LCT

- LCT 实际上是对于一颗**无根树**能够动态修改该树的根，并动态维护树上的链或连通性的树上迄今为止最强的工具
- 思路：$Splay$+实链剖分
- 实质：LCT是一颗$Splay$，但在这颗$Splay$上存在实边与虚边的区别，利用虚边将一颗$Splay$分成若干个，每个$Splay$维护一条实链，即：虚边其实就是实链与实链的连接边。
- **一个重要规定:**
  - **我们在这里规定我们所使用的$Splay$以动态树上当前选定的根到它的深度为关键字，故在$Splay$中$ch[x][0]$是x在当前动态树上的父亲($ch[x][0]$ 的关键字$<x$ 的关键字)，而$ch[x][1]$则为x在当前动态树上的儿子**
- 定义:
  - 实链 是一条从上到下按在原树中深度严格递增的路径，且对该实链的$Splay$做中序遍历后得到的序列每个点的深度严格递增。
  - $f[x]$ 是x节点在大$Splay$上的(父亲)，$ch[x][0,1]$ 表示在各个$Splay$上x的左/右儿子

        - 则当 $f[x] = ch[f[x]][0,1]$ 为**实边**，反之当 $f[x] \neq ch[f[x]][0,1]$ 为**虚边**
        - 作用：在整颗$Splay$上对小的$Splay$进行旋转或维护权值时通过 **实边** 知道节点在节点在小的$Splay$上，而通过 **虚边** 知道节点不再这个小的$Splay$上
- LCT 的一些基本函数：
    - [ ] 利用 $access(x)$（将x所在实链与x到$root$上所有实链合并成一颗$Splay$，即让x与$root$在同一颗小$Splay$上）确保实链剖分不会一直剖分到每条链只有一个点，保证复杂度
    - [ ] 利用 $splay() + rotate()$ 实现实链上的$Splay$双旋操作
    - [ ] 利用 $makeroot()$（换根操作）改变动态树的根
    - [ ] 利用 $isroot()$ 判断是否一颗小$Splay$的根，即**虚边**
    - [ ] 利用 $findroot()$ 寻找在动态树上当前选定的根
    - [ ] 利用 $split()$ 提取链上信息
    - [ ] 利用 $cut()$ 在动态树上删去一条边
    - [ ] 利用 $link()$ 在动态树上添加一条边
- LCT 基本函数的实现：
    - $access(x)$
        - LCT 上最为重要，但较难理解的一个函数

- 首先我们先考虑两条实链该如何合并
  - 上文我们提到了对于LCT中的$Splay$的关键字规定，运用该规定不难想到，对于两个实链，若要让它们上面所有点都合并到一条实链上 大多数情况下不太可能（若可以，则该实链的深度序列不能保证严格递增）
  - 那我们应该如何来做呢?
    - ~~上文其实有剧透~~
    - 我们可以通过将实边改成虚边（即：将部分让该实链的深度序列非严格递增的点踢出实链，让它们自成实链）让该实链合法化
  - 这样我们就解决了让两条链合并的操作
    - 简单来说就是：将一条整链提取出来，再把上面挂的多余儿子剔除，让他们自成一体
- 多次重复上述操作，我们就可以合并多条实链了
- 具体实现:
  - 以$u \to root$为例
  - 开始时，当前要做的节点$x = u$，已经连接好的实链$y = 0$
  - 先找到x所在的小$Splay$，将x旋转到根(注意：和$Splay$一样$f[], ch[][0/1]$也要随$rotate()$操作一起修改)，利用$Splay$上有节点深度比

自己大，将x的右儿子舍弃(即：将该实边改为虚边)，再将已连接好的实链$u \rightarrow y$连到自己上(即$ch[x][1]=y$)
        - 再沿虚边(实链间的连边)将$x \rightarrow$ 上面一条实链上的点(即 $x=f[x]$),同时将已连接好的实链更新到x (即 $y=x$)
        - 重复该上述操作直到 $y=root$
- $splay(x)$
    - 与普通$Splay$不一样的是在LCT上，所有$splay()$操作都是旋转到根，~~其余的都与普通$Splay$一样~~
- $rotate(x)$
    - ~~与普通$Splay$大致一样~~，注意 要修改的是$f[],ch[][0/1]$
- $makeroot(x)$
    - 另一个LCT上的重要操作，LCT就是利用这个操作动态修改树根
    - 看上去很高大上，其实理解起来还是比较简单的
    - 思路:
        - 考虑到在大$Splay$上认定的整个树的树根就是$f[root]=0$，那么也就是说我们得先让x在存在整棵树的树根的小$Splay$上，那么我们想到什么? 对，就是 $access(x)$ 操作!
        - 那么接下来，我们该做什么? $root$还有一个很

好的性质：深度最浅的点，即在大$Splay$上最左边的点，而x这是这颗小$Splay$上深度最深的点。那么我们可以把x利用 $splay()$ 操作旋转到根，再将整棵$Splay$反转，那么x就成为深度最小的点

- $isroot(x)$
  - 这个操作在上文多次被提到，也是支撑整个LCT的$Splay$上分裂实链的函数
  - 若 $f[x] == ch[f[x]][0/1]$ 是实边，不是小$Splay$的根；若 $f[x] \neq ch[f[x]][0/1]$ 是虚边，是小$Splay$的根
- $findroot(x)$
  - $\because$ $root$ 为深度最小的点
  - $\therefore$ 先 $access(x)$ 再 $splay(x)$ ，再在该小$Splay$上最左边的点
- $split(u, v)$
  - 首先注意到 $u \to v$ 不一定能够在当前选定根下成为一条实链，因为u, v 间可能有比他们深度都小的Lca
    - 所以先将u定成动态树上的根(即 $makeroot(u)$ ），再 $access(v)$ 这样 u, v 就在一条实链，且这个实链上只包含 u, v 的路径上的点，所以 u, v 的链上信息就是 $dat[v]$ (v上存的值)

- $cut(u,v)$
  - 考虑到(u,v)有可能本身不存在
  - 所以，我们先将u定成动态树的根，再利用 $findroot()$ 操作判断 u,v 是否连通，然后再看大$Splay$上u,v间是否还有点(即 $ch[u][1] \neq 0$) u,v之间是否只有一条边(即 $f[u] \neq v$)
  - 最后判断合法，就把(u,v) 断开
- $link(u,v)$
  - 先将u定成动态树的根($makeroot(u)$)
  - 若(u,v)本身已存在(即，两点连通 $findroot(v) = u$)
  - 若没有该边，因为u现在为深度最小的点，故我们可以将$f[u]$设成v ($f[u] = v$)