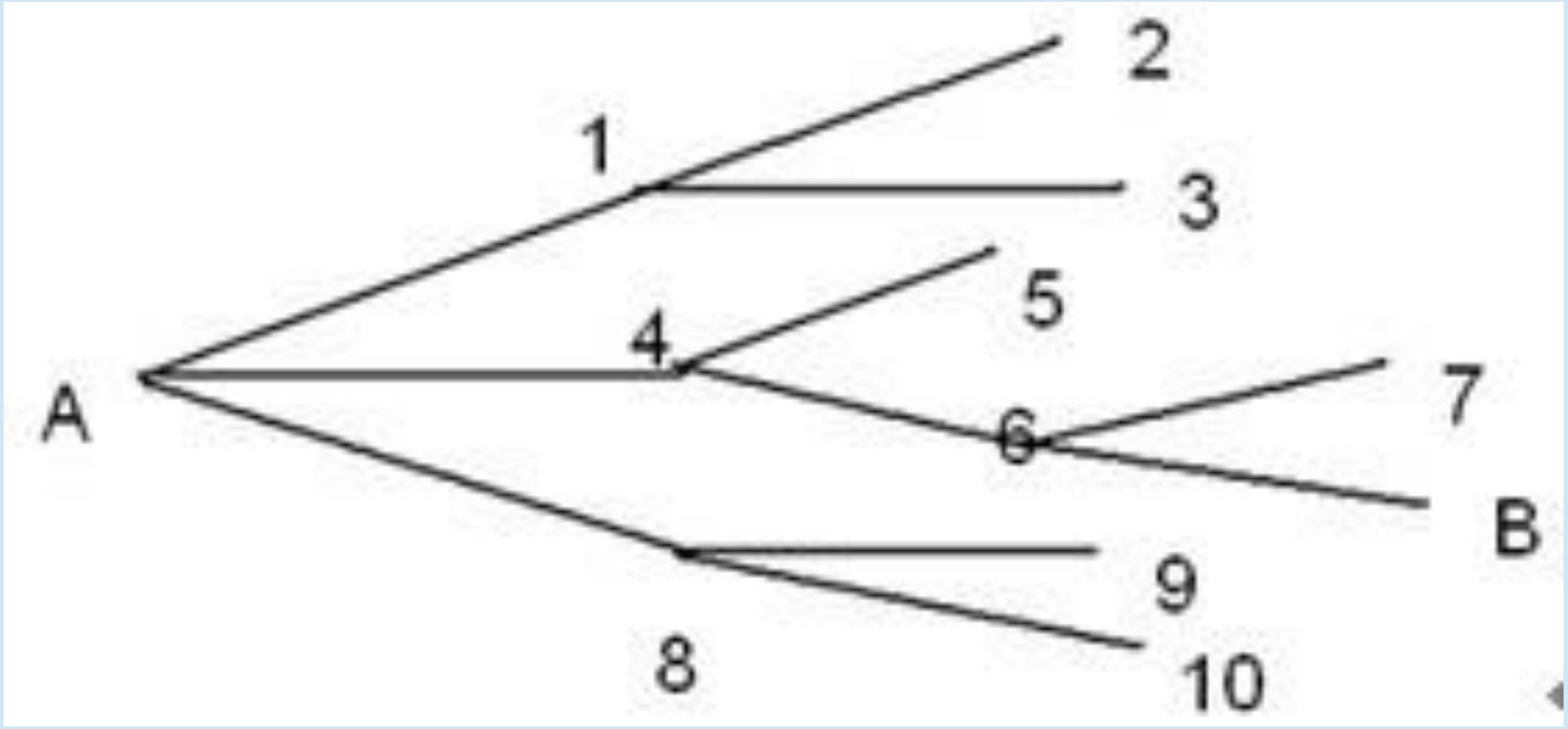
【定义】从问题的某一种可能出发，搜索从这种情况出发所能达到的所有可能，当这一条路走到“尽头”而没达到目的地的时候，再倒回上一个出发点，从另一个可能出发，继续搜索。这种不断“倒回一步”寻找解的方法，称作“回溯法”。

回溯即是较简单、较常用的搜索策略,实质就是一种搜索策略。

【案例】从A到B的路线

如：找一条从A到B的路线

从A到B的路线:A---4---6---B



【案例】从A到B的路线

递归回溯法算法框架[一]

```
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}
                  }
  }
```

【案例】从A到B的路线

递归回溯法算法框架[二]

```
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}
         }
   }
```

【例1】全排列 --1221

【问题描述】编程列举出1、2、…、n的全排列，要求产生的任一个数字序列中不允许出现重复的数字

【文件输入】输入n(1<=n<=9)

【文件输出】有1到n组成的所有不重复数字的序列，每行一个序列

【样例输入】3

【样例输出】

1 2 3

1 3 2

2 1 3

2 3 1

3 1 2

3 2 1

【例1】全排列 --1221

【算法分析】

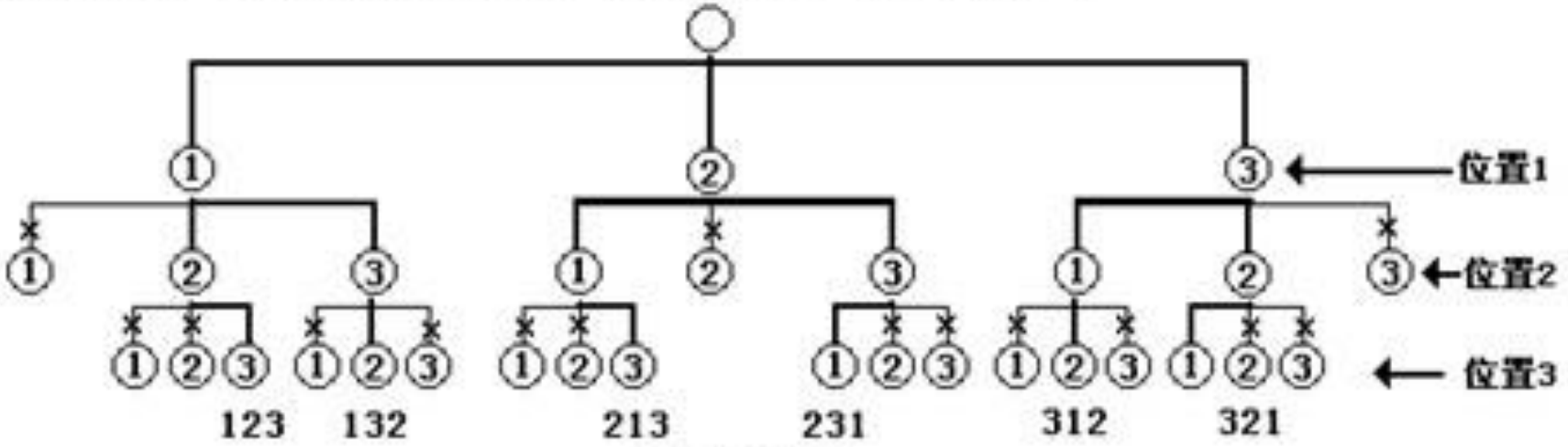
【思路点拨】深度优先搜索算发的本质是：按照深度优先的策略搜索问题的解答树。因此要使用它解决问题，应该先画出问题小范围的解答树。

具体到本题，我们假设 n=3 时，如下图：位置 1 可以放置数字 1、2、3；位置 2 可以放置数字 1、2、3；位置 3 可以放置数字 1、2、3，但是当位置 1 放了数字 1 后位置 2 和位置 3 都不能在放 1，因此画树的约束条件是：各位置的数字不能相同。

【例1】全排列 --1221

【算法分析】

我们画“解答树”时，根结点一般是一个空结点，根结点下面的第1、2、3三层分别对应位置1、位置2、位置3，用“×”标示的分支表示该结点不满足约束条件，不能被扩展出来：



（解答树）

我们用递归过程来描述“解答树”的深度优先搜索

【例1】全排列 --1221


```
void f(int k)    //搜索第k层结点（向第k个位置放数）
{
        int i;
        if (k==n+1) {for(i=1;i<=n;i++)cout<<a[i]<<" ";cout<<endl;}
                                    // 如果搜索到一条路径，则输出一种解
        else for(i=1;i<=n;i++)
                                    //每一个结点可以分解出n个子结点；
        if(b[i]==0) //如果能生成第k层的第i个结点；
        {
                a[k]=i;    //第k个位置为数字i；
                b[i]=1;    //标记数字i已用
                f(k+1);    //扩展第k层的第i个结点(向第k+1个位置放数)
                b[i]=0;    //向上回溯，并恢复数据
        }
}
```

【思考1】无重复元素的全排列 --1248

【问题描述】输入n（<=10）个不同的小写字母，输出n个字符的全部排列。

【样例输入】abc

【样例输出】

1:abc
2:acb
3:bac
4:bca
5:cab
6:cba

【思考1】无重复元素的全排列 --1248

【代码提示】

```
int m[31];
int vis[150];  //当前字母是否已使用，非当前位置
int n,l;
long long ncount=1;  //计数
char s[10];
```

【思考1】无重复元素的全排列 --1248


```
int isearch(int x) //回溯代码
{       if(x==n+1) //第n个位置已放入值
        {
                cout<<ncount<<":";
                for(int i=1;i<=n;i++)
                        cout<<(char)m[i];
                cout<<endl;
                ncount++;
        }
        for(int i=0;i<l;i++)
        {
                if(!vis[i])   //字母i未使用
                {
                        m[x]=s[i];        //将数值i放入第X位置
                        vis[i]=1;       //数值i已使用
                        isearch(x+1); //进行下一个位置值的选择
                        m[x]=0;         //恢复状态
                        vis[i]=0;       //恢复状态
                }
        }
}
```

【思考1】无重复元素的全排列 --1248

【代码提示】

```
int main()    //主函数
{
        cin>>s;
        字符数组 S 中的元素排序
        l=strlen(s);
        n=l;          //元素数
        isearch(1);
        return 0;
}
```

【思考2】有重复元素的全排列--1280

【问题描述】输入n(<=10)个小写字母(可能重复)，输出n个字符的全部排列。

【样例输入】abaab

【样例输出】

1:aaabb
2:aabab
3:aabba
4:abaab
5:ababa
6:abbaa
7:baaab
8:baaba
9:babaa
10:bbaaa

【思考3】

【思考3】求在N个元素中取出M个的选排列问题：

①每个元素只能取一次。

②每个元素可以取任意多次（即可重复的排列）。

【思考4】无重复元素的组合 --1282

【问题描述】输入一串小写字母（无重复字母），从中取出k个字母，输出组合情况(按照字典顺序输出)。

【样例输入】

abcd

3

【样例输出】

abc

abd

acd

bcd

【思考5】有重复元素的组合 --1283

【问题描述】输入一串小写字母（小于30个字母，有重复字母），从中取出k个字母，输出组合情况(按照字典顺序输出)。

【样例输入】

aabbcc

4

【样例输出】

1:aabb

2:aabc

3:aacc

4:abbc

5:abcc

6:bbcc

阶段练习

- 全排列【1221】
- 无重复元素的全排列【1248】
- 有重复元素的全排列【1280】

【例2】素数环 --1274

【问题描述】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。

【文件输入】n(1<= n <= 11)

【文件输出】按照字典顺序不重复的输出所有解，相邻两数之间用一个空格，若无解输出“NO”。

【样例输入】2

【样例输出】

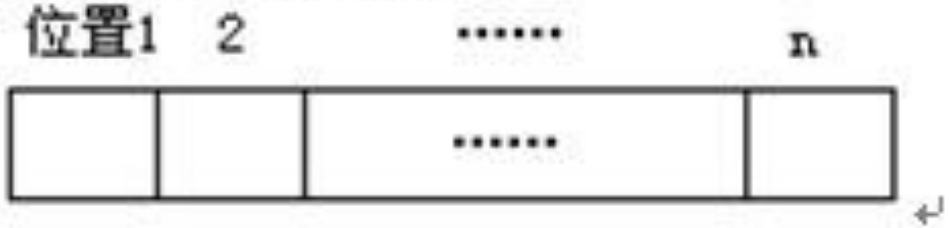
1 2

2 1

【例2】素数环 --1274

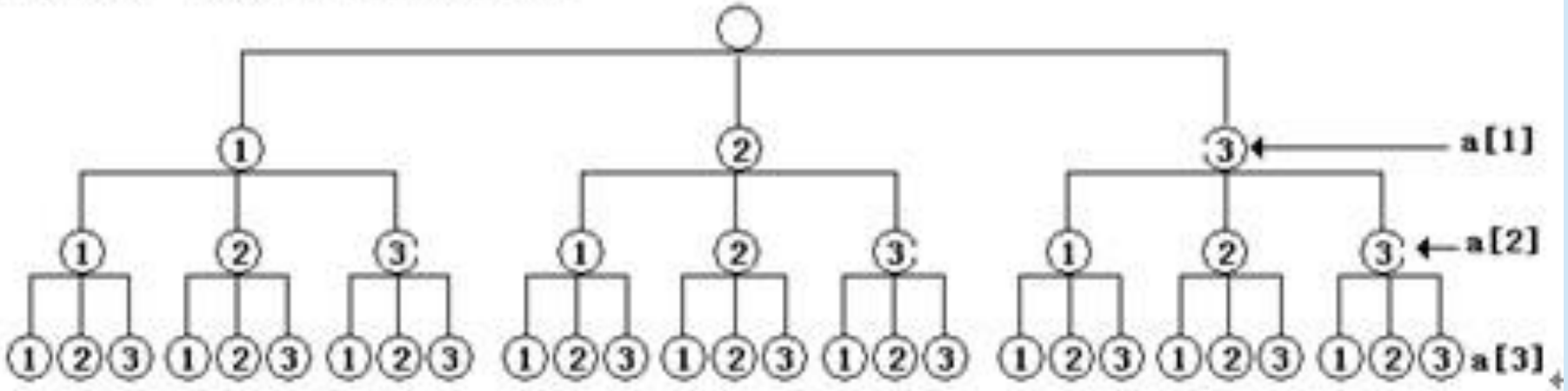
【算法分析】

【问题分析】这是一道搜索的题目。从位置 1 开始，每个空位有 n 个可以填的数字，只要填进去的数合法：



【算法设计】

步骤 1：画解答树（n=3）：



【例2】素数环 --1274

步骤2：确定约束条件和剪枝函数

约束条件：

①.已用过的数字不能再用used[j]=0(0表示未使用，1表示已使用)

②.并且在a[i]中要填的数字j与左边a[i-1]和是素数ok(i-1,j)

这里我们用数组used[j]来表示数字j是否已经使用过：

函数Ok(i,j)是判断a[i]+j是否是素数(其中j是待填入a[i]的数)：

```
int ok(int i,int j)
{    int k,s,p;
     if(i==0)return 1;
      p=1;s=a[i]+j;
     for(k=2;k<=int(sqrt(s));k++) if(s%k==0){p=0;break;}
     if(i==n-1)//由于是环,故当j是第n个数时应与a[1]的和也是素数
     {    s=a[1]+j;
          for(k=2;k<=int(sqrt(s));k++) if(s%k==0){p=0;break;}
     }
     return p;
}
```

【例2】素数环 --1274

步骤3：写核心搜索程序：

```
void try(int i) //试着给a[i]填数
{    int j;
     if(i>n){m++;print();} //如果找到一种填法,则输出
     for(j=1;j<=n;j++)   //a[i]中可以填入数字1~n
          if(!used[j]&&ok(i-1,j)) //如果符合约束条件则填入
          {    a[i]=j;
               used[j]=1;   //在a[i]中填入数字c,并标记数字c已用过
               try(i+1);       //试着给a[i+1]填数
               used[j]=0;   //回溯到上层,并将used[j]清,数字j可用
          }
}
```

请同学们自己将该程序完善

- 全排列【1221】
- 无重复元素的全排列【1248】
- 有重复元素的全排列【1280】
- 无重复元素的组合【1282】
- 有重复元素的组合【1283】
- 素数环【1274】