**数独** 递归回溯算法解决 或算法。改进点：可以找限制最多的单元格填入

**棋盘** 递归回溯获取所有合法的棋局 map存储所有结果

**字符串的替换** 把每个单词视为一个点，把所有点都根据R(优先)和长度进行双关键字排序，然后每一对同义词就是一条有向边。把单词排序后，遍历单词dfs一遍，对某个单词u，把u的所有后代的r和sz都更新为u的r和sz，最后再更新一遍文章的就ok了。

**贪吃蛇** bfs cango go

**SCC** Tarjan缩点，然后dfs //对邻接矩阵进行预处理，处理出来每个节点能达到的所有节点（行）和能到此节点的所有节点（列） //dfs处理邻接矩阵 fa 父城市 传递闭包将每条路变成revolutionary roads，查找最大强连通分量

**LCA** 倍增算法 dfs先预处理树

**pth couson** dfs遍历子节点并标注时间戳，将n倍祖先分解成2的次方

**背包** 递归实现 从最后开始

**花瓶插花** F(i,j) = max{F(i,j-1),F(i-1,j-1)+Vij},前i束花摆在前j个花瓶里

**二分法**对优先级进行二分 每次给一个优先级就模拟一下看是否合格 不合格就二分调整

推箱子两重BFS，第一重是搜索箱子，第二重搜索人能不能到达推箱子的地方;Hash方法记录状态去重

**dp** for(int j=1;j<=V;j++){int minNum = Math.min(F,j);

for(int i=1;i<=minNum;i++){

//记录dp路径，dp[i][j]表示前i多花在j个瓶中的最大值

//dp[i][j] = Math.max(dp[i][j-1], dp[i-1][j-1]+aesValue[i][j]);

记录status，循环结束存储位置

**p-th cousin** void dfs(int v, int lev){

timdfseIn[v] = ++timer; level[v] = lev; //节点v的深度

allLevel[lev].add(v); //在同一深度的list里加入v

parents[v][0] = parent[v] == -1 ? v : parent[v]; //填写v的父节点

for (int i=1; i<parents[v].length; ++i) parents[v][i] = parents[parents[v][i-1]][i-1]; //v的2^k父节点等于V的2^k-1父节点的2^k-1节点

for(int child: childs[v]){dfs(child, lev + 1); //递归，搜索子节点}

timeOut[v] = ++timer;

//求n倍祖先 eg 3=2^1+2^0 int nthParent(int v, int n){

int curV = v; for(int l = parents[v].length - 1; l >= 0; --l){

if(n>=(1<<l)){n-=(1<<l); curV = parents[curV][l];}} return curV;}

**强连通分量** Tarjan缩点，然后dfs

public void preProcessAdjMatrix(){

boolean[] isVisited =new boolean[city\_num+1];Arrays.fill(isVisited, false);

for(int i=1;i<=city\_num;i++){

adj\_matrix[i][i] = 1;dfs(i,0,isVisited);visited[i]=true;Arrays.fill(isVisited, false);}}

//dfs处理邻接矩阵 fa 父城市 传递闭包

public void dfs(int city,int fa\_city,boolean[] isVisited){

isVisited[city] = true;int temp;

for(int i=1;i<=city\_num;i++){temp = adj\_matrix[city][i];

if(temp!=0&&!(isVisited[i])&&!(visited[city]))dfs(i,city,isVisited);}

for(int i=1;i<=city\_num;i++){if((adj\_matrix[city][i])!=0)adj\_matrix[fa\_city][i] = 1;}}

dp思路

先预处理出从第一列到i列的白点与黑点的数量。设dp[i][j]为前i列的以颜色为j的合法状态下的最小更改数量

最小编辑距离，也是求最长公共子序列，dp[i][j]表示第一个字符串的前i个字符要和第二个字符串的前j个字符匹配需要的最少操作次数

\*状态转移公式：若A[i]==B[j],dp[i][j]=dp[i-1][j-1];若不相等，dp[i][j]=min{dp[i-1][j]+1, dp[i-1][m-1]+1, dp[i][j-1]+1}

具有重叠子问题，使用记忆化搜索（空间换时间）

F(W) = max{F(W-Wj)+Vj},(Wj<W).可以采用反向递归的方法或者正向两层for循环的方法

M[i,j] = min{M[i,k]+M[k+1,j]+di\*dk+1\*dj+1}(i<=k<=j-1)其中di是矩阵Ai的第一维,dk+1是断开处矩阵Ak的第二维（即Ak+1的第一维），

\*dj+1是最后一个矩阵 Aj 的第二维。

要使得两组数字的和相差最少，那么每一组的和都应该尽量靠近总和的一半，那么也就是说，从这n个数字中选出一部分，使其和尽可能接近sum/2。显然这是一个0-1背包问题。

首先用dijkstra算法求出每个节点到s的最短路d[]数组，属于DP里面的求解的问题有重叠子问题

F(w,j) = max{F(W,j-1), F(W-Wj,j-1)+Vj}.可以采用正向两重循环或者反向递归的方式

图

popular cows 极大强连通分量+缩点问题。可以通过Tarjan算法或者Kosataju算法求有向图的极大强连通分量，这里使用后者

\*【Kosataju算法步骤】

\*1)在该图的逆图（所有边做反向）上运行dfs；

\*2)在原图上，按第一步得出的后序编号的逆序进行dfs。也就是说，在第二次dfs时，每次都挑选当前未访问的结点中具有最大后序编号的顶点作为dfs树的树根。

字符串替换 把每个单词视为一个点，把所有点都根据R(优先)和长度进行双关键字排序，然后每一对同义词就是一条有向边。

\*把单词排序后，遍历单词dfs一遍，对某个单词u，把u的所有后代的r和sz都更新为u的r和sz，最后再更新一遍文章的就ok了。

Scc Tarjan缩点，然后dfs

递归问题

pair 递归解决，当处理掉一个完整的最大的pair后，处理结束，若int多了则报错

给定矩形和切分数，并给出每一个切分方案的坐标(x1,y1,x2,y2),输出切分后的所有的矩形的面积（一次切分只会分割一个最小矩形）：递归求解，深度优先搜索切分，找出切分后的每一个最小的矩形

数独 典型的递归回溯算法解决。改进点：可以找限制最多的单元格填入

棋盘 dfs递归回溯获取所有合法的棋局

qtree：倍增算法求LCA

推箱子：两重BFS，第一重是搜索箱子，第二重搜索人能不能到达推箱子的地方;Hash方法记录状态去重