导入

有n件物品，每件物品有一个体积Vi，求从中取出若干件物品能够组成的不同体积和有多少种可能。例如，n=3，Vi=(1,3,4)，那么输出6，6种方案不同体积有1，3，4，5，7，8.（N<=20,1<=Vi<=500）

输入：
3
1 3 4
输出：
6

分析

样例：有3件物品的体积1 3 4

组合情况：

选一件物品 1　3　4

选两件物品 1、3　1、4　3、4　　所有物品都可以选或者不选

选三件物品 1、3 、4

分析

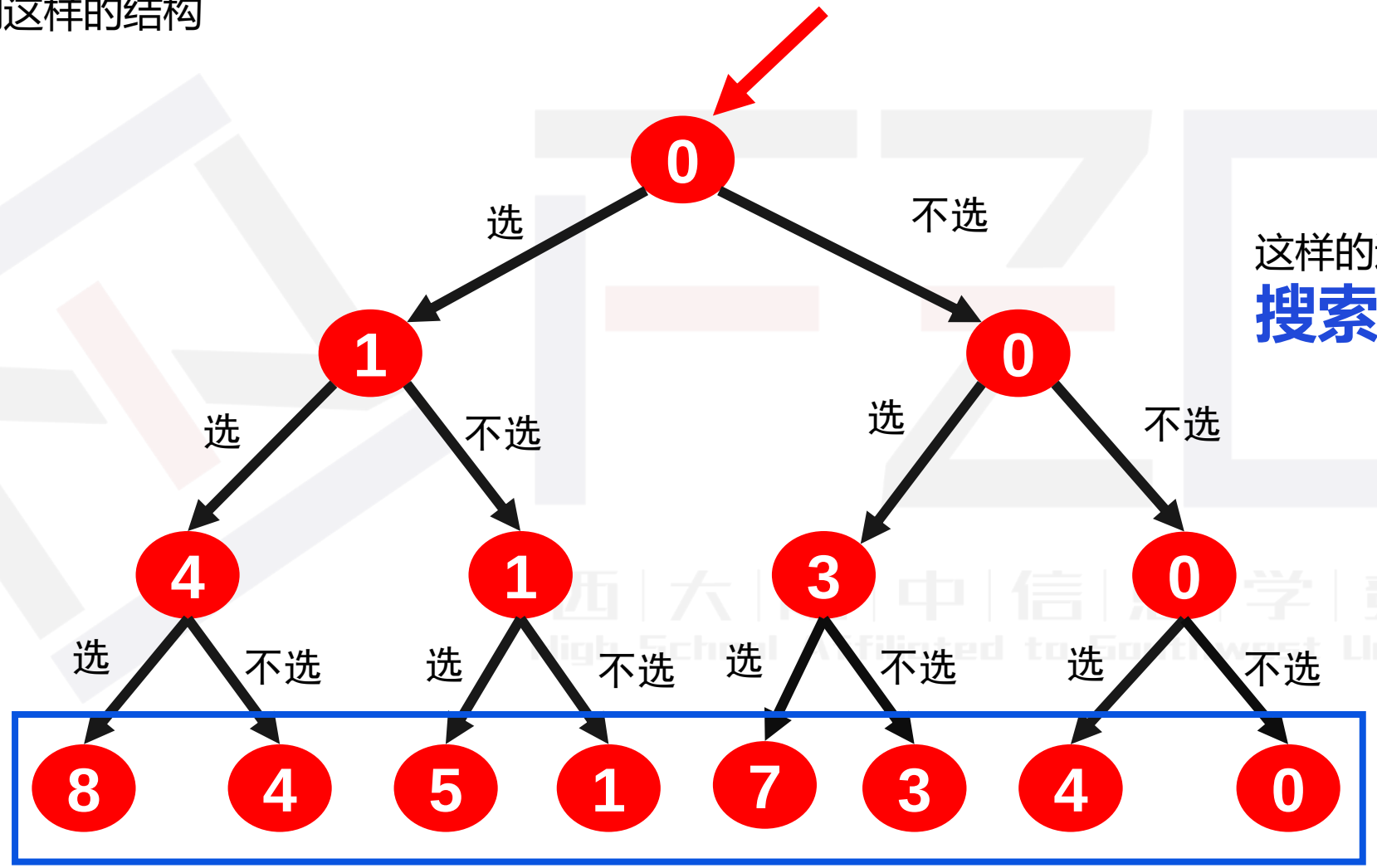
那我们可以得到这样的结构

结点的值表示当前的组合值

对于第一个物品

对于第二个物品

对于第三个物品



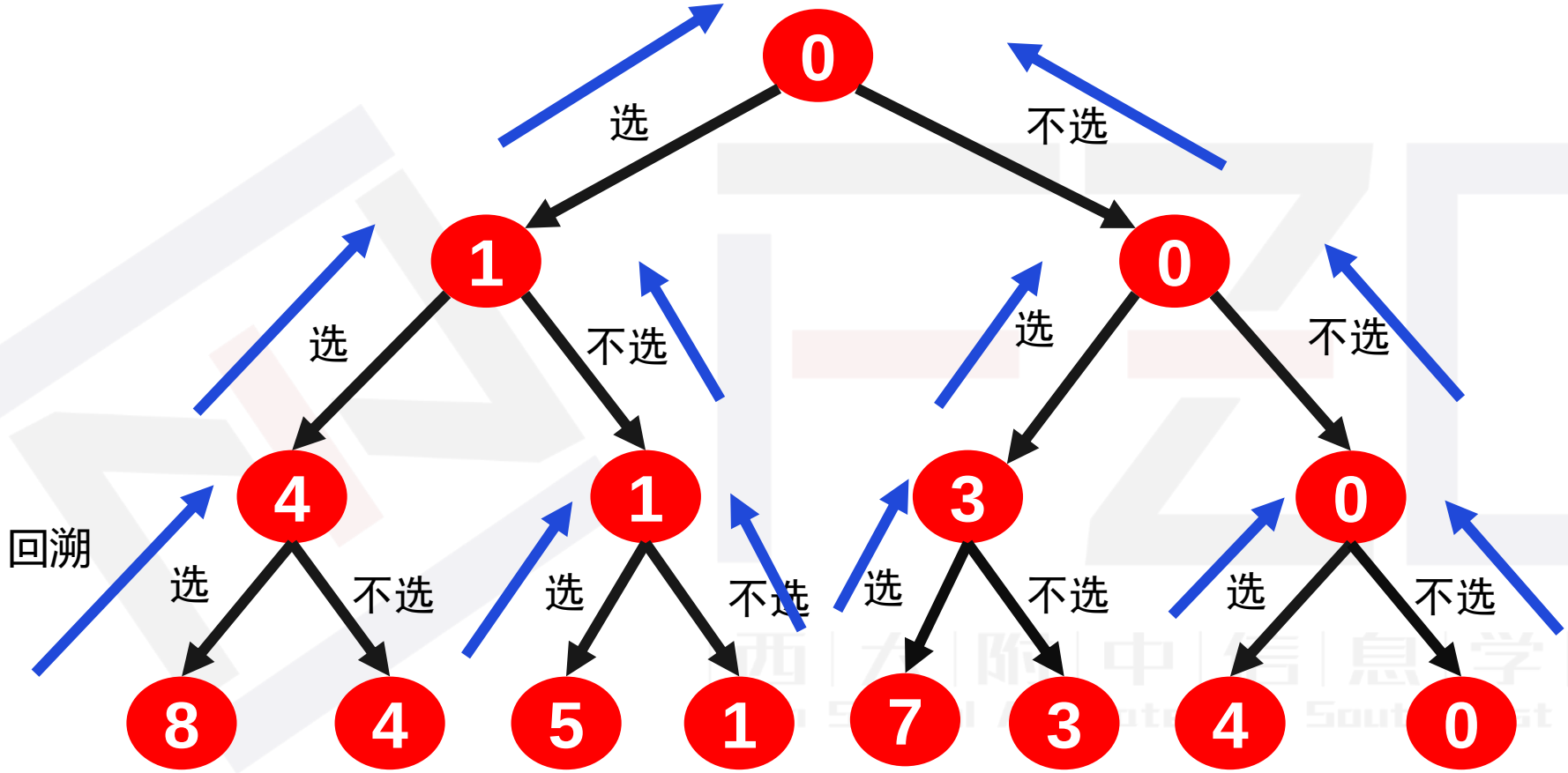
这样的过程称为

搜索

信息学

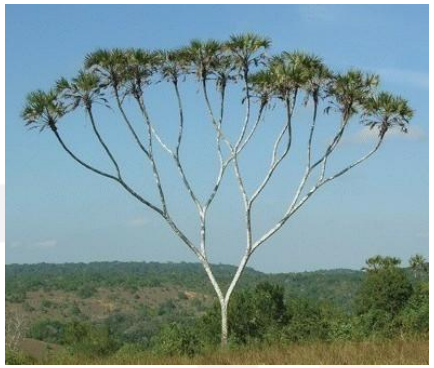
初识DFS (Deep First Search)

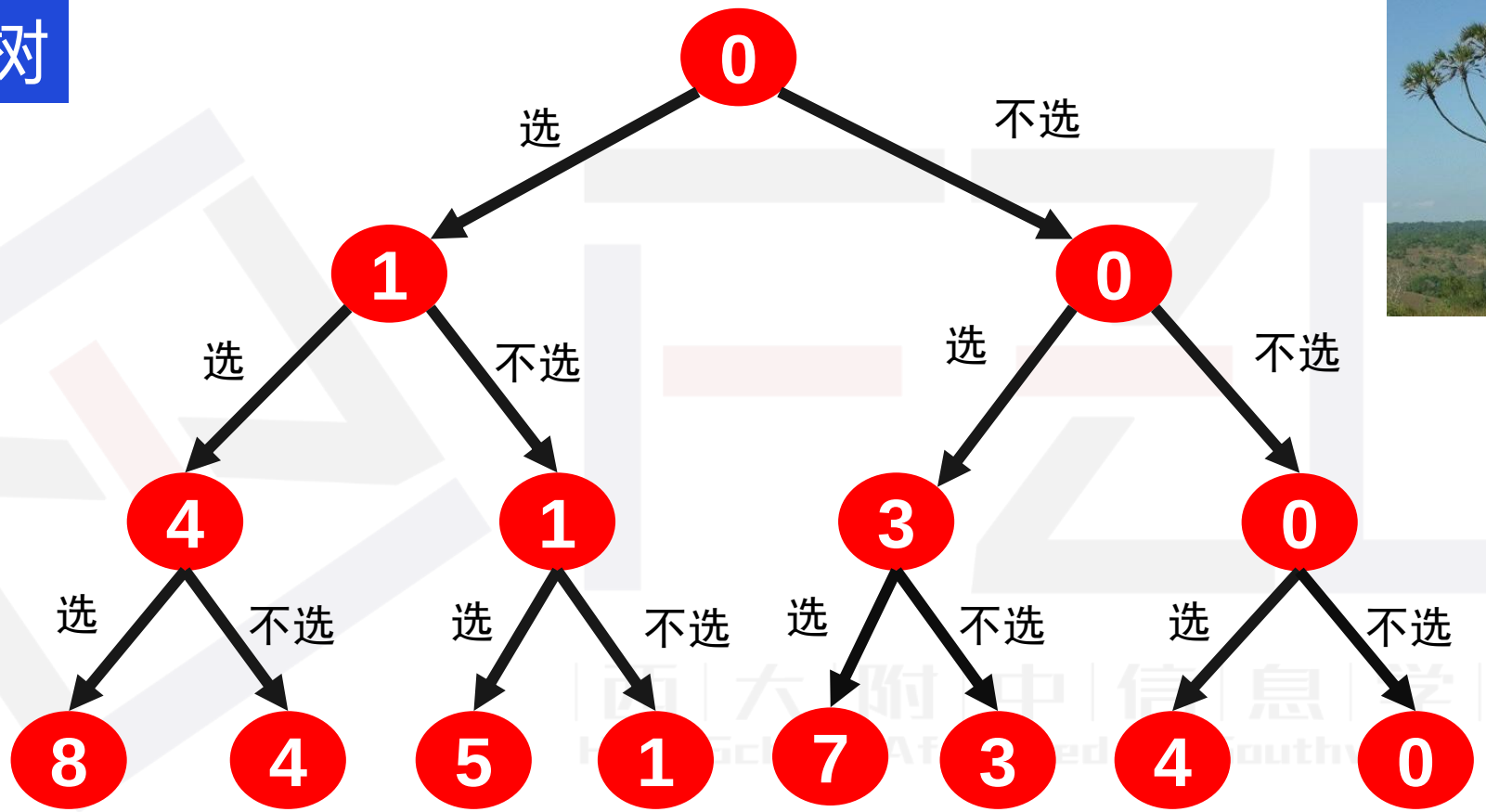
什么是深度优先搜索?



搜索时，尽可能更深的地搜索，称为深度优先搜索

搜索树





搜索的大结构包含了与它相似的子结构，很符合递归的思想，很适合利用递归的方式去解决

基本思路

在搜索树上尽可能“往下”地搜索

对于最新发现的顶点，马上进入该状态并对其进行扩展

BFS(广度优先搜索)并不是这样

当前状态不能扩展时即走到搜索树的尽头，进行回溯

整个过程反复进行直到找到目标状态或者遍历完了整棵搜索树

代码框架

框架一

```
void dfs( int  k )
{
    for(i=1;i<=可能情况数;i++)
    {
        if(满足条件）
        {
            保存结果；
            if(到达目的地） 输出解；
            else
                dfs(k+1);
        }
    }
}
```

框架二

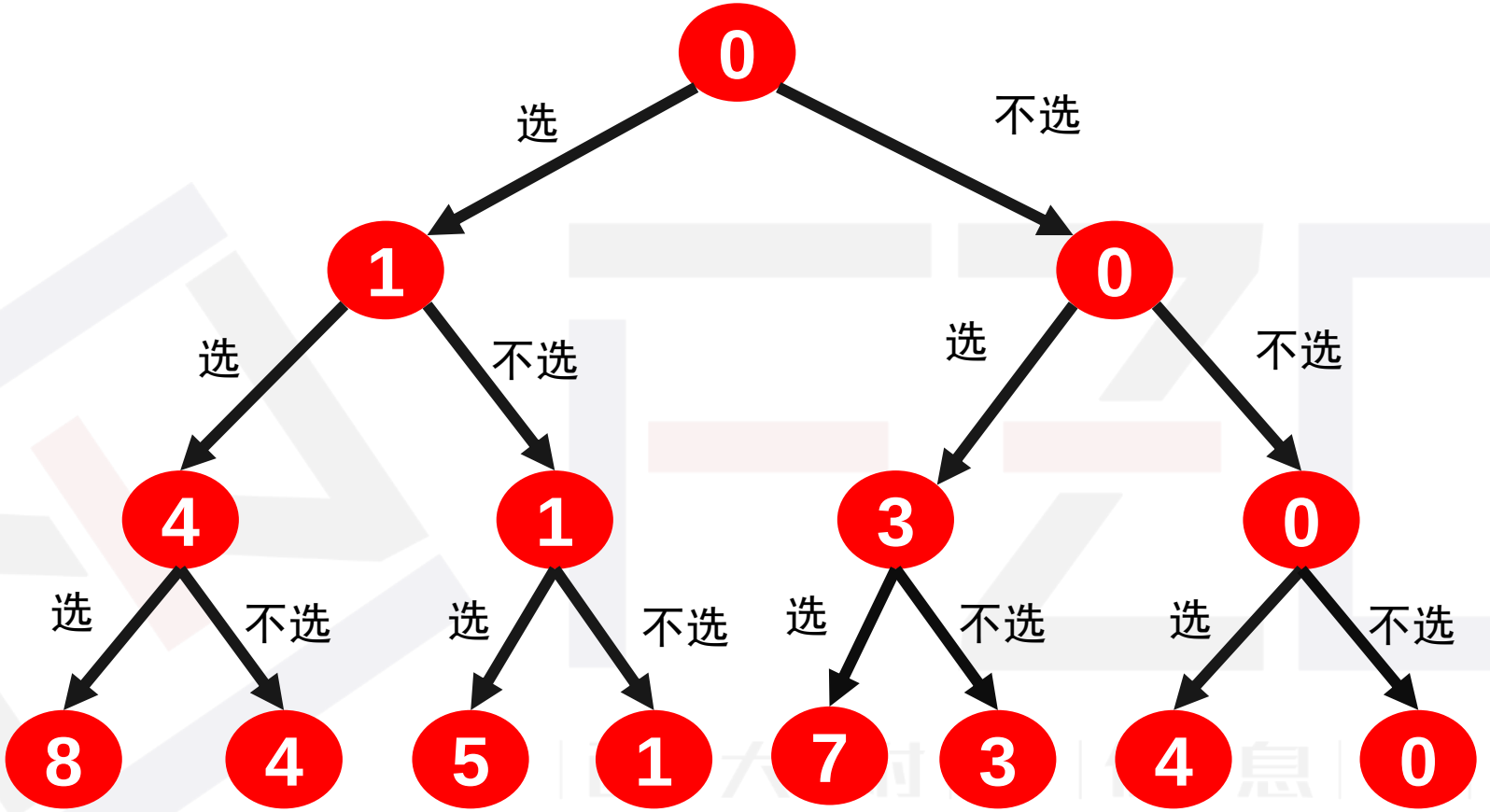
```
void dfs( int  k )
{
    if(到达目的地） 输出解；return;
    for(i=1;i<=可能情况数;i++)状态数
    {
        if(满足条件）
        {
            保存结果；
            dfs(k+1);   往下一层搜索
        }
    }
}
```

DFS解题的关键

遇到一个DFS的题目，你可能需要思考：

- 搜索可能的最大状态数
- 搜索时，上一层需要给下一层传递的信息(递归参数的传递)
- 搜索的边界条件

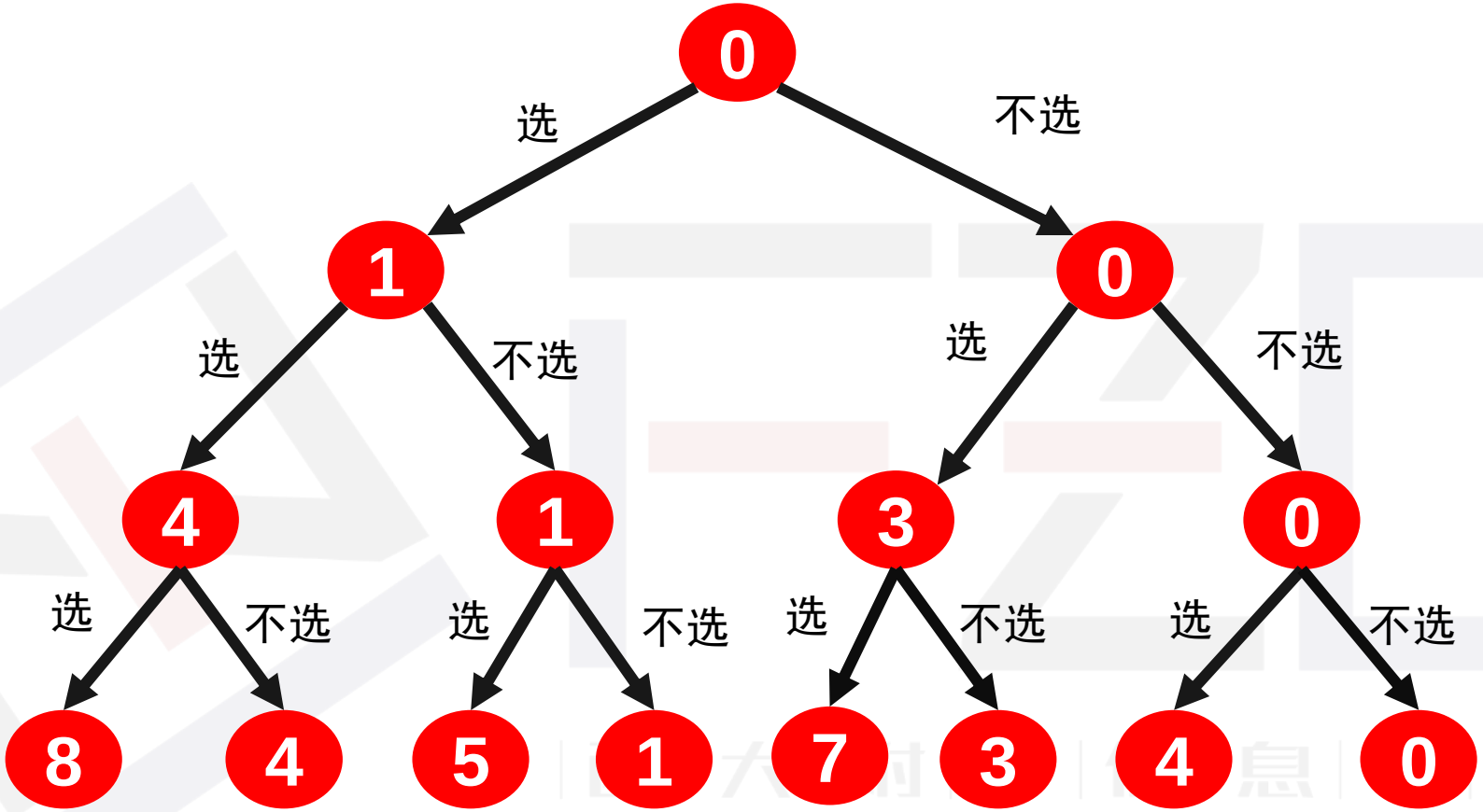
分析



Q：每个节点可能的搜索状态?

每个节点可以选/不选，2种状态

分析



Q：每个节点状态需要转移什么信息?

当前的总和sum
当前选择到了第几个物品k

dfs(k,sum)表示选择到了第k个物品，目前组成的总和为sum

数组a存储各个物品的体积
数组f标记某个体积是否搜索到

边界条件是?

当n个数已经选完，就代表搜索到了一种结果

```
if(k>n)  { f[sum]=1; return ;}
```

分析


```
void dfs( int  k )
{
    if(到达目的地） 输出解；return;
    for(i=1;i<=可能情况数;i++)
    {
        if(满足条件）
        {
            保存结果；
            dfs(k+1);
        }
    }
}
```

```
void dfs( int  k ,int sum)
{
    //边界条件
    if(k>n） {f[sum]=1;return;}
    //选
    dfs(k+1,sum+a[k]);
    //不选
    dfs(k+1,sum);
}
```

dfs结束后，统计一下f数组有多少个1即为答案

排列问题

设有n个整数的集合｛1,2,…,n｝（n<13)，从中取出任意r个数进行排列（r<n），试列出所有的排列。

输入：n、r

以由小到大的字典序输出

样例输入:

3 3

样例输出:

123

132

213

231

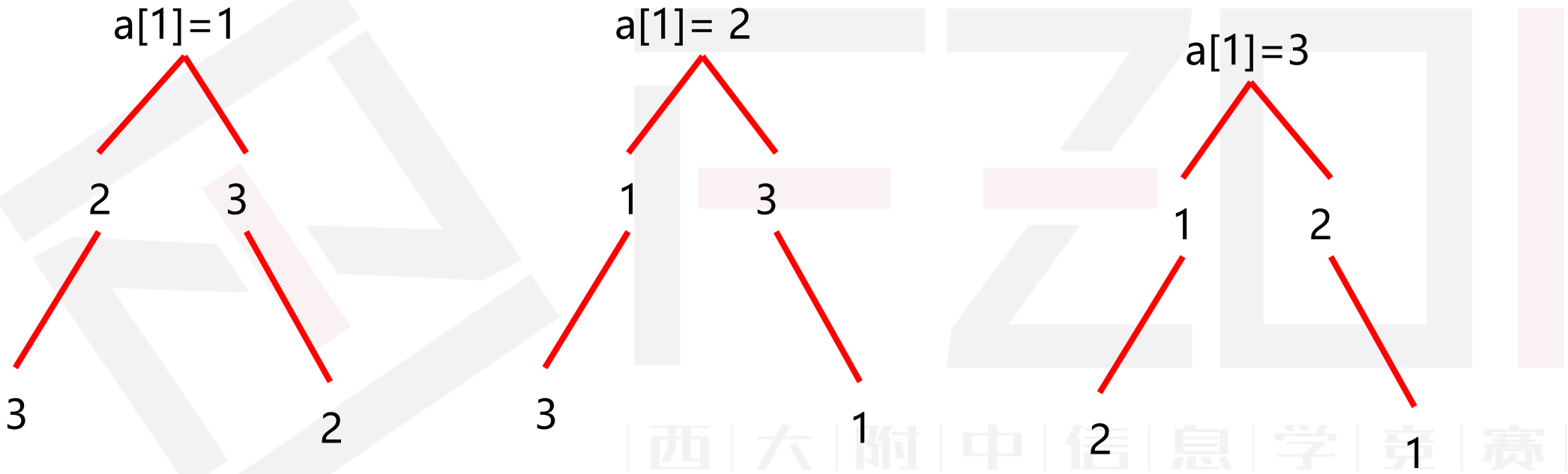
312

321

分析

以n=3，r=3为例

a[ ]存储相应的排列情况



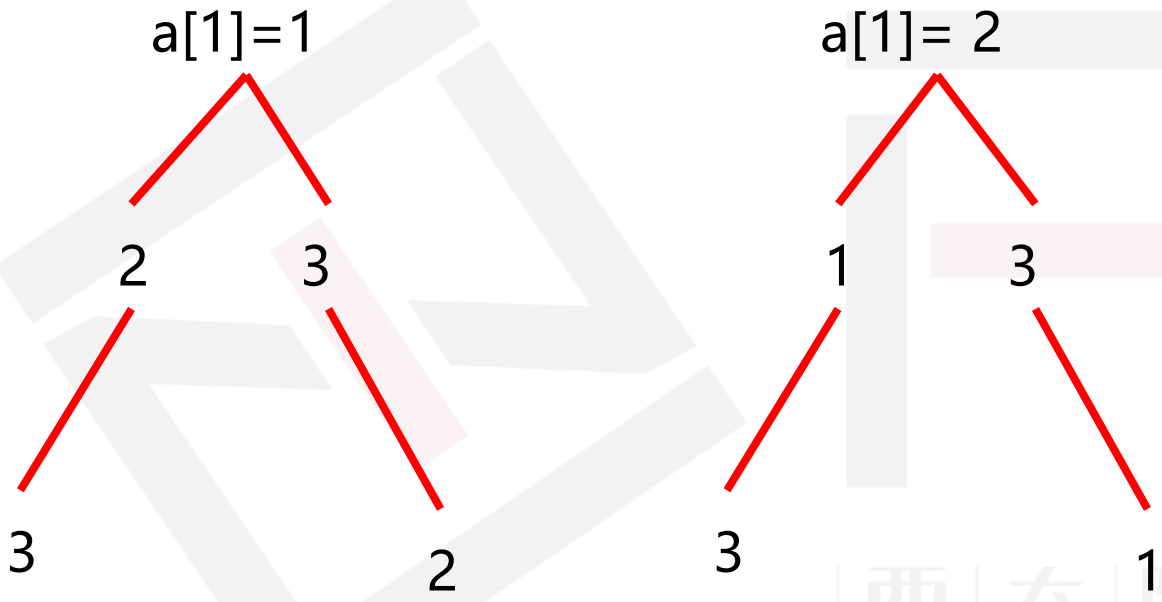
Q：每个节点可能的搜索状态?

第一个格子可以填1~n，n种

分析

以n=3，r=3为例

a[ ]存储相应的排列情况

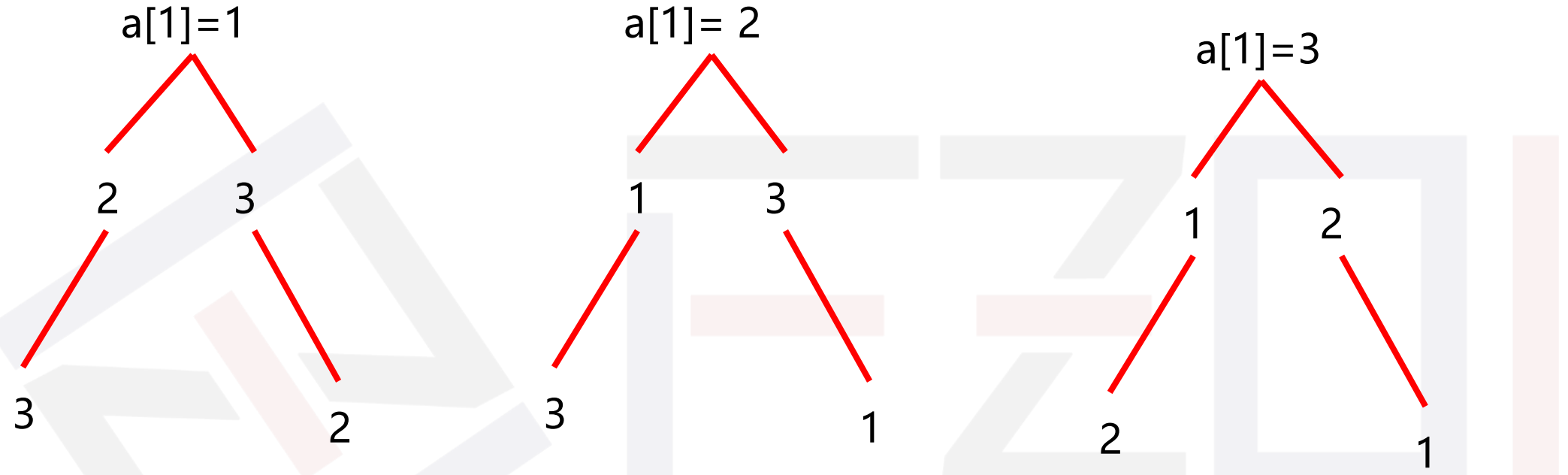


Q：每个节点状态需要转移什么信息?

整个搜索的数字范围n

当前已经选择到了第几个位置k，便于下一层确定搜索的位置

分析

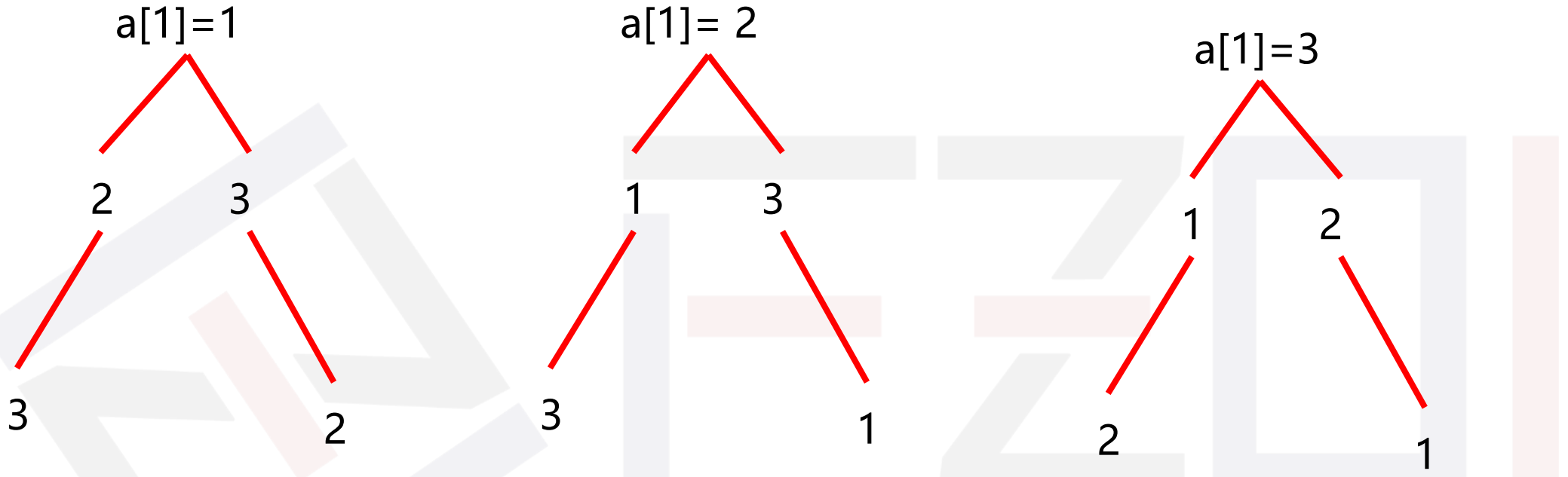


通过搜索树可以发现：每个结点位置的数都有可能是1~n之间的数，但不能与前面已有结点的数重复

Q:如何来避免这种重复选的情况?

设立一个标记数组f [ ]
f[i]=1表示i这个数已经被选择了
f[i]=0表示i还没被选择

分析

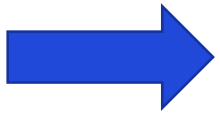


设dfs(n,k)为从n里已经选择到了第k个数进行排列

分析


```
void dfs(int n,int  k )
{
    if(边界条件） 输出解；return;
    for(i=1;i<=可能情况数;i++)
    {
        if(满足条件）
        {
            保存结果；
            dfs(k+1);
        }
    }
}
```

根据刚才的分析



```
void dfs(int n,int  k )
{
    if(边界条件） 输出解；return;
    for(i=1;i<=n;i++)
    {
        if(!f[i]）
        {
            保存结果；
            dfs(n,k+1);
        }
    }
}
```

分析


```
void dfs(int n,int  k )
{
    if(边界条件） 输出解；return;
    for(i=1;i<=n;i++)
    {
        if(!f[i]）
        {
            保存结果；
            dfs(n,k+1);
        }
    }
}
```

边界条件是什么?

已经超过了需要排列的个数r　　k>r或者k==r+1

非边界条件里做什么?

1.保存当前选择的数字

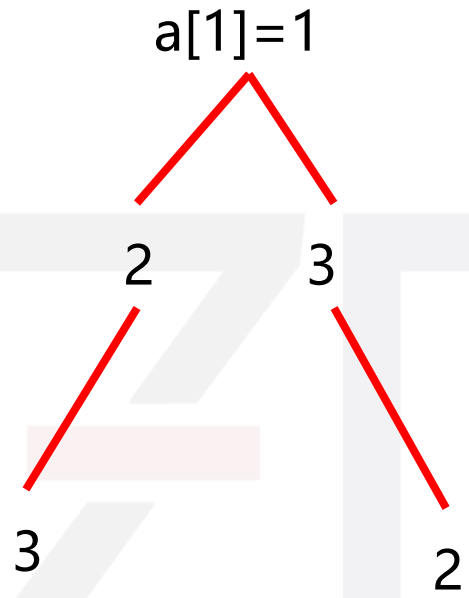
```
a[k]=i;
f[i]=1;
```

2.继续选下一个数

```
dfs(n,k+1);
```

分析


```
void dfs(int n,int  k ){
        if(k>r） {  //边界条件
              for(int i=1;i<=r;i++)  printf("%d ",a[i]);
                printf("\n");
                return;
        }
        for(i=1;i<=n;i++){
                if(!f[i]）{ //这个数没被选择
                        a[k]=i; //保存
                        f[i]=1; //标记为使用过
                        dfs(n,k+1);
                        f[i]=0   //回溯
                }
        }
}
```



当搜索到一种组合后如1 2 3，此时还有一种情况1 3 2，所以需要返回上一层结点继续搜索(回溯)

Q:回溯时需不需要做什么?

需要恢复之前选择的数为未标记状态，否则下一次搜索将无法选择

参考代码

写完代码会发现，参数n在递归时没有变化，可以定义成全局变量传入
dfs函数可改为dfs(int k)

```
void dfs(int n,int  k ){
        if(k>r)  {
           for(int i=1;i<=r;i++)  printf("%d ",a[i]);
           printf("\n");
           return;
        }
        for(i=1;i<=n;i++){
           if(!f[i]) {  //这个数没被选择
             a[k]=i;  //保存
             f[i]=1;  //标记为使用过
             dfs(n,k+1);
             f[i]=0
        }
    }
}
```

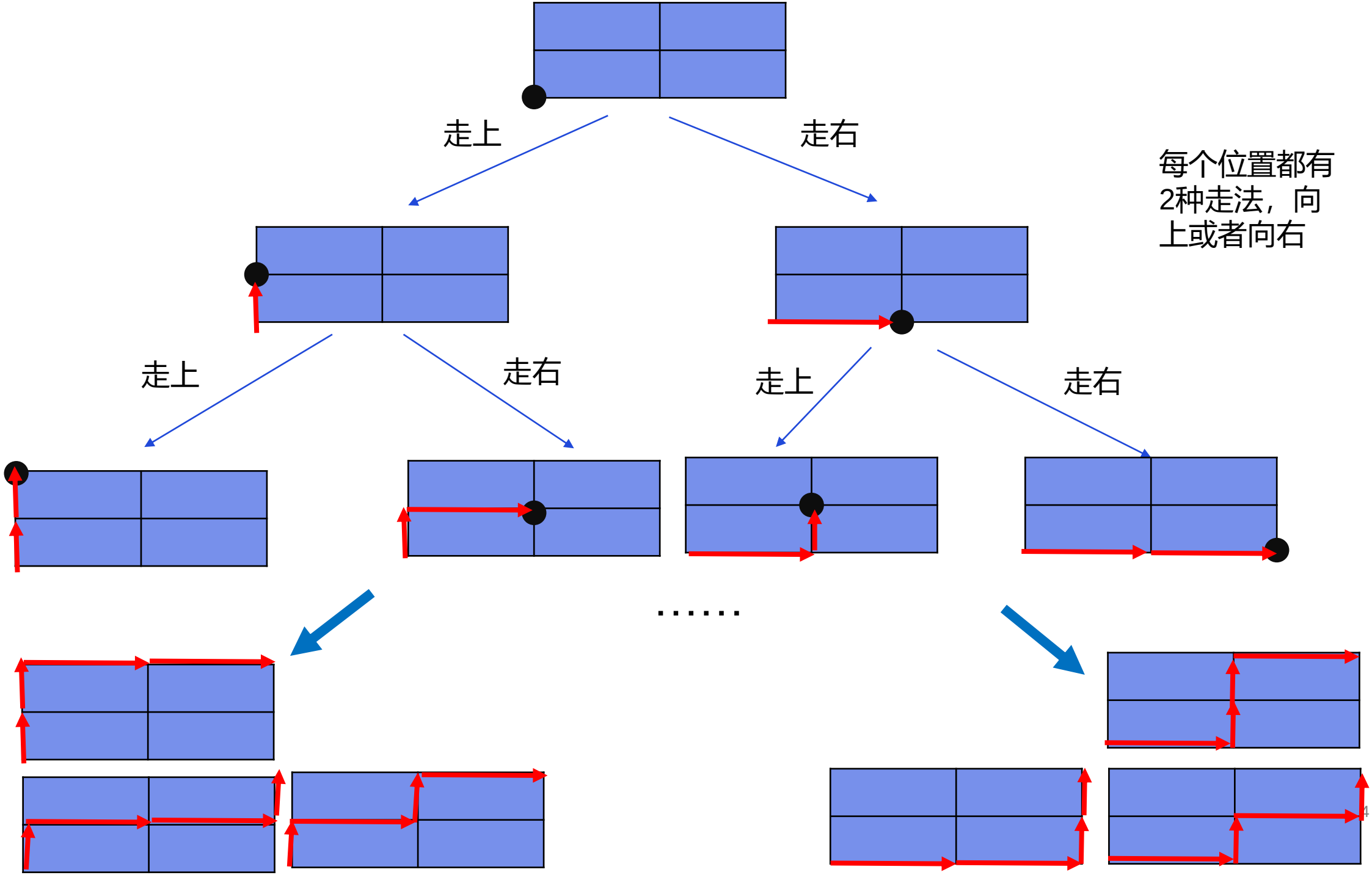
走棋盘

题目描述
在一个n*n的棋盘中(n<20)，从（0,0）点出发，只能向上或者向右走，求走到（m，n）点有多少种走法?

输入
输入终点m，n的坐标

输出
输出走法步数

样例输入
2 2
样例输出
6

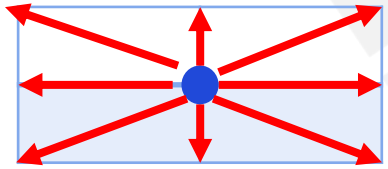

走上
走右
走上
走右
走上
走右
每个位置都有2种走法，向上或者向右
.....

当前点在(x,y)，那么2种走法：
1.向上一步，点在(x,y+1)
2.向右一步，点在(x+1,y)

设dfs(int x,int y)表示搜索从点(x,y)到终点(m,n)的路径

Q：如何更方便的表示从当前的(x,y)向上和向右走呢?

坐标增量数组：
```
int dx[2]={1,0};
int dy[2]={0,1};
```



当i=0：
```
x=x+dx[0];
y=y+dy[0]
```
向右走一步

当i=1：
```
x=x+dx[1];
y=y+dy[1];
```
向上走一步

```
for(i=0;i<=1;i++ ){
    x=x+dx[i];
    y=y+dy[i];
}
```

分析


```
void dfs(int x,int y)
{
        if(边界条件) {总数++;return;}
        for(i=0;i<=1;i++)
        {
              x=x+dx[i];
              y=y+dy[i];
            if(没有超出棋盘边界)
              dfs(x,y);
            //恢复之前状态(回溯);
            x-=dx[i];
            y-=dy[i];
        }
}
```

1.边界条件?

抵达终点就不再继续下一层的搜索了

x==m&&y==n

2.没有超出棋盘边界?

0<=x<=m&&0<=y<=n

3.递归回溯时,是否需要恢复一些状态?

减去之前的坐标增量，恢复到之前的点

参考代码


```
void dfs(int x,int y)
{
    if(x==m&&y==n)  //边界条件
    {
        cnt++;
        return;
    }
    for(int i=0;i<=1;i++)  每个结点有两种情况
    {
        x=x+dx[i];  //坐标增量
        y=y+dy[i];
        if(x<=m&&y<=n)  //没有超出边界
         dfs(x,y);
        x-=dx[i]; //回溯
        y-=dy[i];

    }
}
```

DFS算法：

为了求得问题的解，不断向更深的层次进行探索，在探索的过程中，一旦发现原来的选择是错误的，就退回一步(回溯)重新选择，继续向前探索，如此反复进行，直至得到解或者证明无解。

在递归回溯时需要思考是否需要恢复一些状态，以避免对其他搜索情况的影响

DFS解决的常见问题

- 子集和问题
- 排列组合问题
- 走迷宫问题

DFS也是其他进阶算法的基础

Thanks

For Your Watching