状态空间搜索

肖锋

04计算机系

搜索算法

- 枚举
- 搜索
  - 广度优先搜索
  - 深度优先搜索
  - 双向广搜、迭代加深
- 影响搜索效率的因素
- POJ题目讨论

枚举

- 逐一判断所有可能的方案是否是问题的解
- 例1：求出A-I这九个字母对应的数字（1-9），使得下式成立（一一对应）

$$\begin{array}{r} ABCD \\ \times \quad E \\ \hline FGHI \end{array}$$

枚举

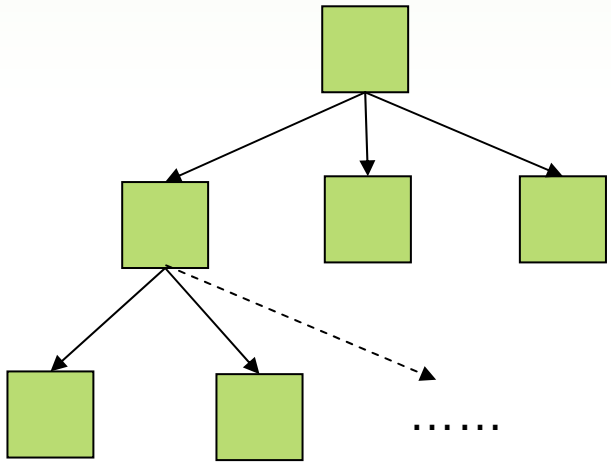
$$\begin{array}{r} ABCD \\ \times \quad E \\ \hline FGHI \end{array}$$

- 解法：
- 枚举ABCDE的值，计算乘积，判断是否符合要求。

- 例2：奇怪的问题
- 1）第一个答案是b的问题是哪一个？c
- （a）2 （b）3 （c）5 （d）4
- 2）恰好有两个连续问题是一样的，它们是：d
- （a）1,2 （b）2,3 （c）4,5 （d）3,4
- 3）本问题答案和哪一个问题的答案相同？a
- （a）4 （b）3 （c）2（d）1
- 4）答案是a的问题的个数是：a
- （a）2（b）1 （c）0 （d）3 （e）4
- 5）本问题的答案和哪一个问题的答案相同？b
- （a）1 （b）无 （c）4 （d）2,3

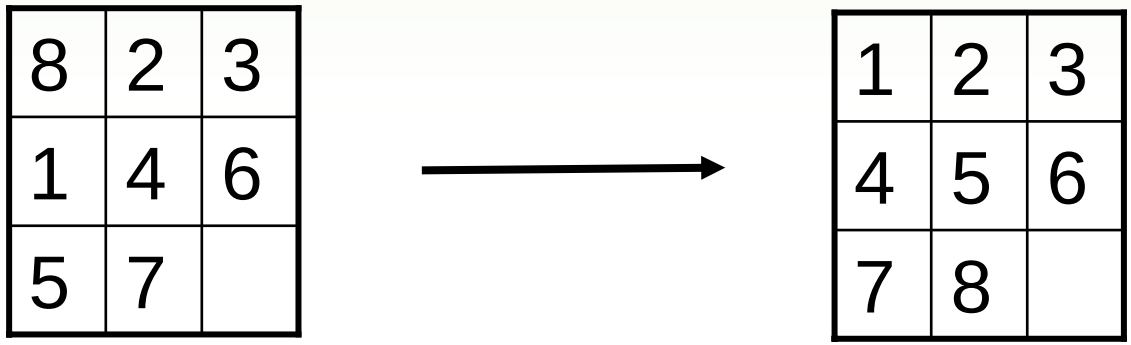
搜索

- 搜索：高级枚举
  - 有顺序有策略地枚举状态空间中的结点，寻找问题的解



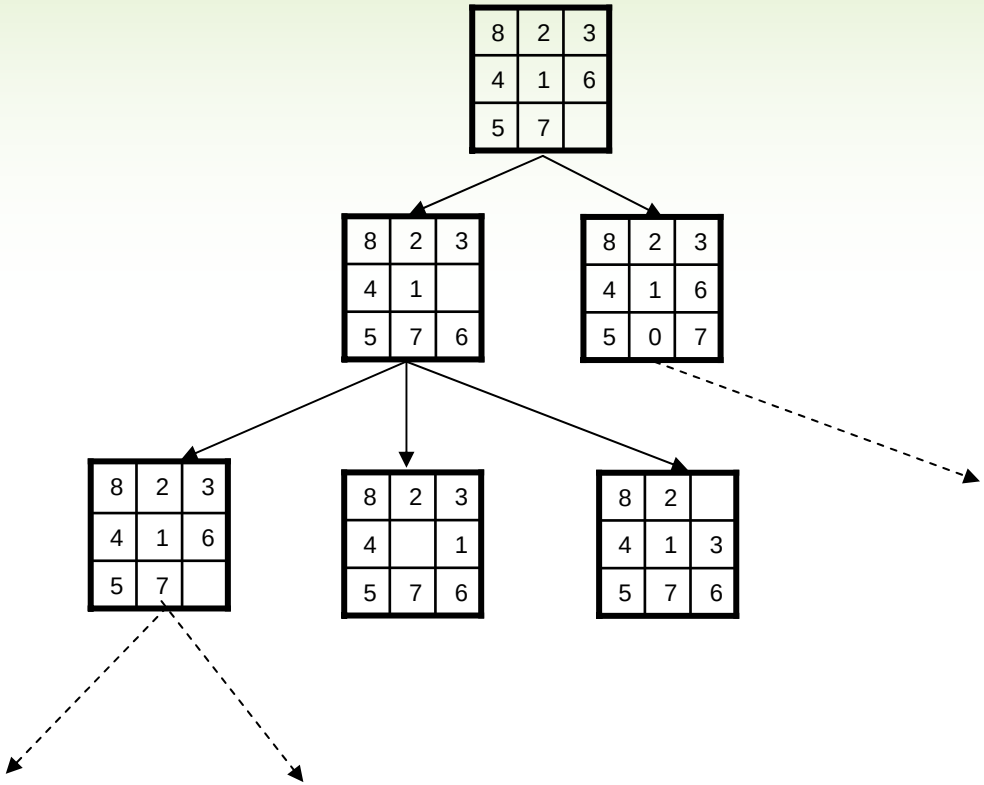
搜索

- POJ1077八数码问题



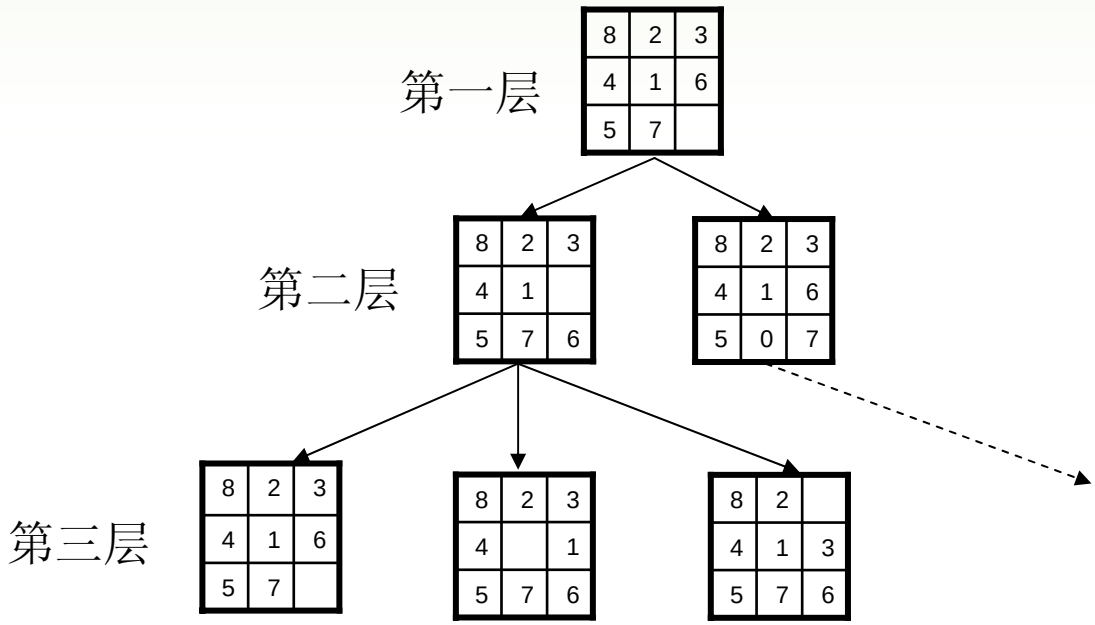
搜索

- 状态空间



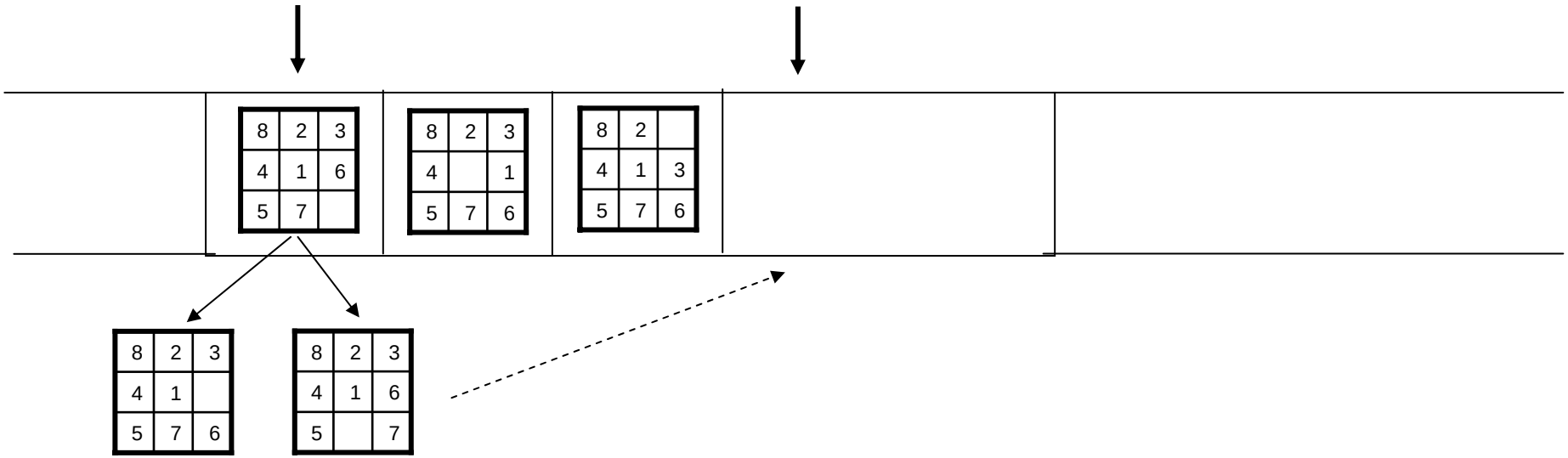
广度优先搜索

- 广度优先搜索
  - 优先扩展浅层结点，逐渐深入



广度优先搜索

- 广度优先搜索
  - 用队列保存待扩展的结点，从队首队取出结点，扩展出的新结点放入队尾，直到找到目标结点（问题的解）



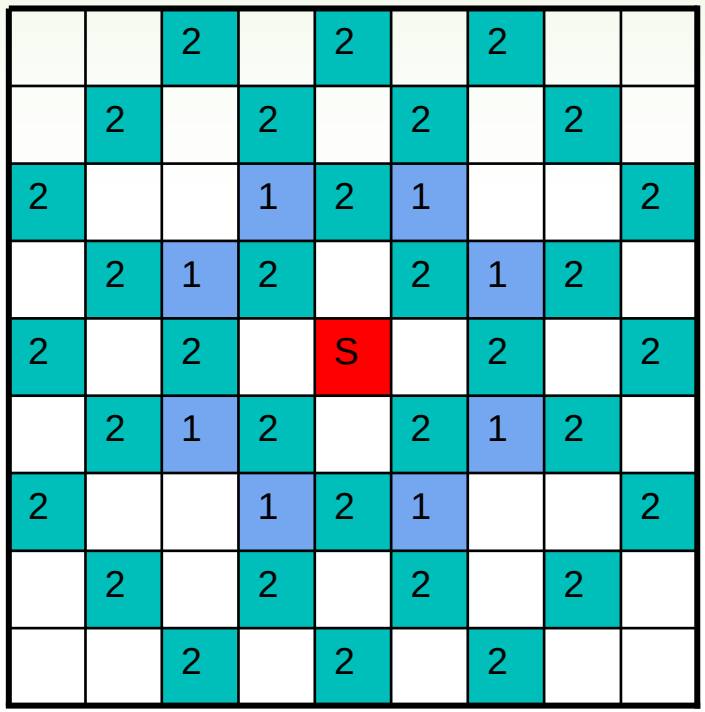
广度优先搜索

- 广度优先搜索的代码框架

```
BFS()
{
        初始化队列
        while(队列不为空且未找到目标结点)
        {
                取队首结点扩展，并将扩展出的结点放入队尾
        }
}
```

广度优先搜索

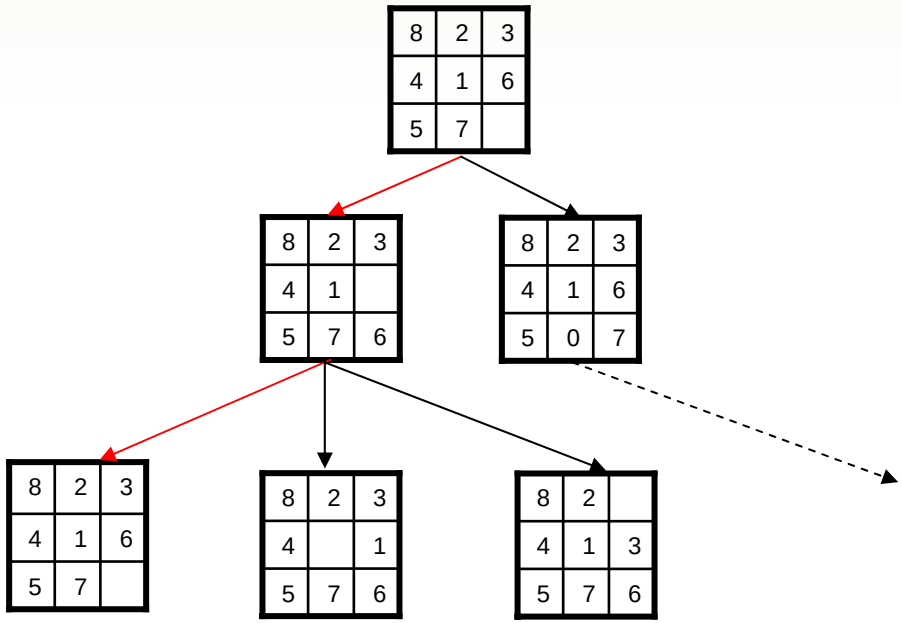
- 例：POJ1915-Knight Moves



- 每个状态结点用两个数（横纵坐标）表示
- 通过一个二维表存储已到达的位置进行判重

深度优先搜索

- 深度优先搜索
  - 优先深入遍历靠前的结点



深度优先搜索

- 深度优先搜索
  - 可以用栈实现，在栈中保存从起始结点到当前结点的路径上的所有结点
  - 一般用递归实现

深度优先搜索

- 深度优先搜索的非递归框架

```
DFS()
{
        初始化栈
        while(栈不为空 且 未找到目标结点)
        {
                取栈顶结点扩展，扩展出的结点放回栈顶
        }
        ……
}
```

深度优先搜索

- 深度优先搜索的递归框架

```
DFS(type cur_node, int depth)
{
        for(cur_node的每个子结点child_node)
        {
                DFS(child_node, depth+1)
        }
}
```

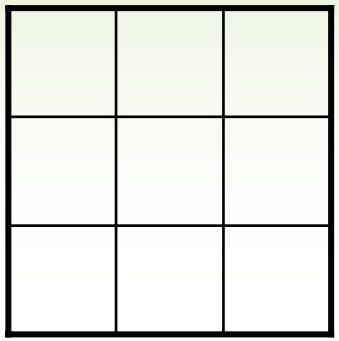
深度优先搜索

- 深度优先搜索的递归框架
  - 在深度优先搜索中，状态空间的图结构并不一定需要显式的存下来。

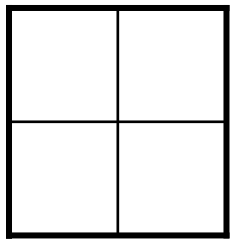
```
type node;
void DFS(int depth){
        for(node的每一个可行变化){
                改变node
                DFS(depth + 1)
                恢复node
        }
}
```

深度优先搜索

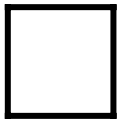
- 例：POJ1020



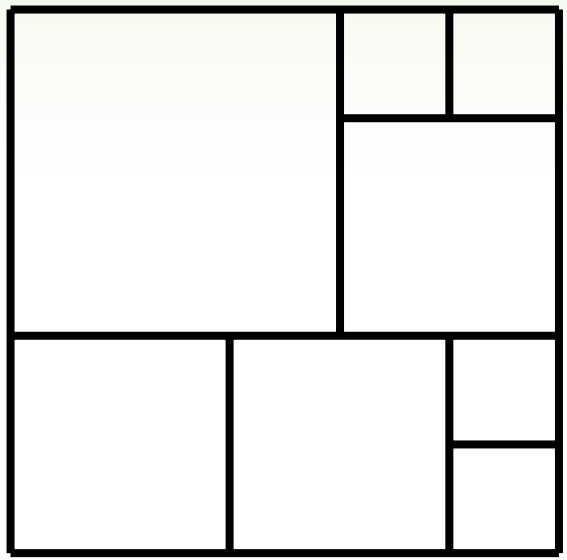
× 1



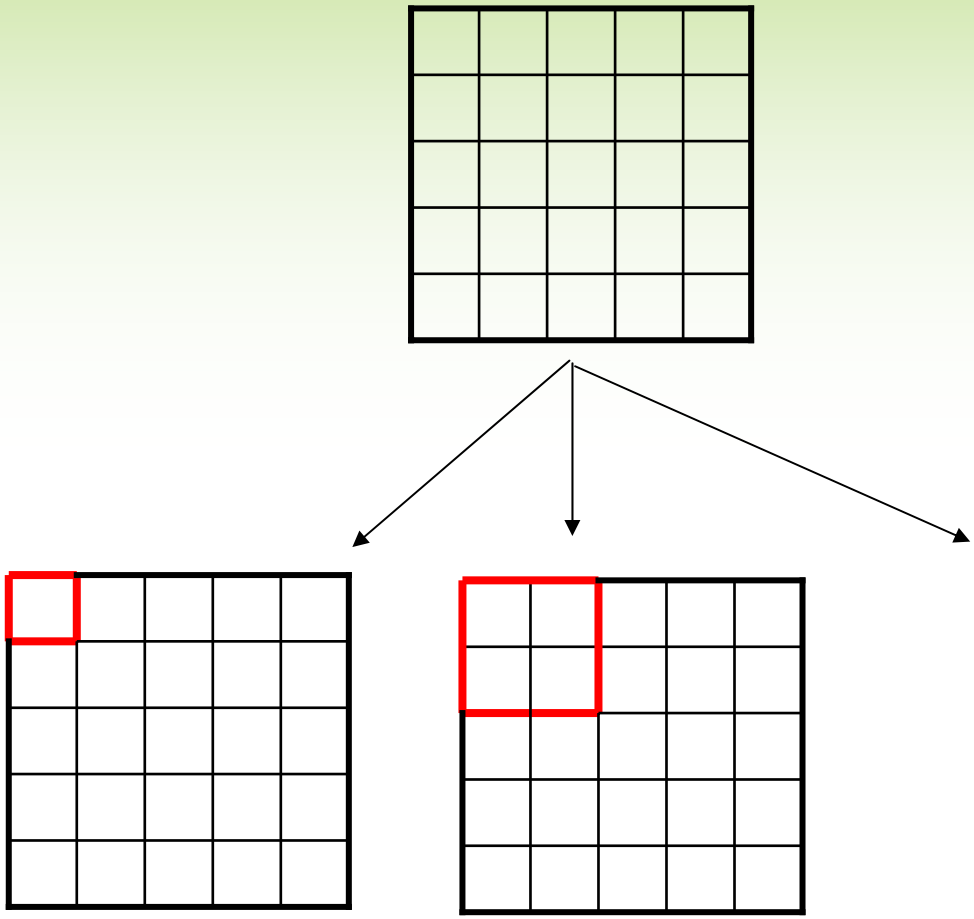
× 3



× 4



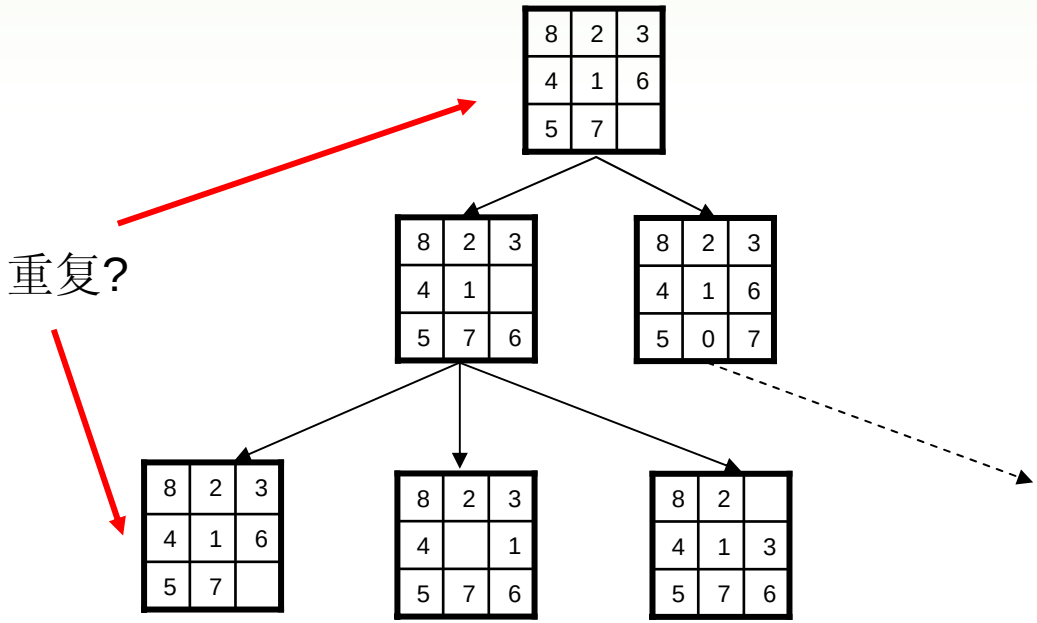
深度优先搜索



```
int map[5][5]; /* 表格状态 */
void DFS(int depth)
{
    if(已经放满) { 程序结束 }
    for(每个可放置的方案)
    {
        在map上放一个方块
        DFS(depth + 1)
        取下刚放置的方块
    }
}
```

判重

- 判重
  - 新扩展出的结点可能和以前扩展出的结点相同
  - 如何判重？



判重

- 需要考虑的问题
  - 状态数目巨大，如何存储？
  - 怎样才能较快的找到重复结点？

判重

- 合理编码，减小存储代价
  - 不同的编码方式所需要的存储空间会有较大差别

8	2	3
4	1	6
5	7	

- 方案一：每个结点用9个整数表示
  - 每个状态需要9字节
- 方案二：为结点编号
  - 把每个结点都看一个排列，以此排列在全部排列中的位置作为其编号
  - 排列总数：9!=362880
  - 只需要一个整数(4字节)即可存下一个结点

判重

- 使用合适的数据结构
  - 使用的数据结构与状态的表示有关
  - 一般使用散列表

8	2	3
4	1	6
5	7	

➢方案一：每个结点用9个整数表示

➢需要使用散列表

➢需要设计一个好的散列函数

➢方案二：为结点编号

➢只需要一个大数组即可

判重

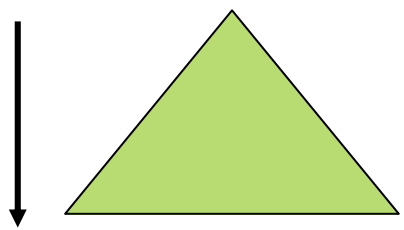
- 时间与空间的权衡
  - 对于状态数较小的问题，可以用最直接的方式编码以空间换时间
  - 对于状态数太大的问题，需要利用好的编码方法以时间换空间
  - 具体问题具体分析

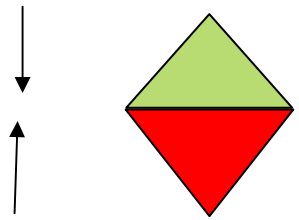
广搜与深搜的比较

- 广搜一般用于状态表示比较简单、求最优策略的问题
  - 需要保存所有扩展出的状态，占用的空间大
  - 每次扩展出结点时所走过的路径均是最短路
- 深搜几乎可以用于任何问题
  - 只需要保存从起始状态到当前状态路径上的结点
- 根据题目要求凭借自己的经验和对两个搜索的熟练程度做出选择

双向广搜

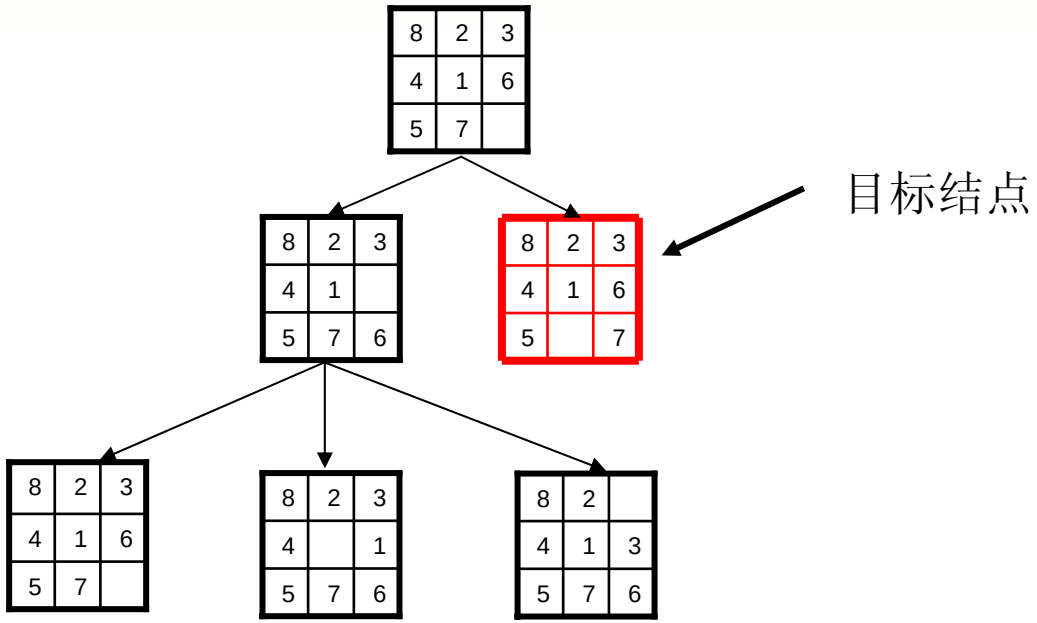
- 双向广搜
  - 广搜时每加深一层状态数成倍增长，呈金字塔形
  - 从起始结点和目标结点同时扩展可以减少扩展的状态数





迭代加深的搜索

- 迭代加深的深度优先搜索
  - 目标结点可能在较浅的层，而深度优先搜索可能扩展出大量深层次结点然后才找到解



迭代加深的搜索

- 优点
  - 使得深度优先搜索可用来求得最优解
  - 当目标结点的深度较浅时，可以比较快的找到
  - 比广搜节省空间
- 缺点
  - 搜索有重复

影响搜索效率的因素

- 影响搜索效率的因素
  - 搜索对象
  - 搜索顺序
  - 剪枝

影响搜索效率的因素

- 搜索对象
  - 直接决定状态空间的大小、形状

例：超长数字串

数字串S：12345678910111213141516171819202122……，任意给一个有限长度的数字串S1，它在S中出现了无穷多次，求编程求它在S中最先出现的位置。

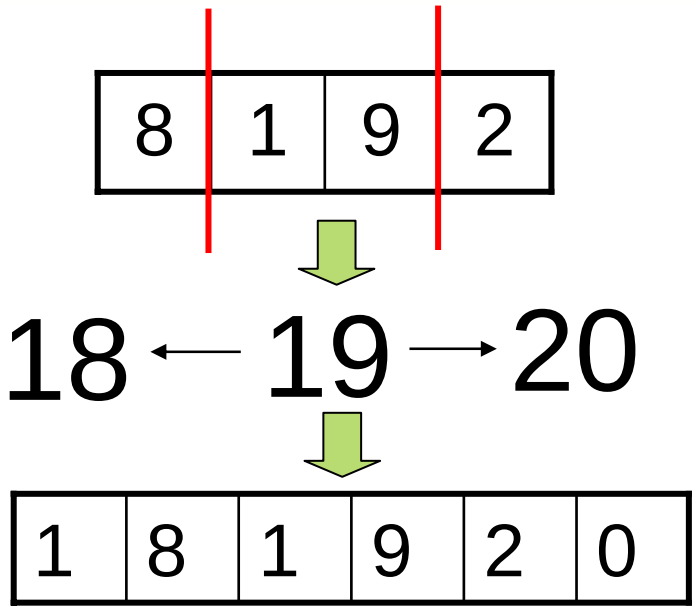
➢方案一：从1开始逐一枚举数字串S，直至找到与S1匹配的子串出现

影响搜索效率的因素

- 搜索对象

例：超长数字串

➢8192出现在……17181920……这个位置，在8192中19是一个完整的数字



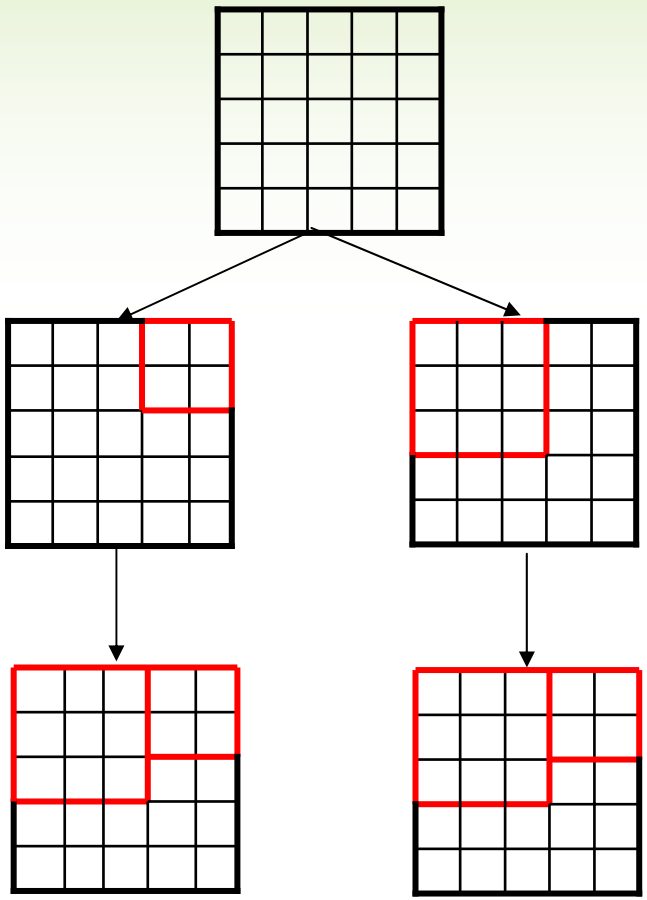
方案二：枚举原串的切片，以之为整数，与其前后的数组成一个串，判断是否与原串匹配

影响搜索效率的因素

- 搜索顺序
  - 合适的搜索顺序有时可以减小扩展出的结点的重复度
  - 好的搜索顺序可以做到尽快的逼近最优解，提高搜索效率

影响搜索效率的因素

- 搜索顺序

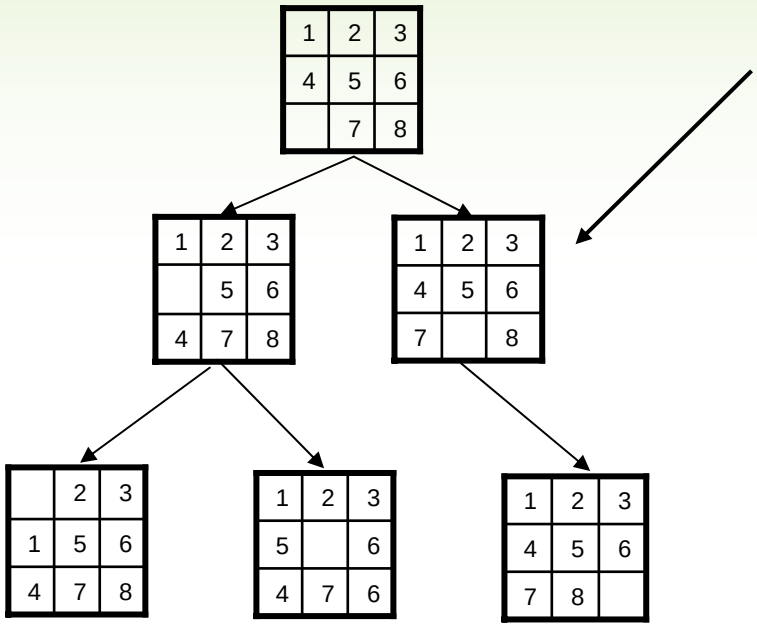


无序搜索会导致大量重复

如果规定当前放置的方块必须覆盖所有未被覆盖的格子中最左上的那一个，则此例中的重复不会出现

影响搜索效率的因素

- 搜索顺序



这个结点更接近目标结点，为什么不先搜索它呢？

在搜索过程中，可以选择“看起来”离目标更近的结点优先搜索，从而更快的找到目标结点。

影响搜索效率的因素

- 剪枝
  - 在大多数情况下，不加剪枝的搜索在时限内无法出解。
  - 超时的程序加上剪枝后可能会瞬间出解。
- 计算剪枝条件所用的代价不能多于剪枝所带来的好处

POJ题目讨论

- 1190 生日蛋糕
- 1167 The Buses
- 1184 聪明的打字员
- 2458 Rigging the Bovine Election
- 1168 The Circle
- 1334 Two Mountaineers
- 2310 Cubic Tick-Tack-Toe
- 3008 Hexerpents of Hexwamp

POJ题目讨论

- 1167 The Buses
- 搜索对象与剪枝
  - 方案一：依次确实每辆车所属的线路，线路数逐渐增加，分枝增加，剪枝可去掉大片枝叶，可以取得较好效果
  - 方案二：每次确定一条线路，剩余车数减少，分枝减少，剪枝效果不如方案一

POJ题目讨论

- 1184 聪明的打字员
- 改变搜索对象-状态分离
  - 把移动光标的操作Left ， Rigth， Swap0，Swap1和改变数值的操作Up，Down分离开
  - 先只进行移动光标的操作，记录得到的数的排列以及光标曾经到过的位置，状态数目6!*2^6=46080
  - 对上一步得到的每一个状态判断通过改变光标经过的位置上的数字能否得到目标状态
  - 取最优即可

POJ题目讨论

- 2458 Rigging the Bovine Election
- 搜索顺序
  - 枚举时只选择与已经选出的方格相邻的
  - 假设A是与已选出的方格相邻的一个方格，那么搜索的分支有两个：一是把A选入，二是把A标记为不可选结点，以后永远不要选入
  - 与已选方格相邻的方格可以用栈保存，每次选栈顶点决定选入还是永久抛弃

POJ题目讨论

- 1168 The Circle
- 搜索顺序
  - 从m至i逐一枚举组成这个数的片断，然后根据这个和值再去确实这个片断里的每个数值

POJ题目讨论

- 1334 Two Mountaineers
- 状态空间
  - 广度优先搜索
  - 只需保存两个人至少有一个人在线段顶点时的状态，都在线段中间的状态不需要保存

POJ题目讨论

- 2310 Cubic Tick-Tack-Toe
- 3008 Hexerpents of Hexwamp
- 解题关键
  - 不要怕麻烦。。。

总结

- 搜索算法十分强大
  - 可以用在一切你想不出其他算法的时候
- 搜索算法十分灵活
  - 千变万化的搜索对象与搜索顺序，更加千变万化的剪枝条件
- 想要比较熟练的运用搜索算法就需要大量练习积累丰富的经验

谢谢