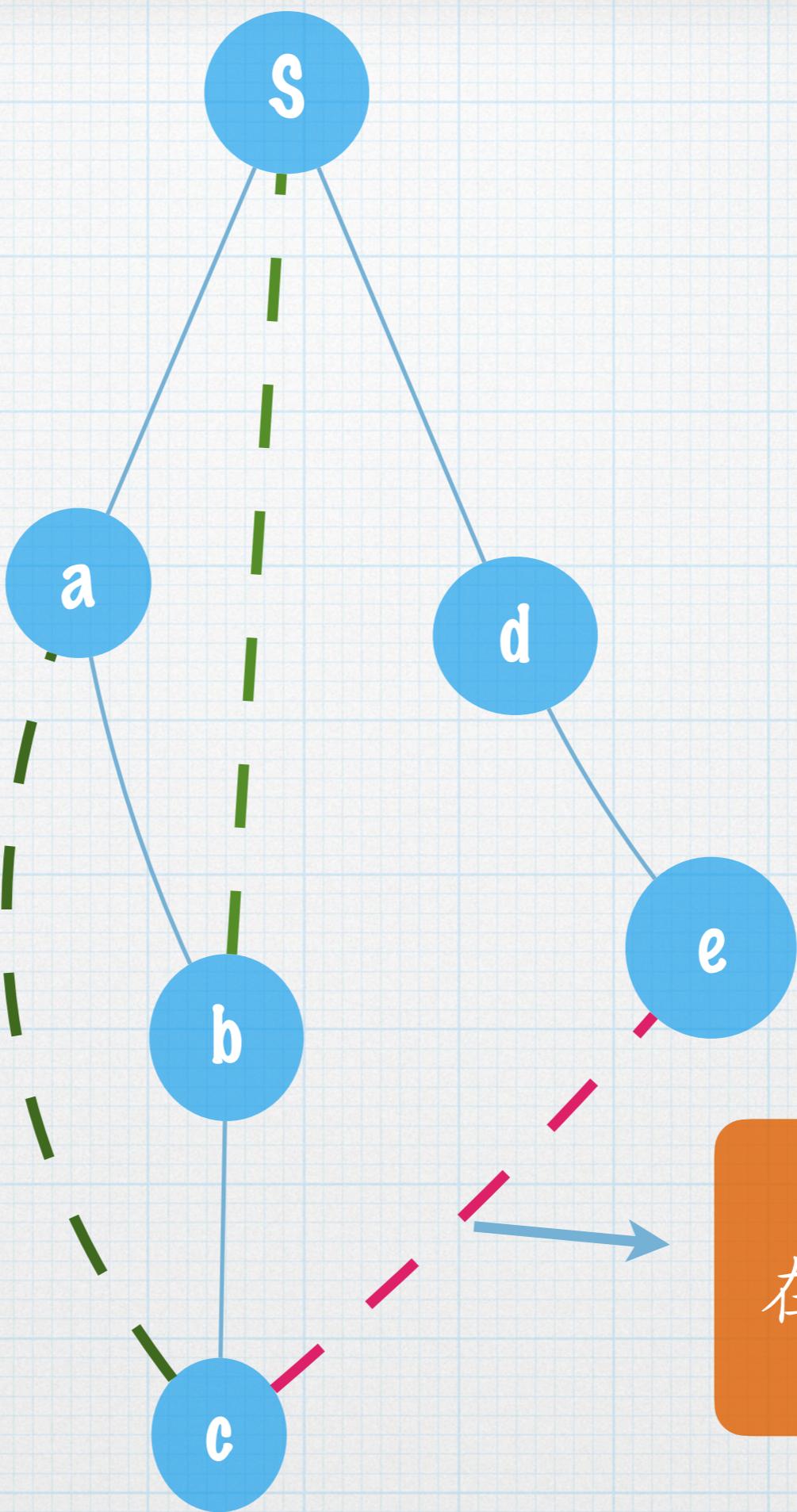


# 图论入门

- \* dfs树-tarjan系列点双，边双，强联通
- \* dfs树与随机数
- \* Dijkstra (heap)
- \* Floyd
- \* 无向图最小环
- \* bellman-ford/spfa
- \* 差分约束
- \* topological sorting
- \* 图的绝对中心

# dfs树

- \* 对一副图进行dfs后按照dfs的顺序形成的一棵生成树
- \* 图中的边可以分为树边与非树边
- \* 无向图中，非树边只会以返祖边（后向边）的形式存在
- \* 有向图中，非树边还会以前向边与横叉边的形式存在



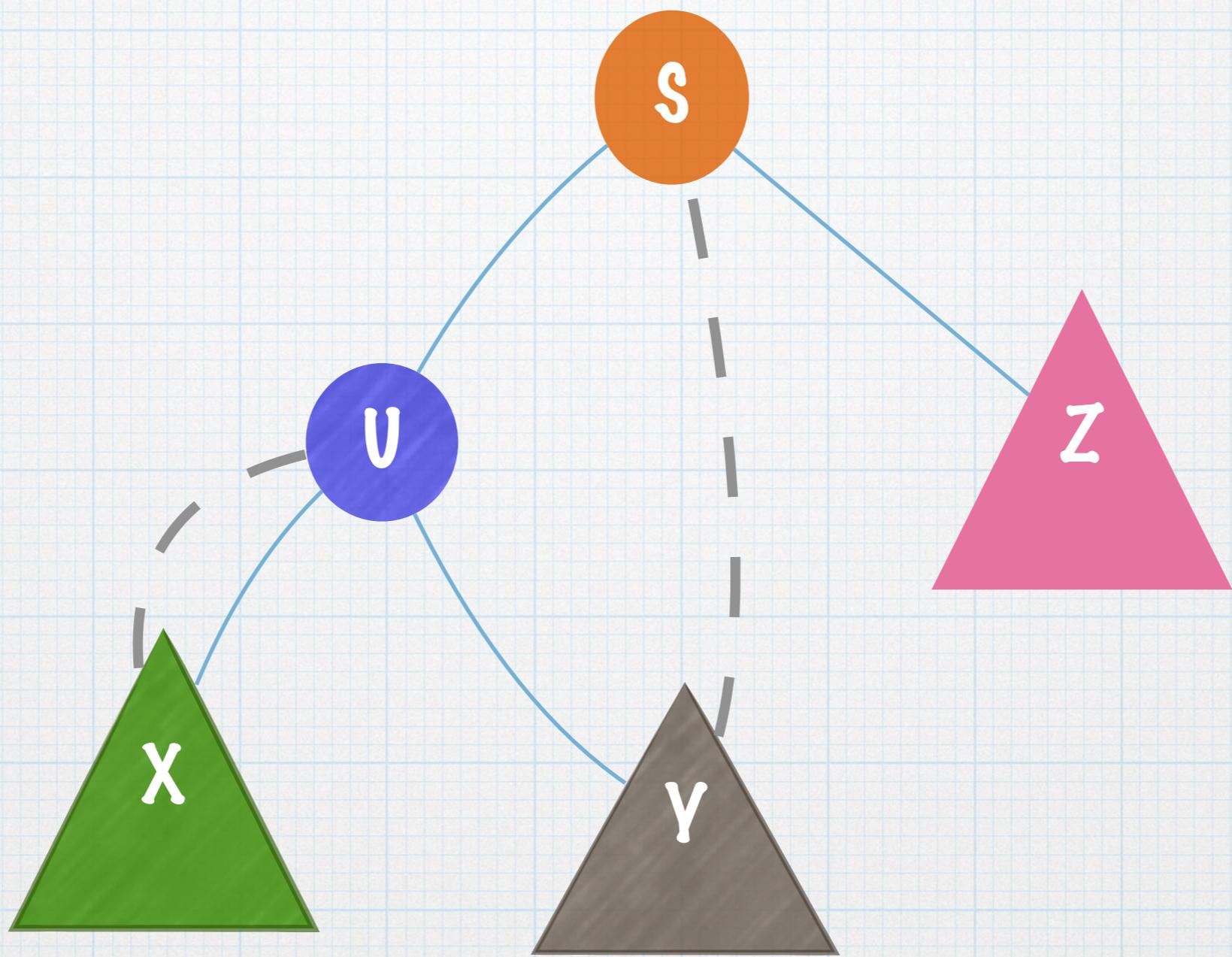
不存在这种边



# 割点

- \* 去掉割点后，图会不连通
- \* 树的非叶节点都是割点

- \* 割点的两种情况
- \* 1: 根节点在dfs树中有两个或以上的儿子，根节点为割点
- \* 2: 非根节点 $u$ 至少有一个子树没有返祖边可以跨过 $u$



# 割边

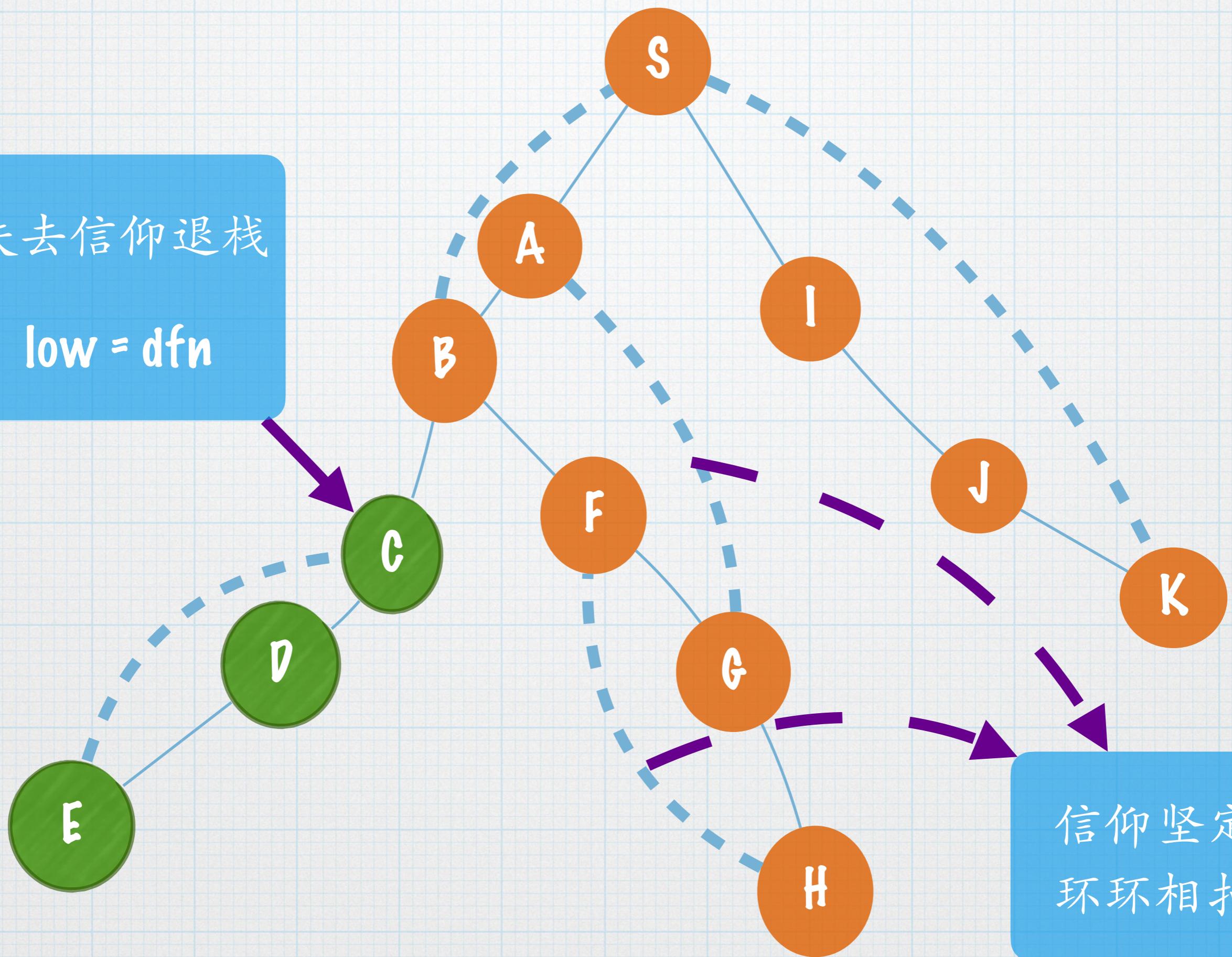
- \* 去掉割边后图就不连通，也称为桥
- \* 一条边是割边当且仅当子树内返祖边都无法跨过这条边
- \* 将所有的割边去掉后，整个图会分成若干个双联通分量（双联通子图），每一个联通块内去掉一条边还是连通

# 点双与边双具体求法

- \* 每个节点维护两个信息
- \* 1: 第一次访问到的时间戳  $\text{dfn}$
- \* 2: 当前节点以及子树内部通过返祖边最早能访问到的时间戳  $\text{low}$
- \* 如果  $\text{low}[\text{from}] == \text{dfn}[\text{from}]$  说明  $\text{from}$  上面的那条边就是割边
- \*  $\text{low}[\text{to}] >= \text{dfn}[\text{from}]$  说明  $\text{to}$  子树无法返回到  $\text{from}$  上面,  $\text{from}$  为割点
- \*  $\text{dfs}$  的时候用一个栈保存搜到的点, 回溯的时候一旦发现  $\text{low}[\text{from}] == \text{dfn}[\text{from}]$  就将栈中  $\text{from}$  子树内的点都退出, 并标记成同一个连通分量
- \* 求边双也可以使用树上差分, 每条返祖边覆盖一段路径, 割边被返祖边覆盖的次数为 0

失去信仰退栈

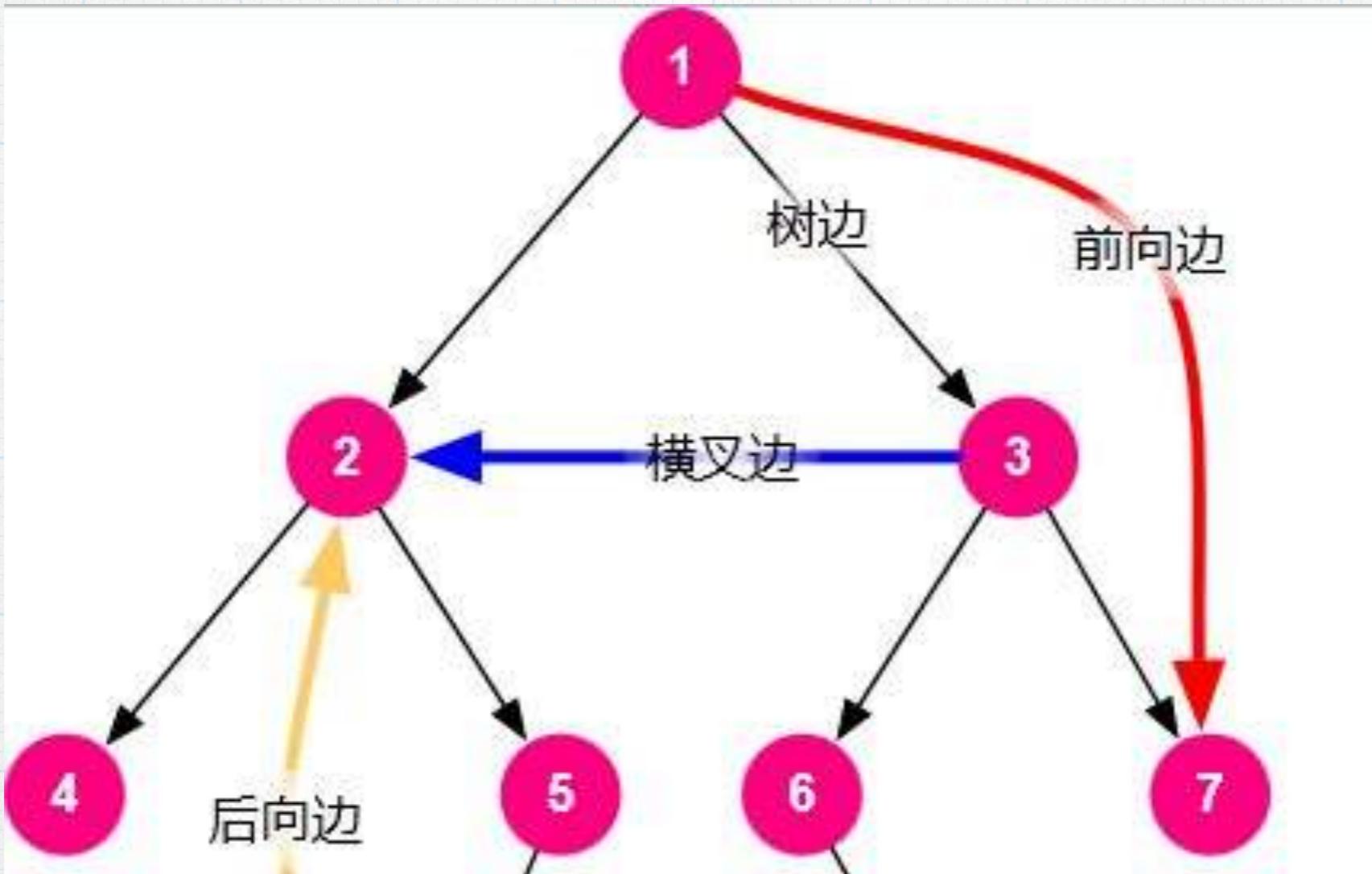
$low = dfn$



信仰坚定  
环环相扣

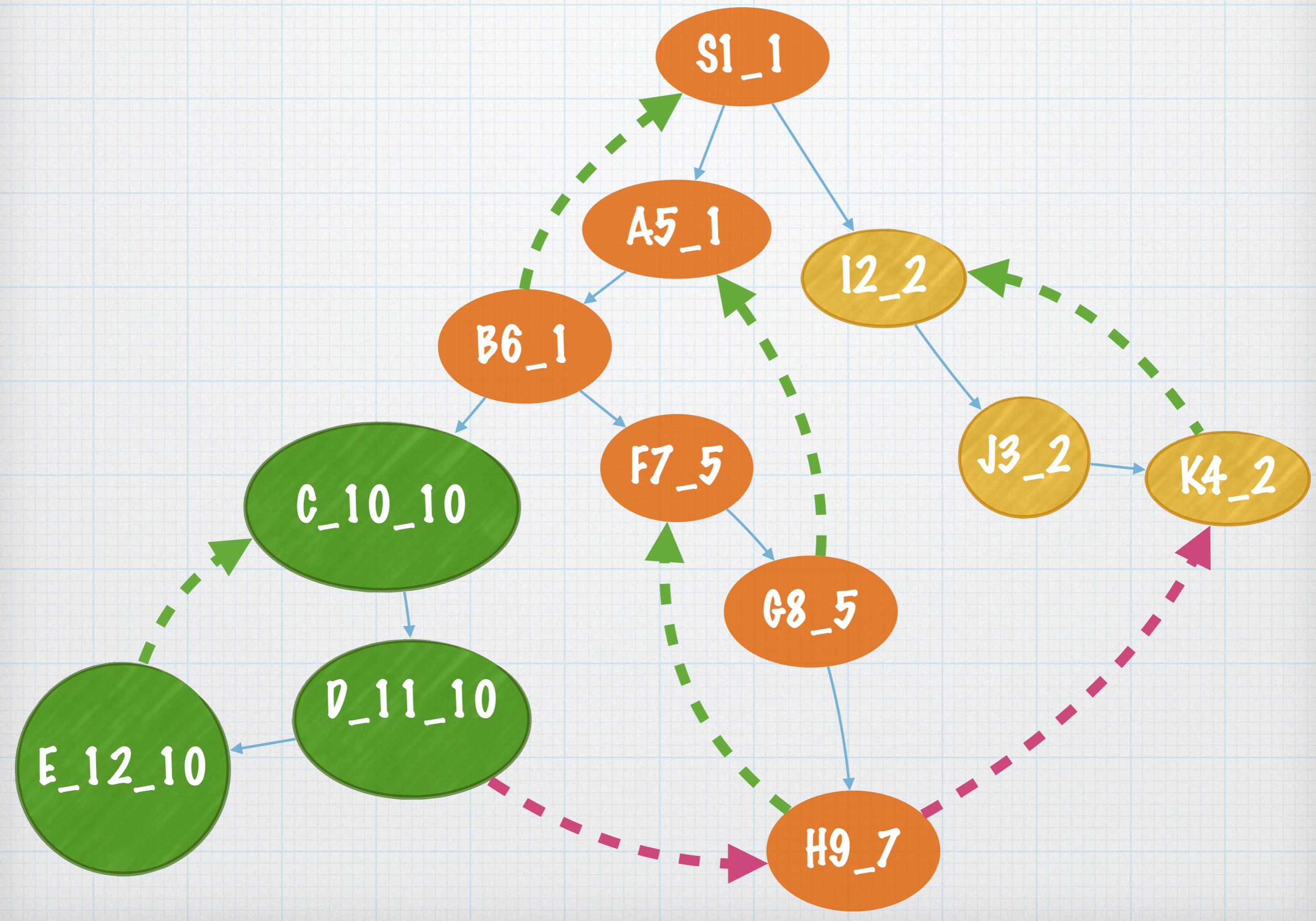
# 强连通

- \* 强连通是针对有向图而言的
- \* 强连通分量(**Strong Connected components**) 内部两两之间都有一条通路( $s \rightarrow t, t \rightarrow s$ )
- \* 强连通分量（极大强连通子图）缩点之后会构成一副有向无环图 (**DAG**)
- \* 四种边：树边，返祖边（后向边），前向边，横插边



# 强连通分量求法

- \* 基本类似于边双的求法，用一个栈保存搜到的点，更新**low**值的时候，如果相邻的点已经访问过，这个点一定要在栈里面，不在栈里面的已经访问过的点是已经与别的点构成了强连通分量的点，比如H->K的边
- \* 性质1：当 $\text{low}[x] == \text{dfn}[x]$ 的时候执行出栈操作，将x能搜到的点都退栈，他们构成了一个强连通分量，它们通过返祖边或者横叉边能直接或者间接的到达x
- \* 性质2：但是每个强连通分量都会在最高点求出来
- \* 性质3：如果**low**值是通过横叉边更新的，说明通过横叉边可以到**LCA**再到自己，形成一个环



# dfs树与随机数

- \* 给出一个无向图，每次询问删掉给定的**k**条边后图是否连通
- \*  $n \leq 100000, m \leq 500000, q \leq 50000, k \leq 15$

- \* 构造dfs树，非树边rand一个值
- \* 树边的值等于所有覆盖它的非树边的值的xor
- \* 问题等价于是否存在一个子集的xor和为0
- \* 同样的思路可以用来求割边的数量

# Dijkstra

- \* 类似于prim，将节点分成两类，已确定最短距离节点与未确定点。每次从未确定点集中选取距离值最小的一个点加入到最短路点集中，因为这个点的最短路径不可能再被更新了，然后用这个点的最短路径去更新邻接点的最短路径

# djikstra+heap

- \* 用一个堆维护距离的最小值，每次从堆中取出一个最小的值，如果这个值大于当前真实的距离，就不要，因为有可能被放入堆之后，又被别的点更新了最短路，当确定是最短路后，就拿这个点的最短路去更新相邻点，把它们的距离值放入堆中，因此一个点会入堆多次，入堆总次数为边的数量，复杂度应该为 $m\log m$

# floyd

- \* 经典的三个循环
- \* 考虑某一条最短路的形成过程，当路径上的点被一个一个考虑进来的时候，会形成一段段区间，每一段区间内部的两两之间的最短路都已经求出，每加入一个点会连通两个相邻的区间。左区间到右区间的点之间的最短路会在此时正式形成
- \* 所以外层的循环random\_shuffle一下也不会影响正确性

# 无向图最小环

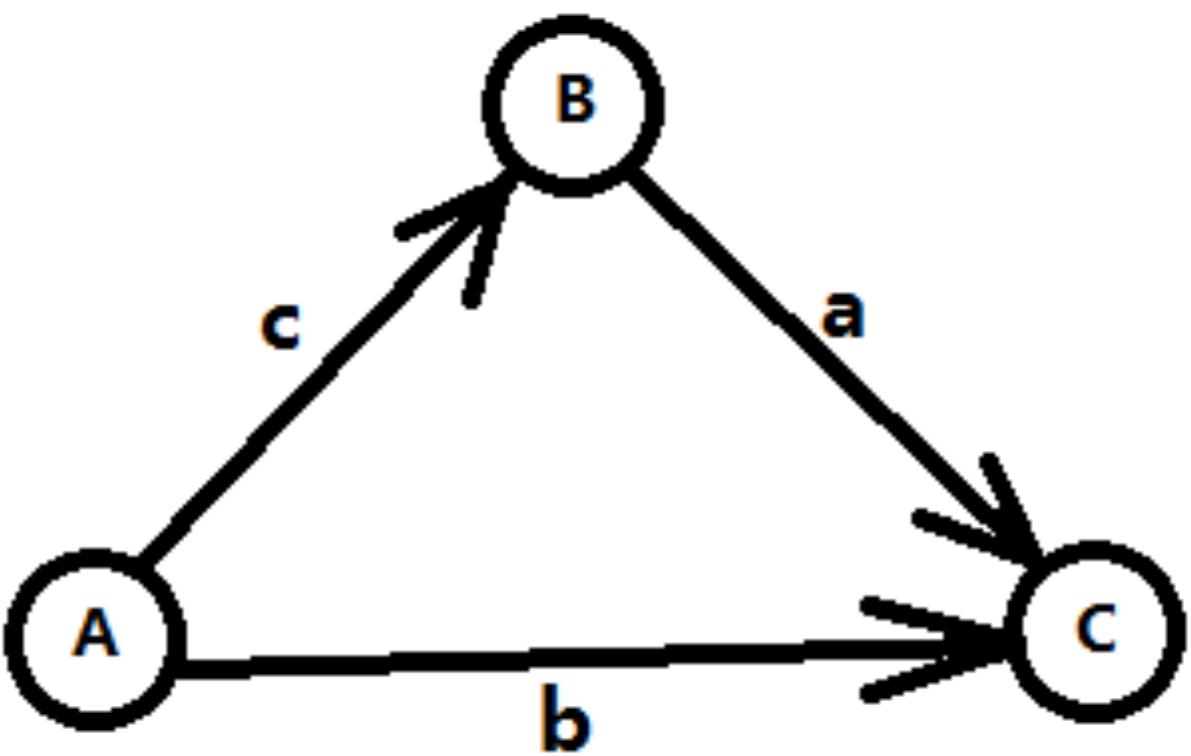
- \* 一个环可以拆成一条最短路加上两条相邻的边的形式
- \* 因此可以在用当前点  $k$  去更新最短路之前先把  $i \rightarrow k$   $k \rightarrow j$  两条边当作环上的两条边去更新最小环
- \* 记录  $\text{pre}[i][j]$  表示  $i$  到  $j$  最短路上的第一个点
- \* 最小环上的点无论以什么顺序考虑进来，在考虑到还差一个点  $k$  的时候（假设环上与  $k$  相邻的点为  $a, b$ ），其他的点肯定无法更新  $ab$  之间的最短路了，不然就会存在更小环

# bellman-ford/spfa

- \* 一个节点的最短距离值最多被更新 $n-1$ 次，所以每一轮都枚举所有边去更新最短路 $dis[i] + w[i][j] \rightarrow dis[j]$ ， $n-1$ 轮之后所有的点的最短路都能更新完毕，如果第 $n$ 轮还能继续更新，说明存在负环。
- \* spfa是bellman-ford的队列优化，每次将刚刚被更新的点放进队列，优先去更新别人，知道队列中元素为空为止。

# 差分约束

- \*  $B - A \leq c$
- \*  $C - B \leq a$
- \*  $C - A \leq b$
- \* 求  $C - A$  的最大值
- \* 答案为  $\min(a + c, b)$



# 拓扑排序

- \* 不断的从入度为零的点开始bfs，搜到的点的度数减1，如果bfs结束后有的点度数不为0，说明有环

# 字典序最小的最短路

- \* 建好最短路图之后直接从起点开始选择编号小的

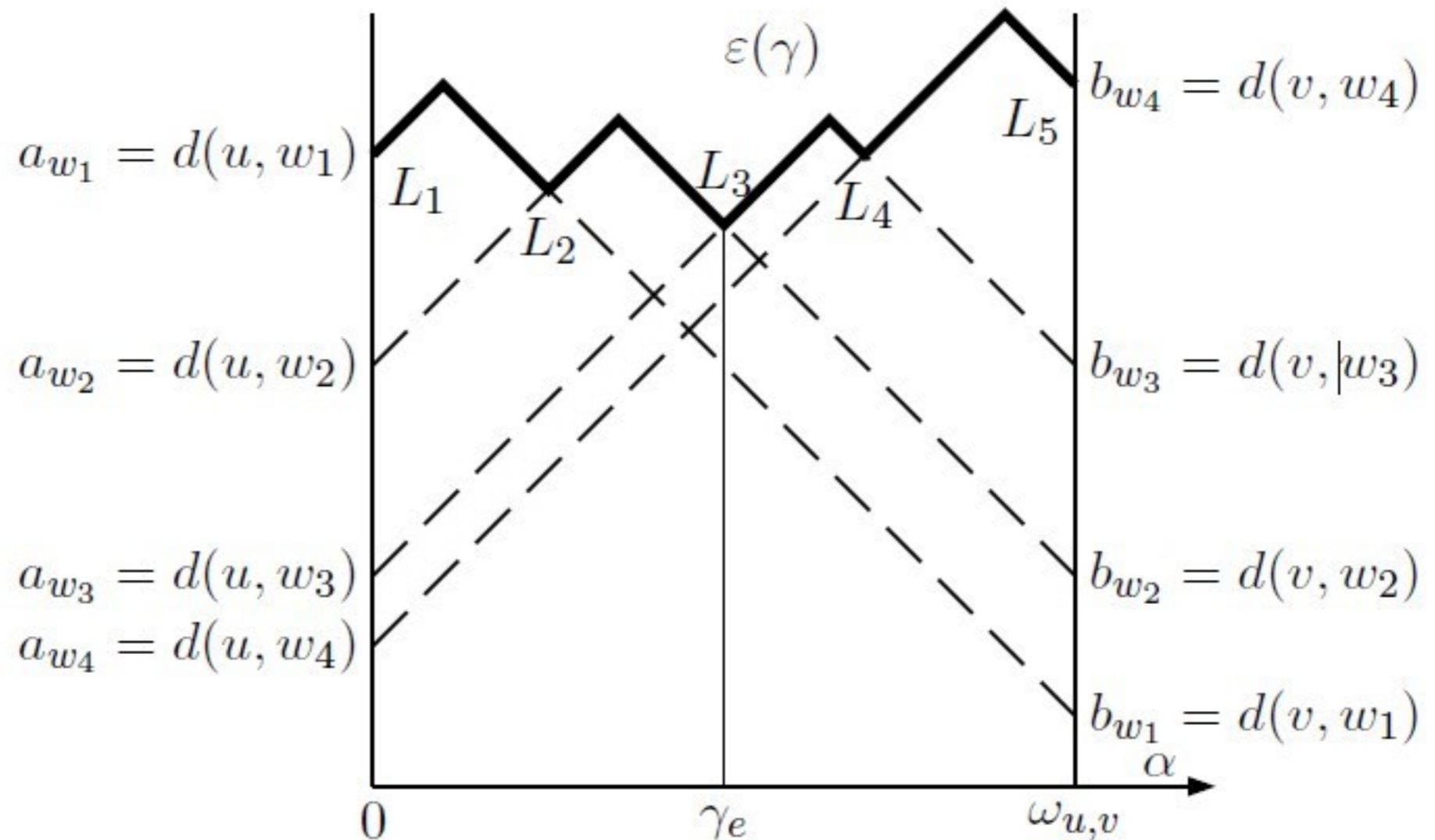
# 图的绝对中心

- \* 这个中心点可以存在与某一条边上，  
这个点到所有点的最短距离的最大值  
最小

\* 绝对中心到所有点的最短距离的最大值肯定会有两个，而且这两个最短距离位于某条边的两端

\* 反证法

\* 枚举每一条边-v, 假设中心在这条边上, 那么中心到某个点s的距离可以表示为 $\min(d[u][s] + x, d[v][s] + l - x)$ , 数形结合一下发现这个是一条折线, 每一条边的折线取max构成了一个总的折线, 折线的最低点就是答案



# 最短路径树

- \* 每个点记录一个最短路径所在的前驱节点，形成的一棵树

# 最小直径生成树MDST

- \* 求一棵生成树，树上最近的两个点距离最小
- \* 先求好绝对中心（所在的边上的两个端点 $u$ - $v$ ），然后从绝对中心出发，位了保证这条边一定在生成树上，一开始应该设设 $d[u]$ ,  $d[v]$ 为真实的值（double），然后搜出一颗最短路径树就是答案

