左偏树

Leftist Tree

Idea：左偏树是一棵二叉树，不仅具有堆的性质，还是“左偏”的。

对于二叉树每个节点一个距离信息 dis：定义仅有一个子节点或没有子节点的点为外点，外点的 dis 为 0；定义一个内点的距离 dis 为它到其子树中最近的外点的距离，空节点的距离 dis 为 -1。

所谓“左偏”，即在合并过程中需要保证每个节点的左子节点的 dis 恒大于等于右子节点的 dis，于是一个点的 dis 等于其右子节点的 dis 加上 1。

Operations：

- 合并 `Merge(x, y)`：以小根堆为例，选取 x 和 y 中根更小的那个（假设是 x）作为合并后的左偏树的根节点，然后递归合并 x 的右子树和 y ；如果合并后左子树 dis 小于右子树 dis，则交换左右子树。
- 删除根 `Pop(x)`：将左右子树合并起来。
- 插入节点：一个节点也可以视为一棵左偏树，合并即可。

ATT：

- 左偏树并不保证深度，找父节点时应采用路径压缩的方式找。
- 初始化时，`tr[i].fa = i`；非树节点 `dis` 为 -1（特别是 `tr[0].dis = -1`）

Complexity：单次操作 $O(\lg n)$

Code：

小根堆为例。

```
struct LeftistTree{
    int l, r, fa, key, dis;
    LeftistTree(){ dis = -1; }
}tr[N];
int findfa(int x){ return x == tr[x].fa ? x : tr[x].fa = findfa(tr[x].fa); }
int Merge(int x, int y){
    // merge two heaps whose roots are x and y
    // if x,y are not roots, apply x = findfa(x), y = findfa(y) beforehand
    if(!x || !y)    return x + y;
    if(x == y)  return x;
    if(tr[x].key > tr[y].key)   swap(x, y);
    tr[x].r = Merge(tr[x].r, y);
    if(tr[tr[x].l].dis < tr[tr[x].r].dis)   swap(tr[x].l, tr[x].r);
    tr[tr[x].l].fa = tr[tr[x].r].fa = tr[x].fa = x;
    tr[x].dis = tr[tr[x].r].dis + 1;
    return x;
}
int Pop(int x){
    // pop the first element in the heap rooted at x
    // if x is not root, apply x = findfa(x) beforehand
    tr[tr[x].l].fa = tr[x].l, tr[tr[x].r].fa = tr[x].r;
    tr[x].dis = -1;
    tr[x].fa = Merge(tr[x].l, tr[x].r);
    tr[x].l = tr[x].r = 0;
    return tr[x].fa;
}
```