搜索专题

谭松松

Outline

- 引子:什么是搜索
- 两个搜索框架DFS/BFS
- 简单的经典例题
- 搜索的优化-高级的例题
- 一个去年专题例题的代码

8数码问题

- 在3*3的组成的九宫格棋盘上摆了八个牌，每一个都刻有1-8中的某一个数码

8数码问题

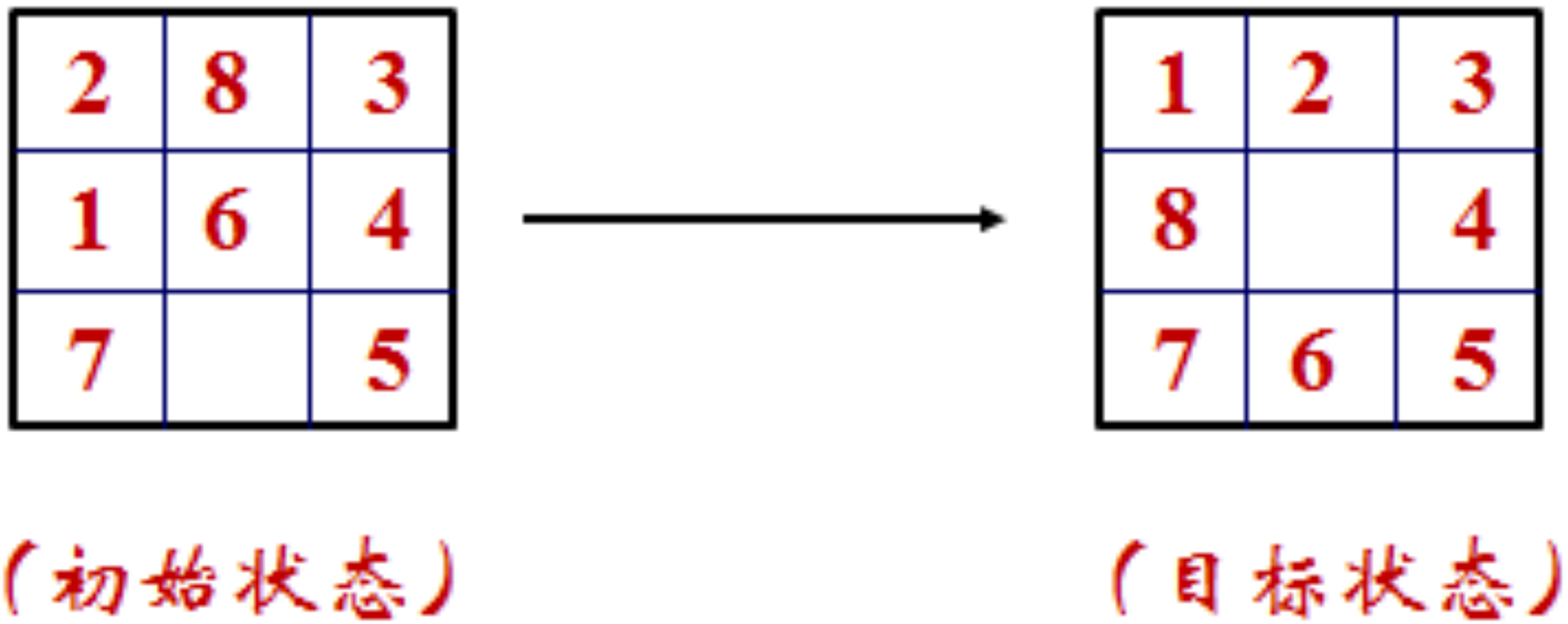
- 在3*3的组成的九宫格棋盘上摆了八个牌，每一个都刻有1-8中的某一个数码
- 棋盘中有一个空格，允许其周围的某一个牌向空格移动，通过移动空格可以改变整个盘面

8数码问题

- 在3*3的组成的九宫格棋盘上摆了八个牌，每一个都刻有1-8中的某一个数码
- 棋盘中有一个空格，允许其周围的某一个牌向空格移动，通过移动空格可以改变整个盘面
- 问题：给定一个初始牌面，问是否能够通过一系列操作达到另一个盘面

8数码问题

八数码难题（8-puzzle problem）



（初始状态）　　（目标状态）

简单？困难？

- 简单
  - BFS/DFS/随机化/启发式
  - 描述简单，思想简单

- 困难
  - 怎么搜（好的状态表示）、什么顺序搜怎么搜的快（剪枝优化）
  - 敢不敢搜?

搜索问题

- 在一个状态空间中寻找目标
  - 搜索什么（目标）
  - 在哪里搜索（状态空间）
    - 状态转移（生成规则）
- 如何扩展状态空间
- 盲目搜索/启发式搜索
  - 是否在搜索中加入信息指导搜索的方向

搜索问题

广义的图的遍历问题

程序结构

- 状态数据库

  如何表述一个状态，存储一个状态，如何判断重复状态

  在一些情况下，若按照某种顺序搜索，不会产生重复状态

- 生成规则

  在什么条件下，从老的状态产生新的状态

- 控制策略

  按照什么路线调用生成规则生成新状态

控制策略

深度优先搜索（DFS）

广度优先搜索（BFS）

深度优先搜索（DFS）

- 也称回溯搜索：能进则进，进不了则换，换不成则退
- 优点：对空间消耗小，是与深度呈线性相关的
- 问题：可能搜索到错误的路径上，在无限的空间中陷入无限的搜索
- 问题：最初搜到的结果不一定是最近最优解
- 问题：容易暴栈

深度优先搜索（DFS）

```
void dfs(当前状态){
    if(达到目标状态) {
...
        return;
    }
    for(遍历下一层状态){
        选择下一层状态
        dfs(下一层状态);
        恢复下一层状态
    }
}//非常简单的代码
```

深度优先搜索（DFS）

dfs走方格图

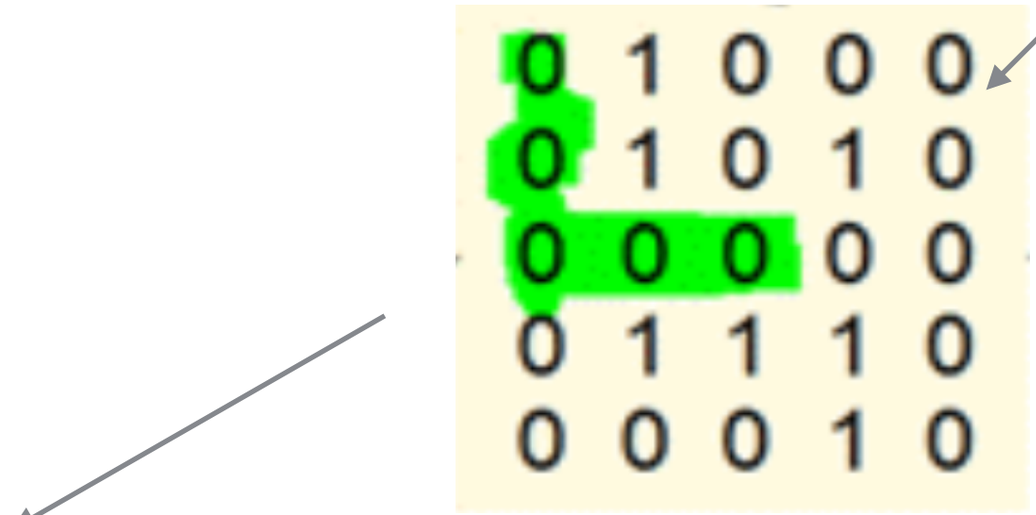
```
int map[size][size];
int vis[size][size];
int dx[4]={1,-1,0,0};int dy[4]={0,0,-1,1};
bool can_move(int x,int y)
{
    return x>=0&&x<n&&y>=0&&y<m&&vis[x][y]==0;
}
int ans=999999999;
void search(int x,int y,int step)
{
    if(x==endx&&y==endy)
    {
        ans=min(ans,step);
        return ;
    }
    vis[x][y]=1;
    for(int i=0;i<4;i++)
    {
        int nx=x+dx[i];
        int ny=y+dy[i];
        if(can_move(nx,ny))
            search(nx,ny,step+1);
    }
    vis[x][y]=0;
}
```

0	1	0	0	0
0	1	0	1	0
0	0	0	0	0
0	1	1	1	0
0	0	0	1	0

0	1	0	0	0
0	1	0	1	0
0	0	0	0	0
0	1	1	1	0
0	0	0	1	0

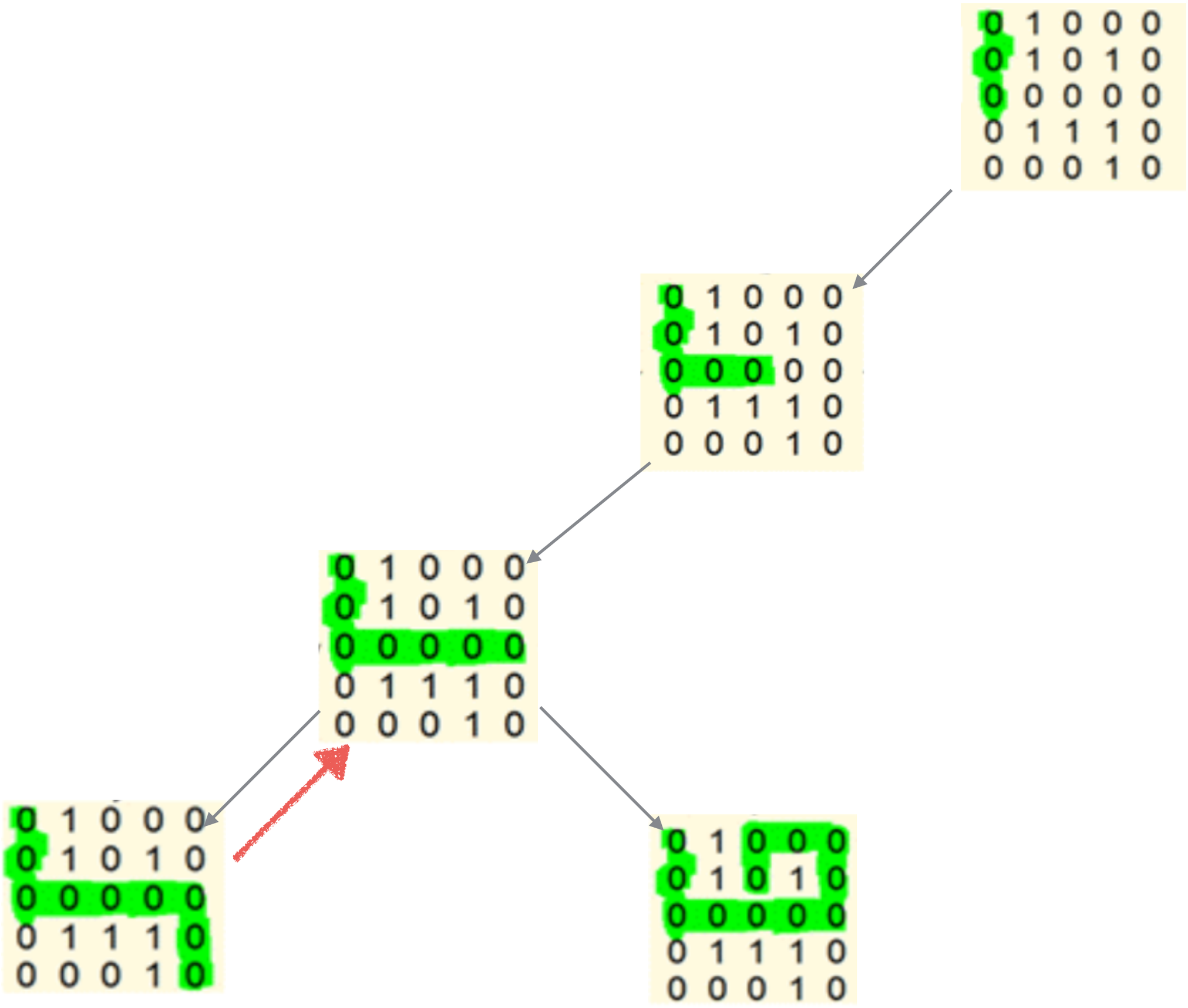
0	1	0	0	0
0	1	0	1	0
0	0	0	0	0
0	1	1	1	0
0	0	0	1	0

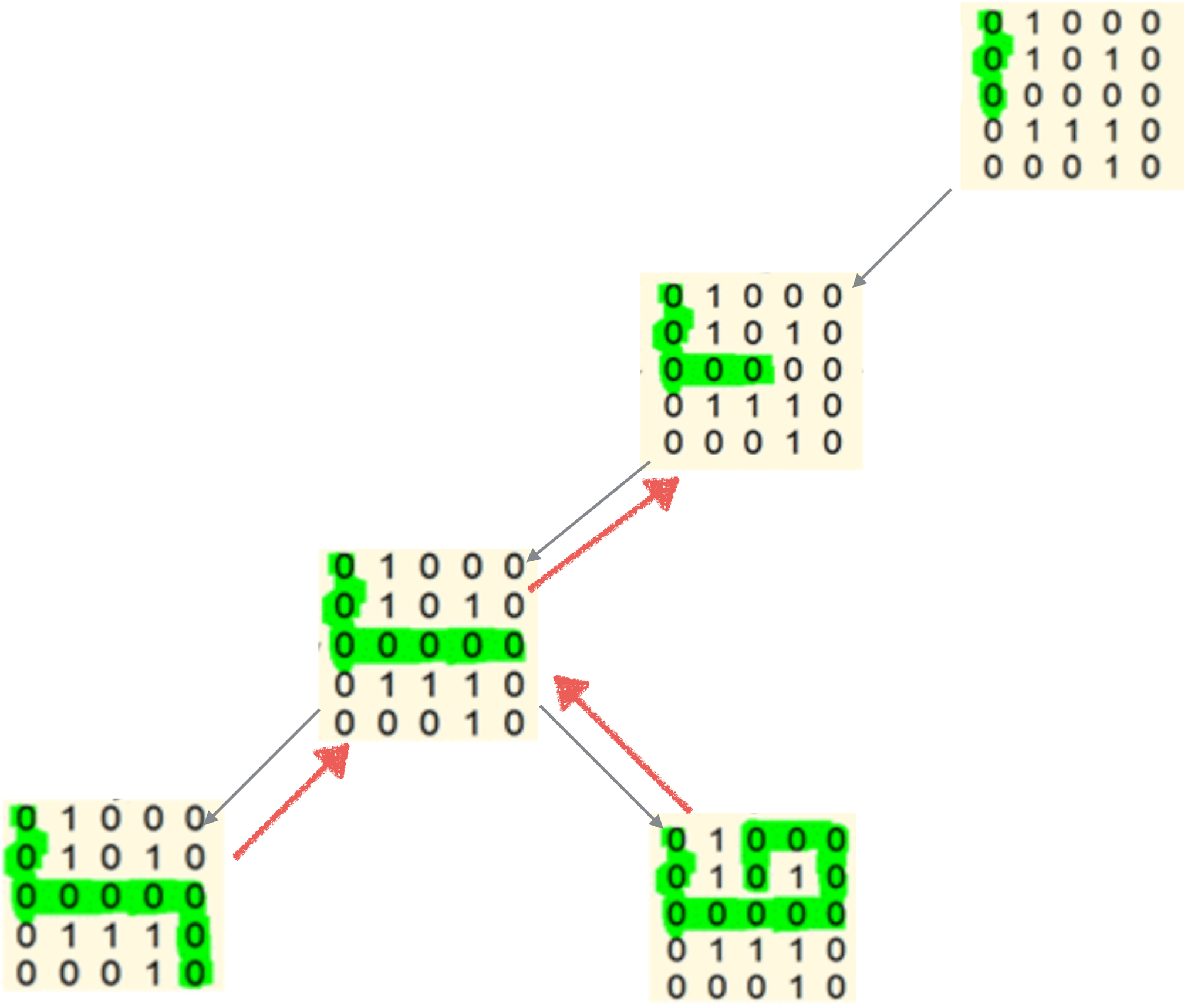
0	1	0	0	0
0	1	0	1	0
0	0	0	0	0
0	1	1	1	0
0	0	0	1	0

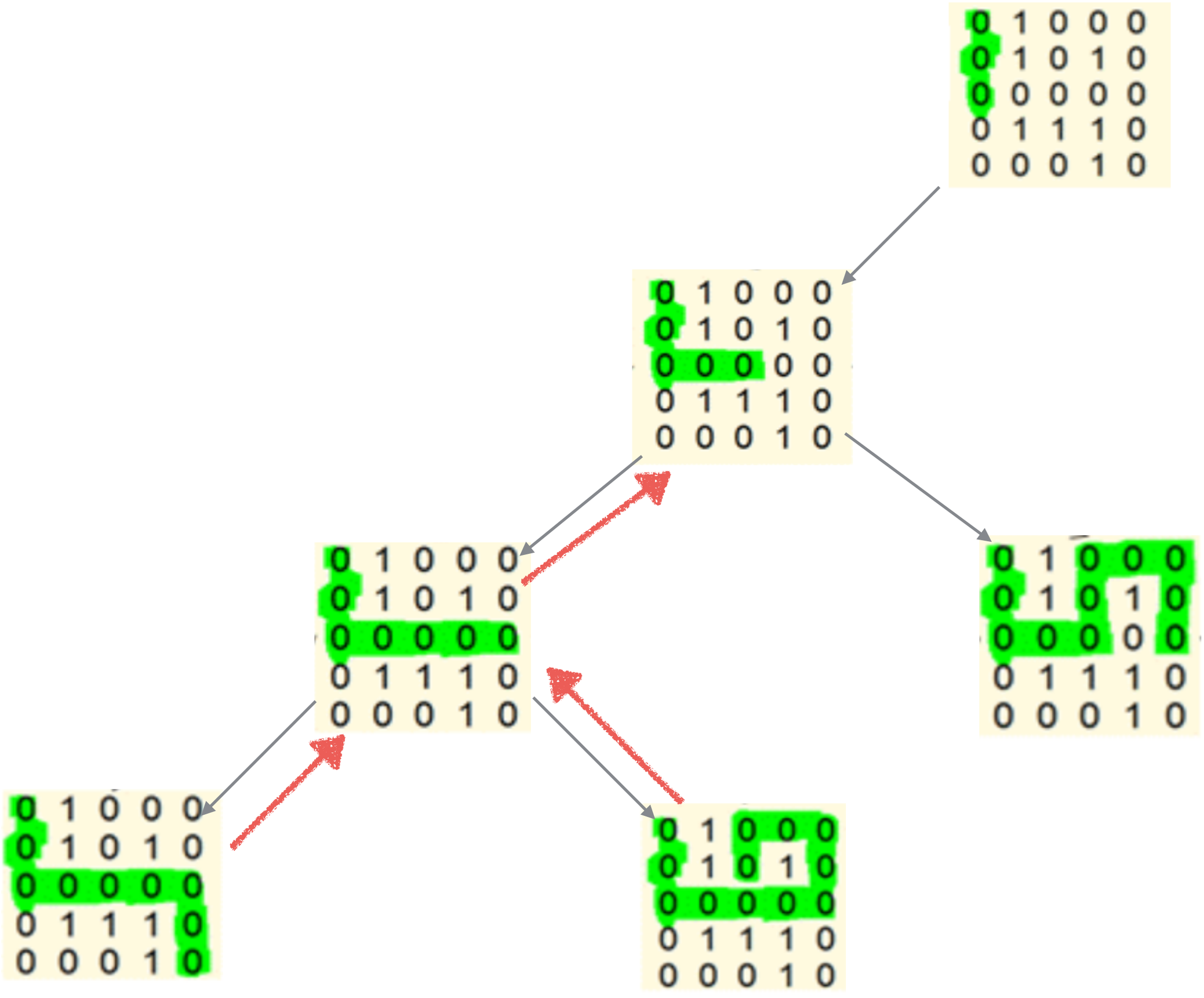


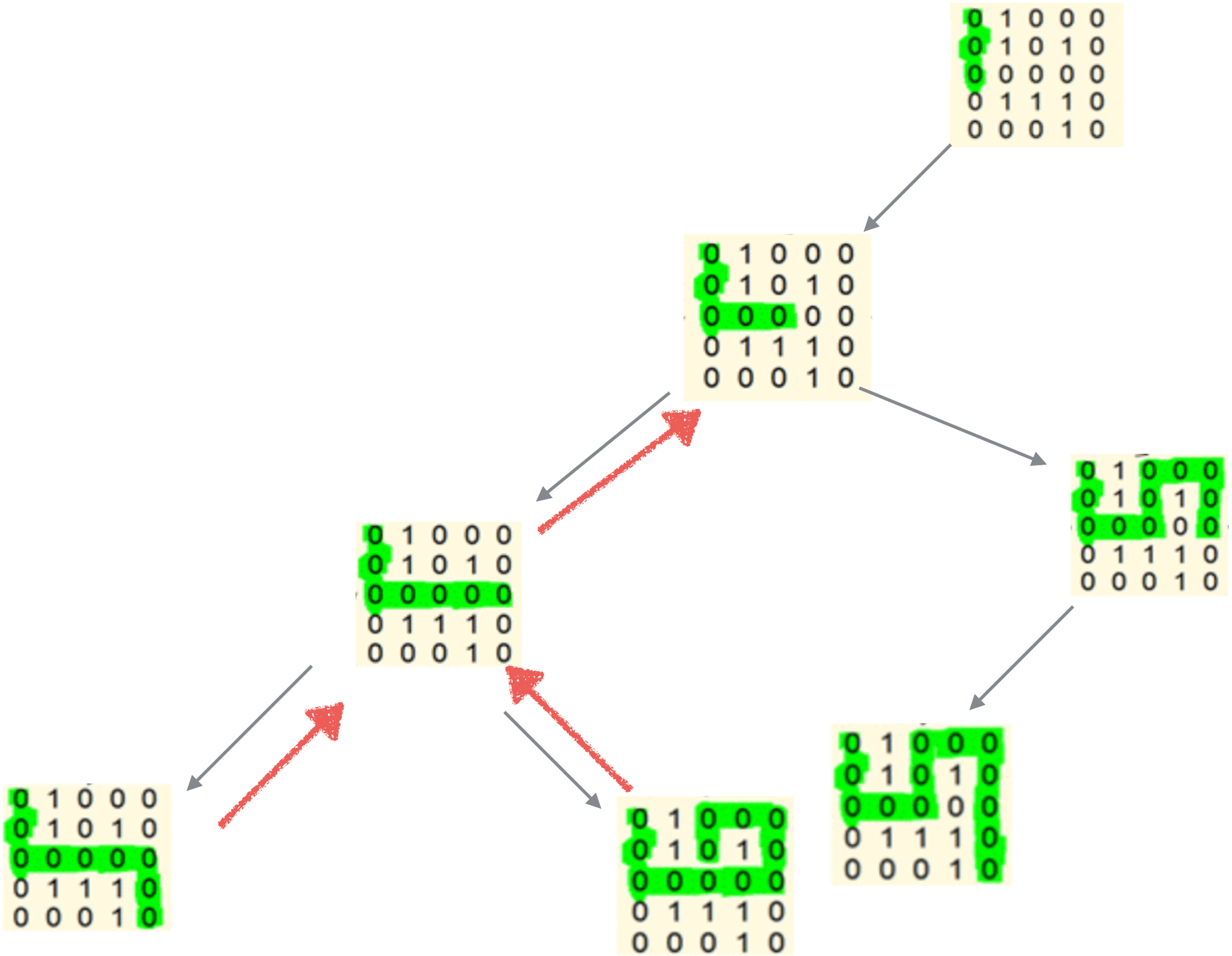
0	1	0	0	0
0	1	0	1	0
0	0	0	0	0
0	1	1	1	0
0	0	0	1	0

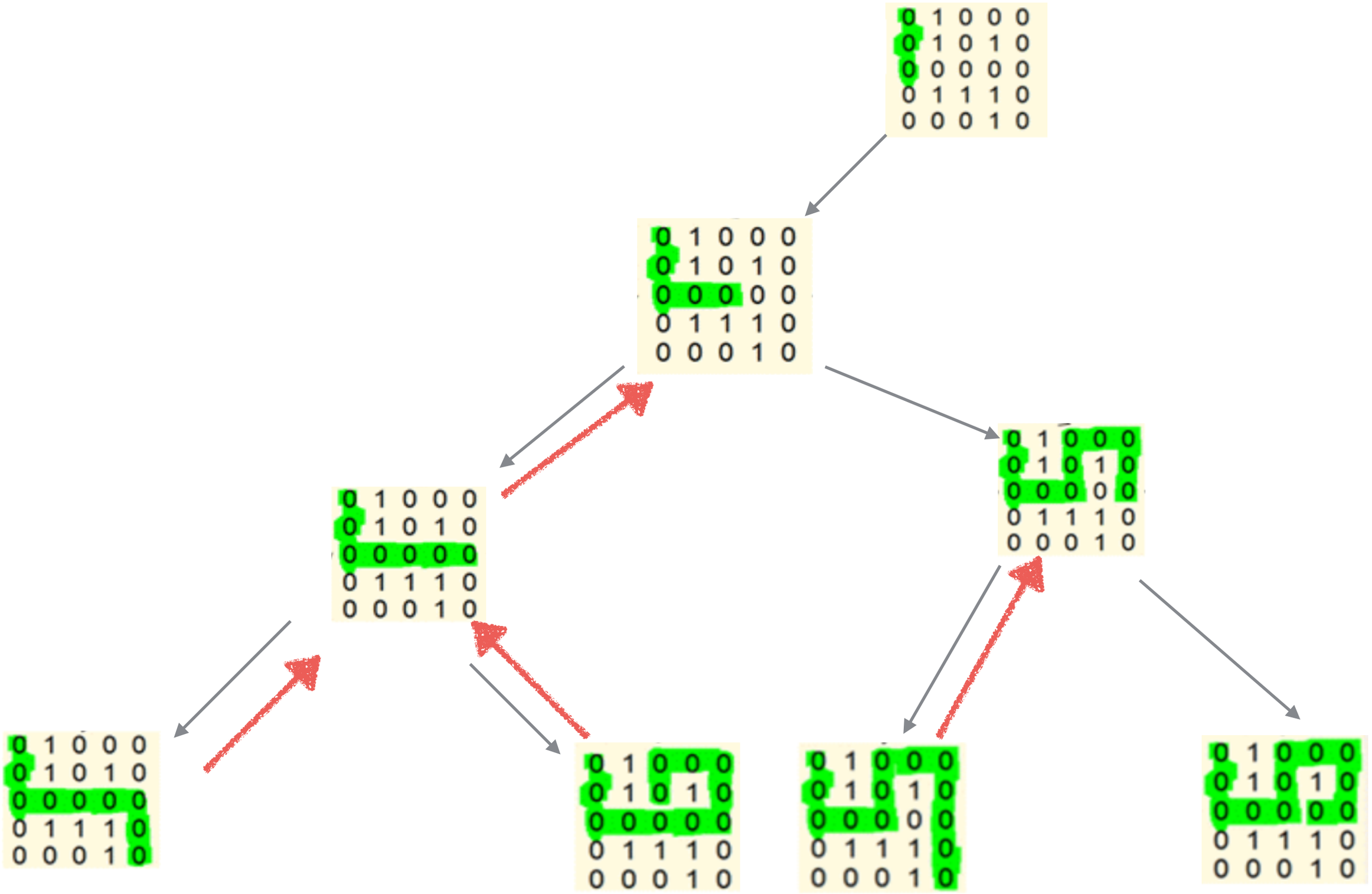
0	1	0	0	0
0	1	0	1	0
0	0	0	0	0
0	1	1	1	0
0	0	0	1	0

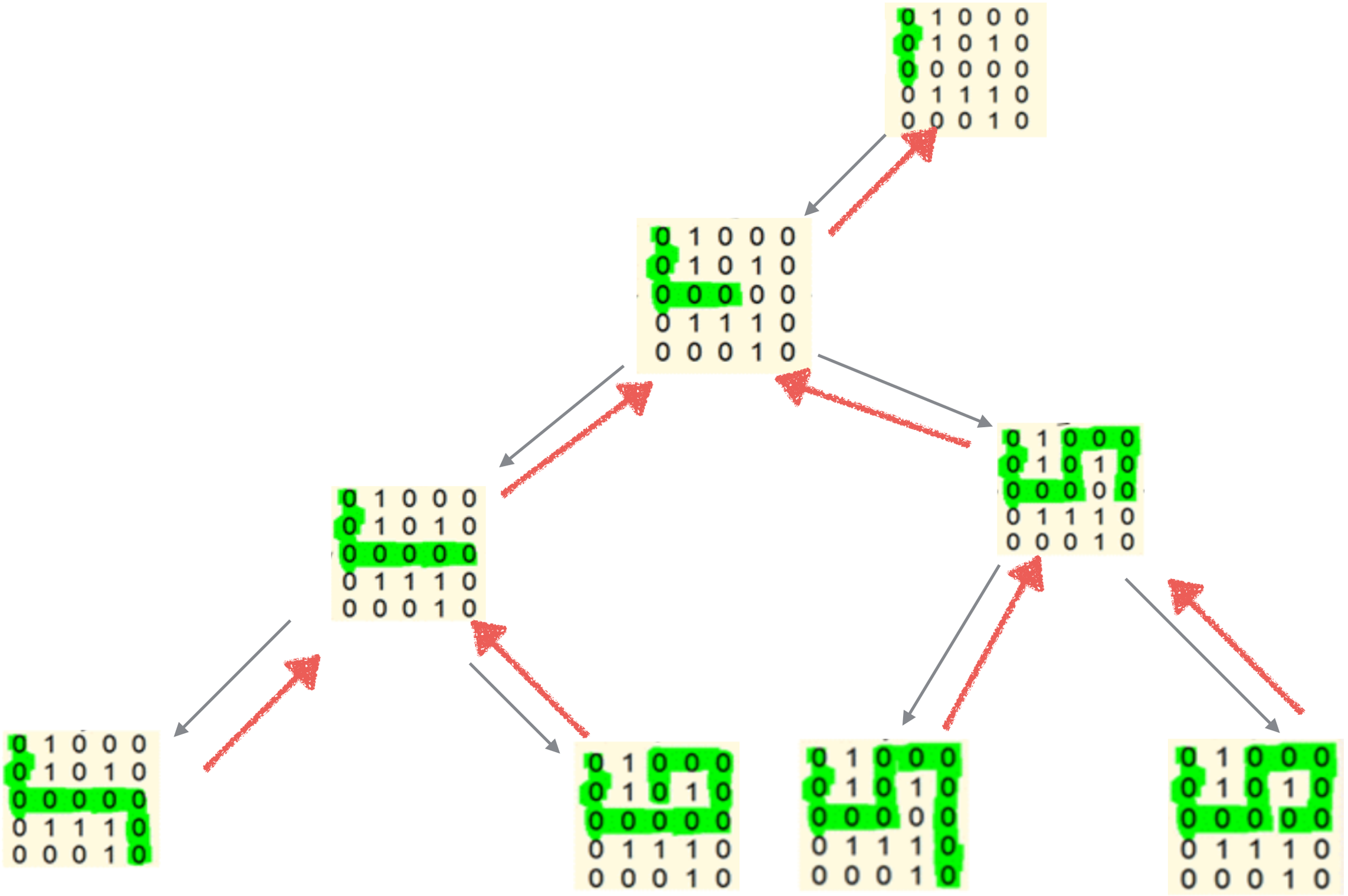

0 1 0 0 0
0 1 0 1 0
0 0 0 0 0
0 1 1 1 0
0 0 0 1 0
0 1 0 0 0
0 1 0 1 0
0 0 0 0 0
0 1 1 1 0
0 0 0 1 0
0 1 0 0 0
0 1 0 1 0
0 0 0 0 0
0 1 1 1 0
0 0 0 1 0
0 1 0 0 0
0 1 0 1 0
0 0 0 0 0
0 1 1 1 0
0 0 0 1 0

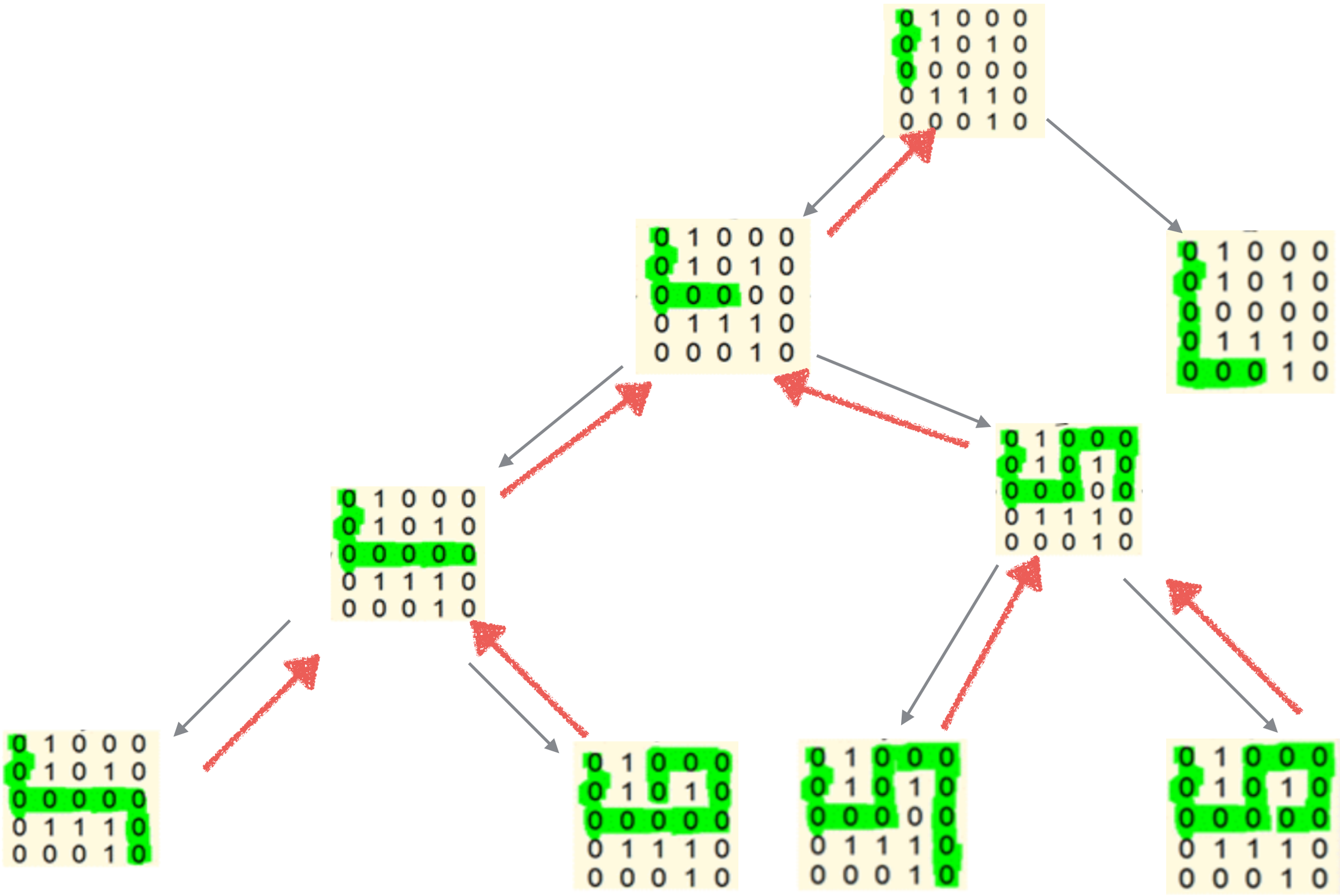

0 1 0 0 0
0 1 0 1 0
0 0 0 0 0
0 1 1 1 0
0 0 0 1 0
0 1 0 0 0
0 1 0 1 0
0 0 0 0 0
0 1 1 1 0
0 0 0 1 0
0 1 0 0 0
0 1 0 1 0
0 0 0 0 0
0 1 1 1 0
0 0 0 1 0
0 1 0 0 0
0 1 0 1 0
0 0 0 0 0
0 1 1 1 0
0 0 0 1 0

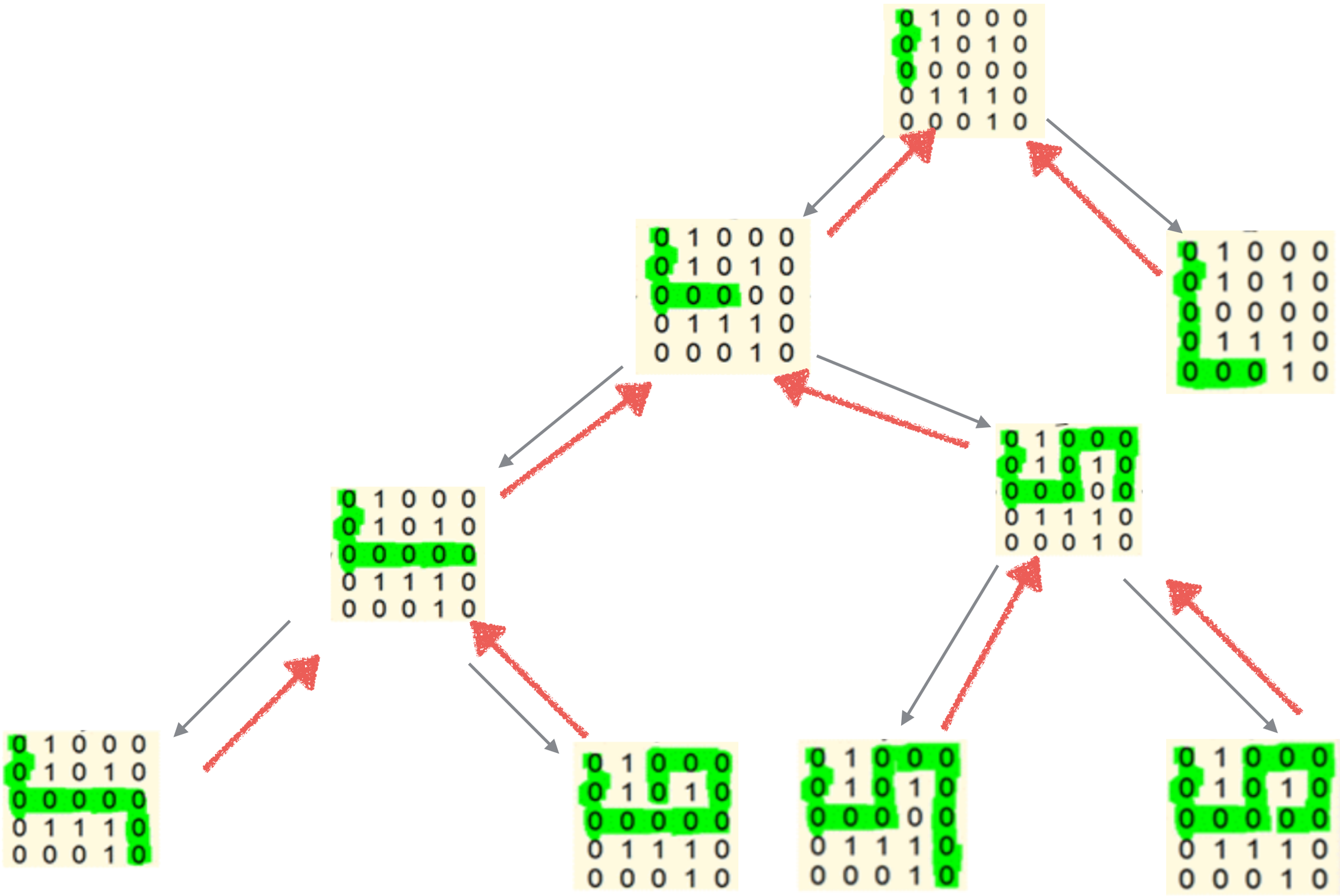

0 1 0 0 0
0 1 0 1 0
0 0 0 0 0
0 1 1 1 0
0 0 0 1 0


0 1 0 0 0
0 1 0 1 0
0 0 0 0 0
0 1 1 1 0
0 0 0 1 0


0 1 0 0 0
0 1 0 1 0
0 0 0 0 0
0 1 1 1 0
0 0 0 1 0




0 1 0 0 0
0 1 0 1 0
0 0 0 0 0
0 1 1 1 0
0 0 0 1 0


0 1 0 0 0
0 1 0 1 0
0 0 0 0 0
0 1 1 1 0
0 0 0 1 0

广度优先搜索（BFS）

- 从队列头部提取一个状态，使用生成规则拓展出所有合法状态，放入队列尾部
- 优点：如果搜索目标在解空间中存在，总能找到深度最小的路径
- 优点：不会暴栈
- 缺点：空间复杂度很大，搜索目标离初始目标很远的时候，会消耗大量内存

广度优先搜索（BFS）

```
while(状态队列非空)
    从队列中取出当前要搜索的状态
    if(达到目标状态) {
        ...
        return;
    }
    for(遍历下一层状态){
        ...
        将下一层状态加入队列
        ...
    }
}//非常简单的代码
```

0	1	0	0	0
0	1	0	1	0
0	0	0	0	0
0	1	1	1	0
0	0	0	1	0


0 1 0 0 0
0 1 0 1 0
0 0 0 0 0
0 1 1 1 0
0 0 0 1 0

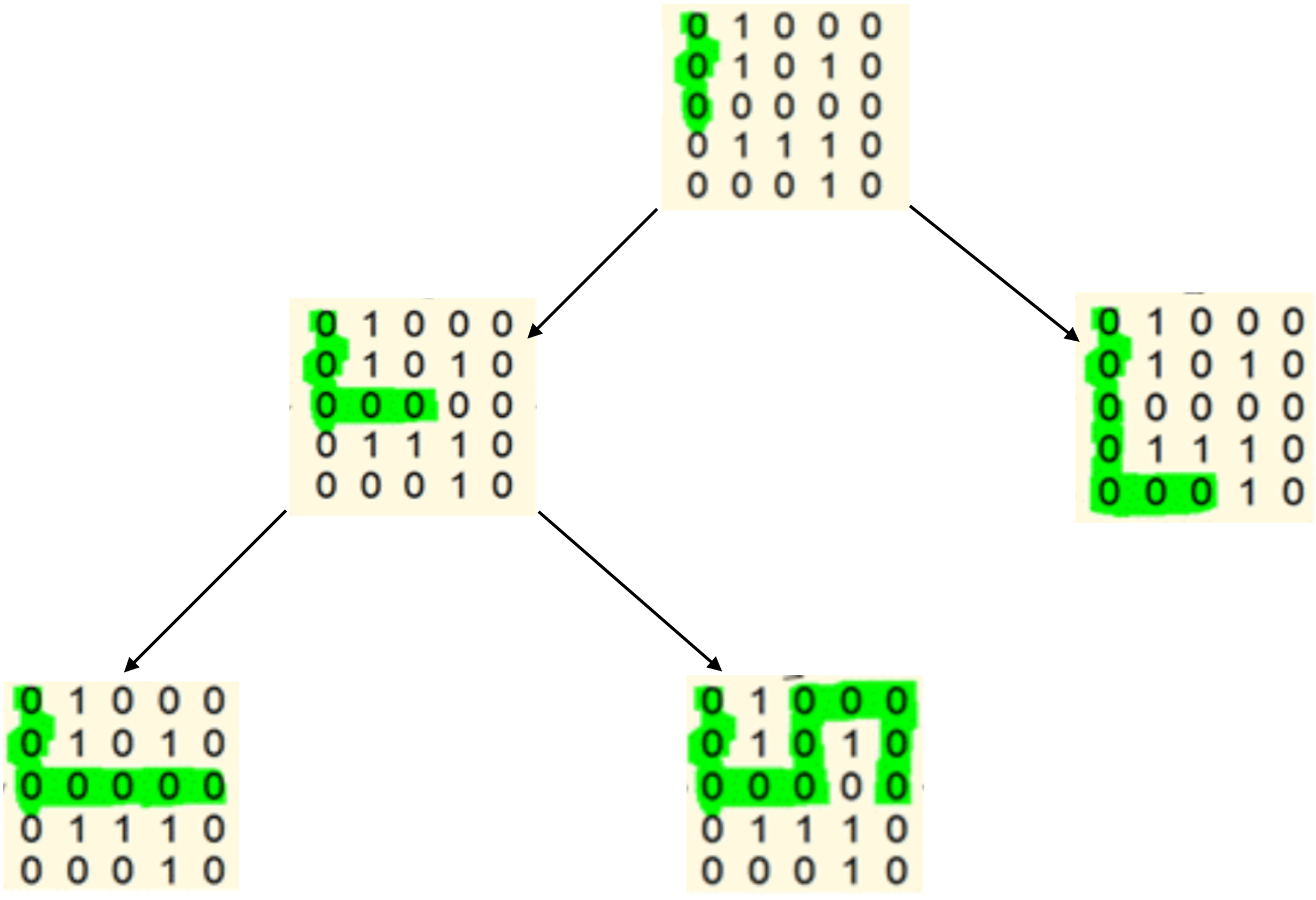

0 1 0 0 0
0 1 0 1 0
0 0 0 0 0
0 1 1 1 0
0 0 0 1 0

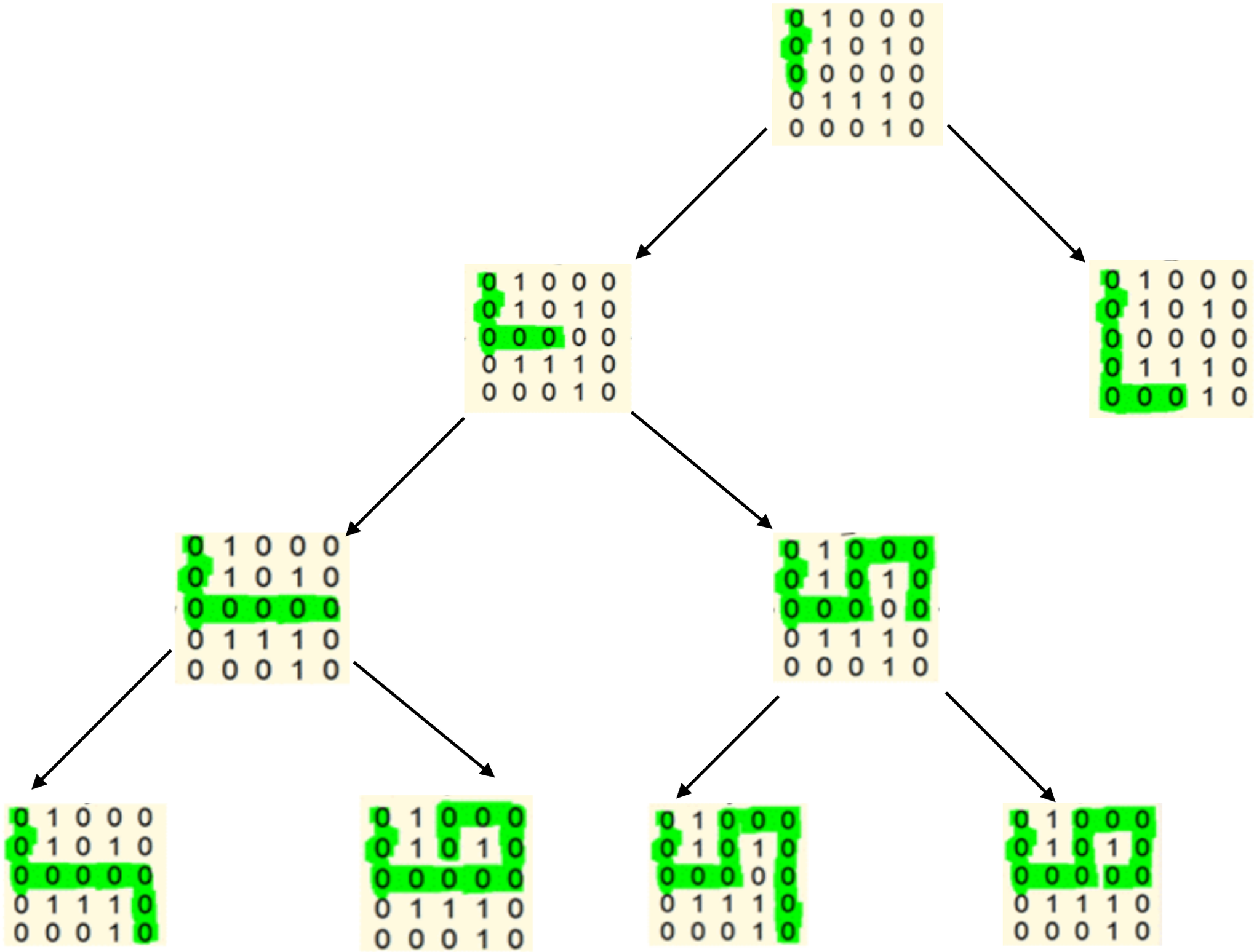

0 1 0 0 0
0 1 0 1 0
0 0 0 0 0
0 1 1 1 0
0 0 0 1 0


0 1 0 0 0
0 1 0 1 0
0 0 0 0 0
0 1 1 1 0
0 0 0 1 0


0 1 0 0 0
0 1 0 1 0
0 0 0 0 0
0 1 1 1 0
0 0 0 1 0


0 1 0 0 0
0 1 0 1 0
0 0 0 0 0
0 1 1 1 0
0 0 0 1 0


0 1 0 0 0
0 1 0 1 0
0 0 0 0 0
0 1 1 1 0
0 0 0 1 0
0 1 0 0 0
0 1 0 1 0
0 0 0 0 0
0 1 1 1 0
0 0 0 1 0
0 1 0 0 0
0 1 0 1 0
0 0 0 0 0
0 1 1 1 0
0 0 0 1 0
0 1 0 0 0
0 1 0 1 0
0 0 0 0 0
0 1 1 1 0
0 0 0 1 0


0 1 0 0 0
0 1 0 1 0
0 0 0 0 0
0 1 1 1 0
0 0 0 1 0

生成排列

- 设有n个整数的集合{1，2，…，n}，从中任取r个进行排列，输出所有排列
- 如果不是1-n呢?

生成排列

- 构造类搜索
- 状态表示： int num[]
- 生成规则： pos->pos+1
- 控制策略： DFS

生成排列

```
int num[maxn];
dfs(int pos){
   if(pos == n)输出
   for(i = 1->9)
      i若在num[1...pos]出现过
     continue
   num[pos + 1] = i
   dfs(pos + 1)
}
```

素数环

- 1到20这二十个数摆成一个环，要求相邻两个数的和是一个素数

素数环

- 构造类搜索
- 状态表示： int num[]
- 生成规则： pos->pos+1
- 控制策略： DFS

素数环

```
int num[maxn]
dfs(当前位置pos){
    if(pos == n && 与第一位的和为素数)
        搜索出结果，输出方案
    for(所有没有被用的数)
        如果不能与当前位置数组成素数
            continue
        放入num[pos+1]
        dfs(pos+1)
}
```

n皇后问题

- n*n的棋盘上放置n个皇后（n<=10），皇后会攻击上下左右以及四个对角线上的其他皇后
- 专题中I题

n皇后问题

- 构造类问题
- 状态表示：按行搜索int line[n]; int sta;
- 生成规则：k行->k+1行

  ➡ 怎么生成下一行?
- 控制策略：DFS

n皇后问题

- 最简单的搜法：全程开数组

```
25    void dfs(int x) {
26        if (x == n + 1) {
27            tot++;
28            for (int i = 1; i <= n; i++)
29                ans[tot][i] = a[i];
30            return;
31        }
32        for (int i = 1; i <= n; i++)
33            if (check(x, i)) {
34                a[x] = i;
35                dfs(x + 1);
36            }
37    }
```

棋盘上的马

- 给出中国象棋（n，m<=400）棋盘上的一个点，问马从这一点可以最短的以多少步走到棋盘上其他的点上，输出每个点的最少步数

棋盘上的马

- 遍历类搜索
- 状态表示：（x，y）；pair<int,int>，int dist[][]
- 生成规则：（x，y）->（x+dx，y+dy）
- 控制策略：BFS

倒水

- 大家一定觉的运动以后喝可乐是一件很惬意的事情，但是seeyou却不这么认为。因为每次当seeyou买了可乐以后，阿牛就要求和seeyou一起分享这一瓶可乐，而且一定要喝的和seeyou一样多。但seeyou的手中只有两个杯子，它们的容量分别是N 毫升和M 毫升 可乐的体积为S（S<101）毫升 (正好装满一瓶)，它们三个之间可以相互倒可乐 (都是没有刻度的，且 S==N+M，101＞S＞0，N＞0，M＞0)。聪明的ACMER你们说他们能平分吗？如果能请输出倒可乐的最少的次数，如果不能输出"NO"。

- http://acm.hdu.edu.cn/showproblem.php?pid=1495

倒水

- 隐式图搜索，遍历式问题
- 状态设计（S,M,N）->（M,N）
- 状态转移：倒水（Mn,Nn）->（Mn+1,Nn+1）
- 控制策略：BFS

填图颜色

- 由数字0 组成的方阵中，有一任意形状闭合圈，闭合圈由数字1构成，围圈时只走上下左右4个方向。现要求把闭合圈内的所有空间都填写成2.例如：6X6的方阵（n=6），涂色前和涂色后的方阵如下：

0 0 0 0 0 0
0 0 1 1 1 1
0 1 1 0 0 1
1 1 0 0 0 1
1 0 0 0 0 1
1 1 1 1 1 1

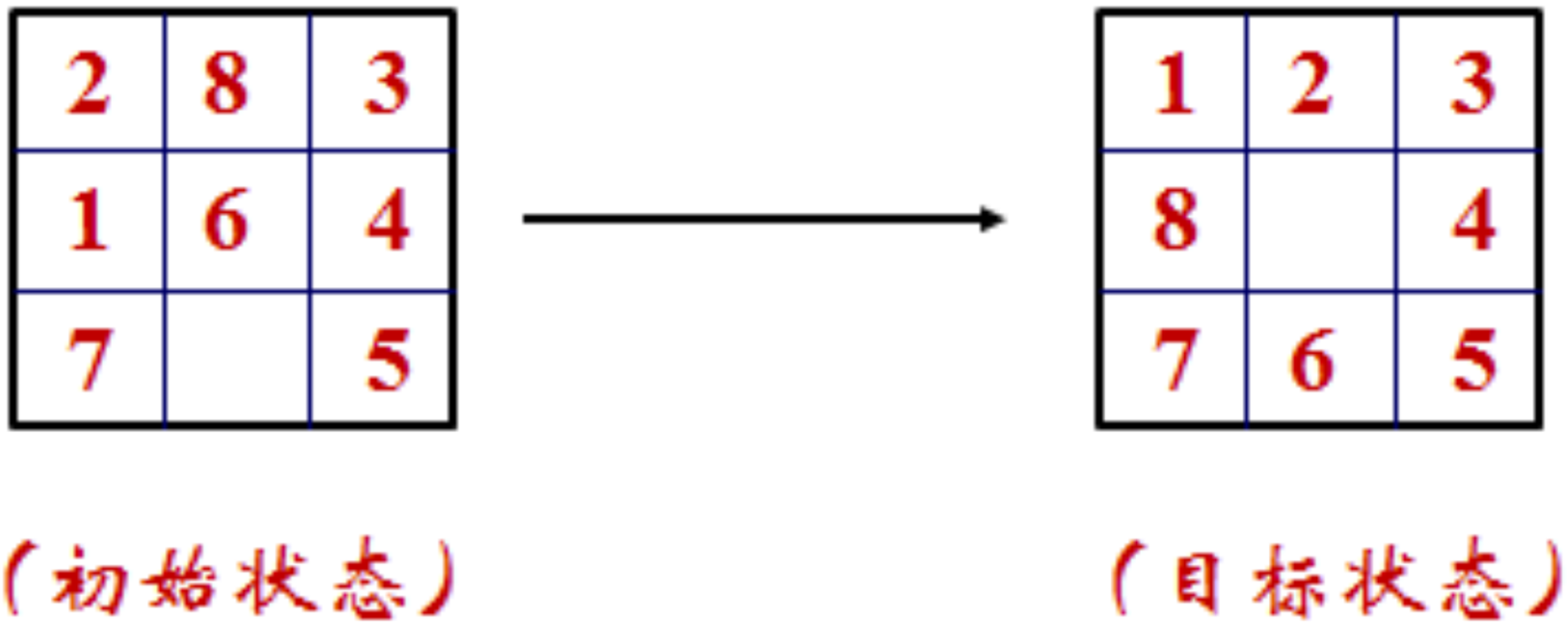
0 0 0 0 0 0
0 0 1 1 1 1
0 1 1 2 2 1
1 1 2 2 2 1
1 2 2 2 2 1
1 1 1 1 1 1

填图颜色

- 染色问题
- 从一个零向四周扩展进行判断，如果泄露了就非法，否则把相邻的所有零都赋值成2
- 在走过的格子上打标记，不走回头路
- FloodFill，染色，BFS不容易暴栈

8数码问题

八数码难题（8-puzzle problem）



（初始状态）　　（目标状态）

8数码问题：状态表示

- 选择一种数据结构来表示将牌的布局。对八数码问题,选用二维数组来表示将牌的布局很直观,因此该问题的综合数据库可以如下形式表示:

- (Sij),其中1<=i、j<=3,Sij∈{0,1,…,8},且Sij互不相等

- 之后会介绍状态表示的一种优化

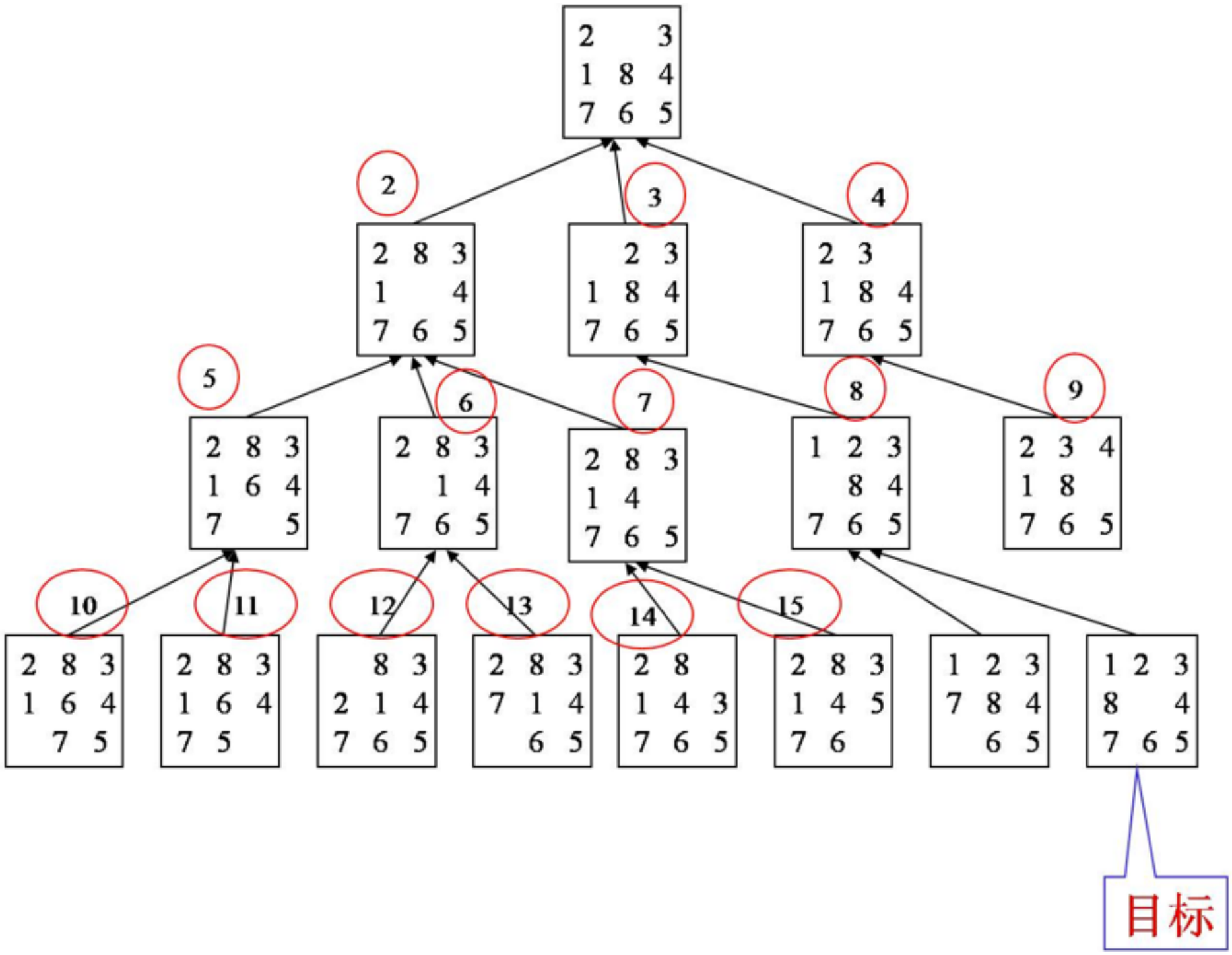
8数码问题：生成规则

- 改变状态的四种走法：上下左右
- 设Si,j记矩阵第i行第j列的数码,i0、j0记空格所在的行、列数值,即Si0,j0=0,则
- 1.if j0-1≥1 then Si0j0:=Si0(j0-1), Si0(j0-1):=0;(Si0j0向左)
- 2.if i0-1≥1 then Si0j0 :=S(i0-1)j0, S(i0-1)j0:=0;(Si0j0向上)
- 3.if j0+1≤3 then Si0j0 :=Si0(j0+1), Si0(j0+1):=0;(Si0j0向右)
- 4.if i0+1≤3 then Si0j0 :=S(i0+1)j0, S(i0+1)j0:=0;(Si0j0向下)

8数码问题： 控制策略

- DFS
- BFS
- 启发式/A*

8数码问题


2 3
1 8 4
7 6 5
2
3
4
2 8 3
1 4
7 6 5
2 3
1 8 4
7 6 5
2 3
1 8 4
7 6 5
5
6
7
8
9
2 8 3
1 6 4
7 5
2 8 3
1 4
7 6 5
2 8 3
1 4
7 6 5
1 2 3
8 4
7 6 5
2 3 4
1 8
7 6 5
10
11
12
13
14
15
2 8 3
1 6 4
7 5
2 8 3
1 6 4
7 5
8 3
2 1 4
7 6 5
2 8 3
7 1 4
6 5
2 8
1 4 3
7 6 5
2 8 3
1 4 5
7 6
1 2 3
7 8 4
6 5
1 2 3
8 4
7 6 5
目标

重复状态判断

- 之前的一些搜索中，只要使用一定的策略，或者简单的vis数组，可以保证状态不重复
- 如果会有状态重复怎么办?
- Hash？花式Hash！！将状态转换成一个方便判断是否相同的格式

重复状态判断

- 数组不行的话，用STL
- set<type>,map<type1,type2>,unordered_set/map
- 判断是否在内:set.find(xxx)==set.end(),map.count(xxx)=0
- type可以是个vector,string,pair,以及定义了小于号还是啥的class

8数码问题

- 状态编码：康拓展开
- n排列中每一个排列都可以与n!中的一个数对应
- 比如，1324，第一位是1小于1的数没有，是0个 0*3!；第二位是3，小于3的数有1和2，但1已经在第一位了，即1未出现在前面的低位当中，所以只有一个数2，1*2!；第三位是2，小于2的数是1，但1在第一位，即1未出现在前面的低位当中，所以有0个数 0*1!；
- 比1324小的排列有0*3!+1*2!+0*1!=2
- 9! = 362880

8数码问题

```
long long fact[12]={0,1,2,6,24,120,720,5040,40320,362880,
                    3628800,39916800};
longlongCantor(string buf) // buf = "1324"
{
    int i,j,counted;
    long long result=0;
    for(i=0;i<4;++i)
    {
        counted=0;
        for(j=i+1;j<4;++j)
            if(buf[i]>buf[j])
                ++counted;
        result=result+counted*factory[4-i-1];
    }
    return result;
}
```

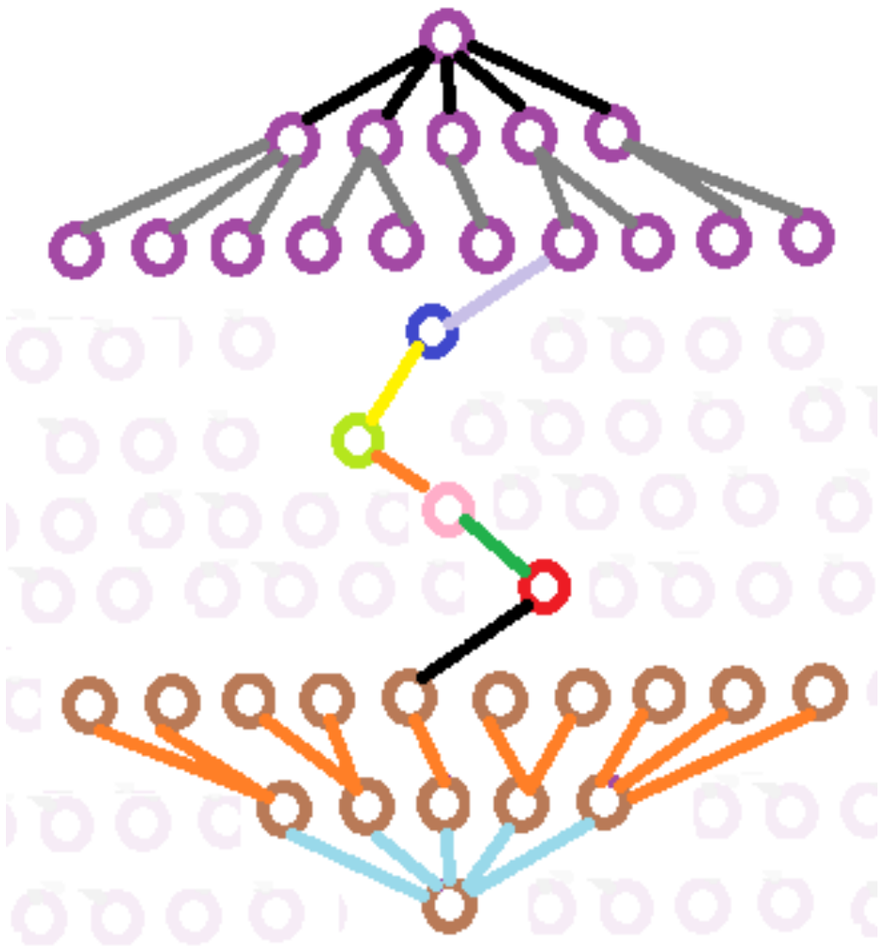
状态表示优化

- 位运算讲稿 - Matrix67
- 用位运算来存储状态，状态->long long int
- 加快判重速度，减少存储空间

搜索策略优化

- 双向宽度优先搜索
- 从起点和终点两个状态同时扩展，检查两边状态是否有重叠
- 会写宽搜就会写双向宽搜，但不是所有宽搜都能写成双向
- 理论上能减少一半的搜索量，实践中更优秀
- 不是交替拓展，而是控制两个队列大小接近

搜索策略优化



搜索策略优化

- 根据题目性质，减小搜索状态空间

- 给出n个正整数a1,a2,a3,...,an，和一个正整数k, 问是否能从n个数中选出若干个数，使其和为k

- 裸的爆搜

搜索策略优化

- 根据题目性质剪枝

- 给出n个正整数 a1,a2,a3,...,an，和一个正整数k, 问是否能从n个数中选出若干个数，使其和为k

```
bool answer=false;
void dfs(int i,int sum)
{
    if(i==n)
    {
        if(sum==k)
            answer=1;
        return;
    }
    dfs(i+1,sum+a[i]);
    dfs(i+1,sum);
}
```

- 裸的爆搜

搜索策略优化

- 给出n个正整数a1,a2,a3,...,an，和一个正整数k, 问是否能从n个数中选出若干个数，使其和为k

- 剪枝1：当sum>k时，直接返回

- 剪枝2：设s[i]表示第i个数到最后一个数的所有数和
  当遍历到第i个数时，若此时sum加上所有剩下的数也达不到k，即sum+s[i]<k，直接返回

搜索策略优化

- 迭代加深搜索

- 总体上按照深度优先算法方法进行

- 对搜索深度需要给出一个深度限制dm,当深度 达到了dm的时候,如果还没有找到解答,就停 止对该分支的搜索,换到另外一个分支继续进 行搜索。

- dm从小到大依次增大(因此称为迭代加深)

- 重复搜索深度较小的节点,是否浪费?

搜索策略优化

- 卿式剪枝
- (clock() - start / CLOCKS_PER_SEC)计算程序跑了多久，快TLE了就停下来
- rand()一下，一定的概率不向某个状态搜下去
- 提示：专题里的背包可以用

Jack and Jill

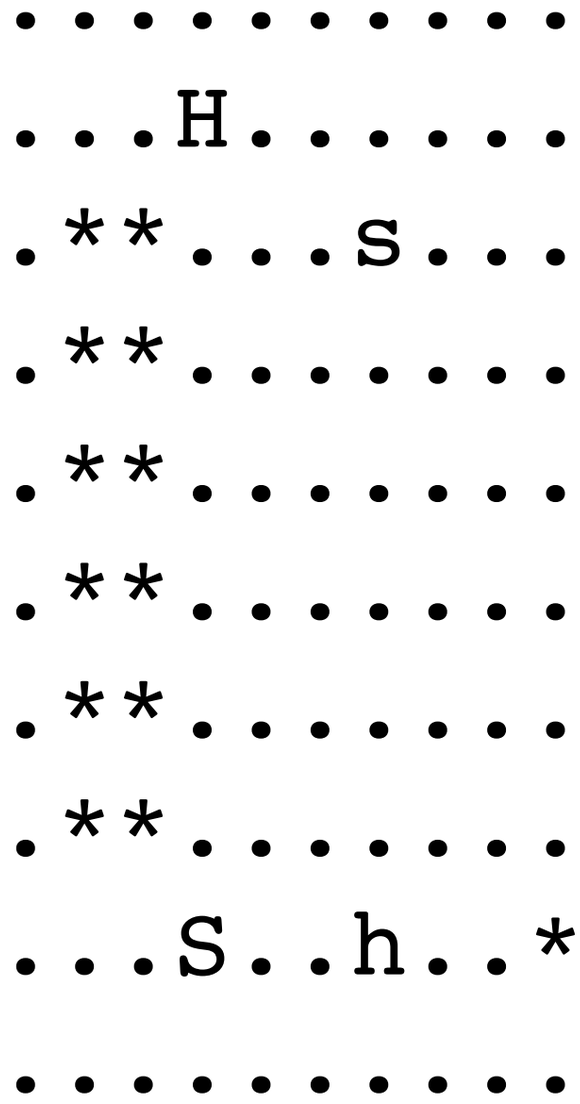
- Jack和Jill要从各自的家走到各自的学校,但是他们互相不喜欢对方,因此希望你找到一个行走方案,使得在行 走的过程中他们之间的直线最近距离最远。

- 可以看做他们都是处于一个n*n的矩形的某个格子上（n<=30）。单位时间内每个人都可以从一个格子走到相邻的（上下左右）四个格子之一,也可以停止不动。

- 计算距离的时候只算单位时间移动结束之后两人所在位置的直线距离,不考虑移动过程中的距离。

Jack and Jill

- H:Jack的家，S:Jack的学校
- h:Jill的的家，s:Jill的学校
- *:不可进入的地方
- .:可以通过的地方



Jack and Jill

- 数据库:
  - 设计状态（p1,p2）为两人所在格子位置
- 生成规则:
  - 一个格子对向另一个格子对转移

Jack and Jill

- 控制系统:
  - 多种多样，DFS + 剪枝，BFS + 双向，BFS + A*
  - 在DFS时不断控制距离
  - 在BFS时限定距离，双向搜
  - 在BFS时使用优先队列维护（最简单的A*）

Knight's Problem

- 有一个骑士,在一个无穷大的棋盘上。骑士一开始位于坐标(fx,fy),目的地坐标是(tx, ty)。骑士拥有m种行走规则。一个行走规则 用(mx,my)表示,说明骑士如果当前处于 坐标(x,y),那么使用这种行走规则后下一步会到达坐标(x+mx,y+my)。问骑士从起 点到目标点最少需要多少步?

- 数据范围:
  -10<=mx,my<=10
  -5000<=fx,fy,tx,ty<=5000

Knight's Problem

- 暴力BFS
  - 以起点坐标为起始点,使用广度优先搜索。
  - 每次枚举行走规则,并将新走到的点加入到广搜队列中。
  - 使用hash表来判重,对于已经搜到的点不再重复搜索。
  - 一旦搜索到目标节点,那么当前所需步数就是最少步数,退出搜索,输出。

Knight’s Problem

- 猜想1:
  - 总有一种最优方案,使得骑士整个行走路径在以起点和终点为对角的矩形内或者离该矩形边界不超过一次行走的最大距离(对于本题来说, 一次行走的最大距离在x方向和y方向上都不超过10)。

- 复杂度O(5010×2×5010×2)

Knight’s Problem

- 猜想2:
  - 存在一条最优路径,该路径离上的任何一点, 距离起点到终点所连线段的距离都不超过3K（K为一次行走的最大距离）

- 复杂度约O(5000×10×10)

摧毁车站

- 给定你一个n个点,m条边的有向图,边的长度都是1。问你最少需要删掉多少个点 （删掉一个点会将与其相连的边同时删除），才能使得1号点到n号点之间不存在 长度小于等于k的路径。不允许删除1号或者n号点。

- 输入n,m,k和图的连边情况,
  n<=50,m<=4000,k<=1000

摧毁车站

- 这真的是搜索么?
- 删除的点必须在最短路上
- n很小，每次枚举一个在最短路上的点删掉进行DFS
- 大力出奇迹

吴队长征婚

Time Limit: 10000/4000MS (Java/Others) Memory Limit: 65535/65535KB (Java/Others)

吴队长征婚这件事因为请客而没有传出去(虽然他忘了请一个队吃饭)，于是吴队长高兴地玩起了木棒。吴队长拿了一些长度相同的木(guang)棒(gun)，随机的把它们截成了N段，每一段最长50。现在他想把这些木棒还原成原来的状态，但是他忘记了原来的木棒有多少根，也忘记了每一根有多长。请帮助他设计一个程序，帮他计算之前木棒可能的最小长度。输入数据保证每一段木棒长度都大于0。

Input

输入有多组数据，每组数据分为两行。
第一行一个整数N，表示截断之后木棒的个数。$1 \leq N \leq 64$
第二行N个整数，表示截断之后的每一段小木棒的长度。 当N=0时，表示输入结束。

Output

每组数据输出一个整数占一行，表示原木棒可能的最小长度。

吴队长征婚

- 剪枝1：吴队长拿了一些长度相同的木(guang)棒(gun)，所以总长度是固定的，且原来每根木棒必须是总长度的约数
- 所以枚举长度
- 剪枝2：既然枚举了长度，那么如果DFS出的长度超过了当前枚举的长度，GG（从大到小排序）

吴队长征婚

- 剪枝1：吴队长拿了一些长度相同的木(guang)棒(gun)，所以总长度是固定的，且原来每根木棒必须是总长度的约数
- 所以枚举长度
- 剪枝2：既然枚举了长度，那么如果DFS出的长度超过了当前枚举的长度，GG（从大到小排序）
- 剪枝3：如果并没有搜出来当前的木棒长度，那么也不用继续搜了（相同值只考虑一次）
- 剪枝4：由于我们保证了长度是总长的约数，所以只要搜出来一个合法长度，就不用拆开它，直接用剩下的木棍搜剩下的合法长度

```
bool dfs(int x,int y,int z,int len){//x表示装好了几个木棍，y表示装了多少，z表示上一个用的位置，len表示木棍长度
        if(x == num)return 1;//如果搜索到最后一根木棍
        for(int i=z; i<n; i++){
                if(vis[i]) continue;
                if(y+a[i] == len){
                        vis[i] = 1;
                        if(dfs(x+1,0,0,len)) return 1; // 如果当前搜出了一个符合长度的，那么它肯定是一个合法的情况，就不需要拆它了
                        vis[i] = 0;
                        return 0; //如果当前这个木棒在合成后正好符合长度要求，但是之后的搜索确是失败的，说明后面也无法合成满足要求的木棒，也直接return false就可以
                }else if(y+a[i] < len){
                        vis[i] = 1;
                        if(dfs(x,y+a[i],i+1,len)) return 1; // 如果当前搜出了一个符合长度的，那么它肯定是一个合法的情况，就不需要拆它了
                        vis[i] = 0;
                        if(y==0) return 0; // 从长度为0开始dfs 最后回到了0，在这一层没有满足的，回溯
                        while(a[i]==a[i+1]) i++; // 如果某个木棒在这个状态搜索失败，那所有与这个木棒相同长度的木棒就不用在这次搜索中搜索了
                }
        }
        return 0;
}
```

Thanks For Your Attention