堆来堆去

关于堆的一堆事

重庆南开信竞基础课程 helang

可删堆：盒子小球

有一个容量无限大的空盒子，有若干小球，每个小球上都有一个编号，小球编号互不相同，编号范围$[0,10^9]$。

接下来进行m(1<=m<=200000)次操作，操作有如下三种：

1.查找：查找当前盒中编号最大的小球，并输出它的编号；

2.添加：往盒中添加一个编号为x的小球；

3.删除：删除盒中编号为y的小球；

可删堆

将盒子看作一个大根堆s

再开一个删除堆（大根堆）s'，从s删除一个小球的时候将要删除小球的编号加入到s'里面。

获取s堆顶元素：如果s和s'的堆顶元素相同就同时pop掉，直到两个堆顶不同，此时s的堆顶就是真正的堆顶元素。

时间复杂度O(mlogm)

可删堆


```
struct heap
{
   priority_queue<int> a,b;       //b为a的删除堆
   void Insert(int x){ a.push(x); }
   void Delete(int x){ b.push(x); }
   int getSize(){ return a.size()-b.size(); }
   int getTop()
   {
     if(!getSize()) return -1;
     while (!b.empty() && a.top() == b.top())
              a.pop(),b.pop();
     return a.top();
   }
};
```

对顶堆：动态中位数 NKOJ7350

依次读入一个长度为n的整数序列A，每当已经读入的整数个数为奇数时，输出已读入的整数构成的序列的中位数。

$Ai<=10^9,n<=100000$

对顶堆：动态中位数 NKOJ7350

声明一个大根堆和一个小根堆

大根堆存较小的数字们，堆顶为较小数字中的最大者

小根堆存较大的数字们，堆顶为较大数字中的最小者

依次入读数字，对于读入的数字：

将大于大根堆堆顶的数放入小根堆

将小于小根堆堆顶的数放入大根堆

若两堆元素个数差<=1，则元素个数较多的堆的堆顶即为当前中位数

若两堆元素个数差>1，则将元素较多的堆的堆顶元素取出加入元素较少那个堆即可。

每次增加2个数字，每次操作最多2个数字，时间复杂度O(nlogn)

引例：猴子大王 NKOJ2260

有N只猴子，每只猴子都有一定的强壮值。开始时，所有猴子互不相识（每只猴子位于单独的猴群）。接下来发生了M次矛盾。

不在同一猴群的两只猴子如果产生了矛盾，他们会邀请各自猴群里，最强壮的一只(可能是自己)出来为自己撑腰。被请出来的那两只猴子会进行搏斗，强壮值大的那只猴子一定会赢，搏斗结束后他俩的强壮值都会减半(例如10会减为5，5会减为2）。强壮值较小的那个猴子所在的猴群会诚服于胜者，加入胜者所在猴群。

给出M次矛盾的两只猴子的信息，如果两只猴子是不同族群，那么输出搏斗后后两只猴子的所在猴群里最强壮的猴子的强壮值，否则输出-1。

N,M<=100,000

概念：可并堆

可并堆(Mergeable Heap)是一种抽象数据类型，它除了支持堆(优先队列)的三个基本操作(插入，取最值，删最值)，还支持一个额外的操作—合并。

常见的可并堆:

- **左偏树(Leftist Tree)**
- **斜堆(Skew Heap)**
- 二项堆(Binomial Heap)
- 配对堆(Pairing Heap)
- 斐波那契堆(Fibonacci Heap)

以上几种堆的合并操作(Merge)的时间复杂度是O(logn)或O(1)

左偏树

左偏树(Leftist Tree)是一种可并堆(Mergeable Heap)

它支持优先队列的三个基本操作（插入，删最值，取最值），还支持合并操作。

左偏树是一棵堆有序(Heap Ordered)二叉树。

左偏树满足左偏性质(Leftist Property)。

左偏树的每个结点的左子树和右子树都是左偏树。

左偏树的性质

重庆南开信竞基础课程

左偏树中，**外节点**(External Node)为左子树或右子树为空的节点(下图中紫色点)。

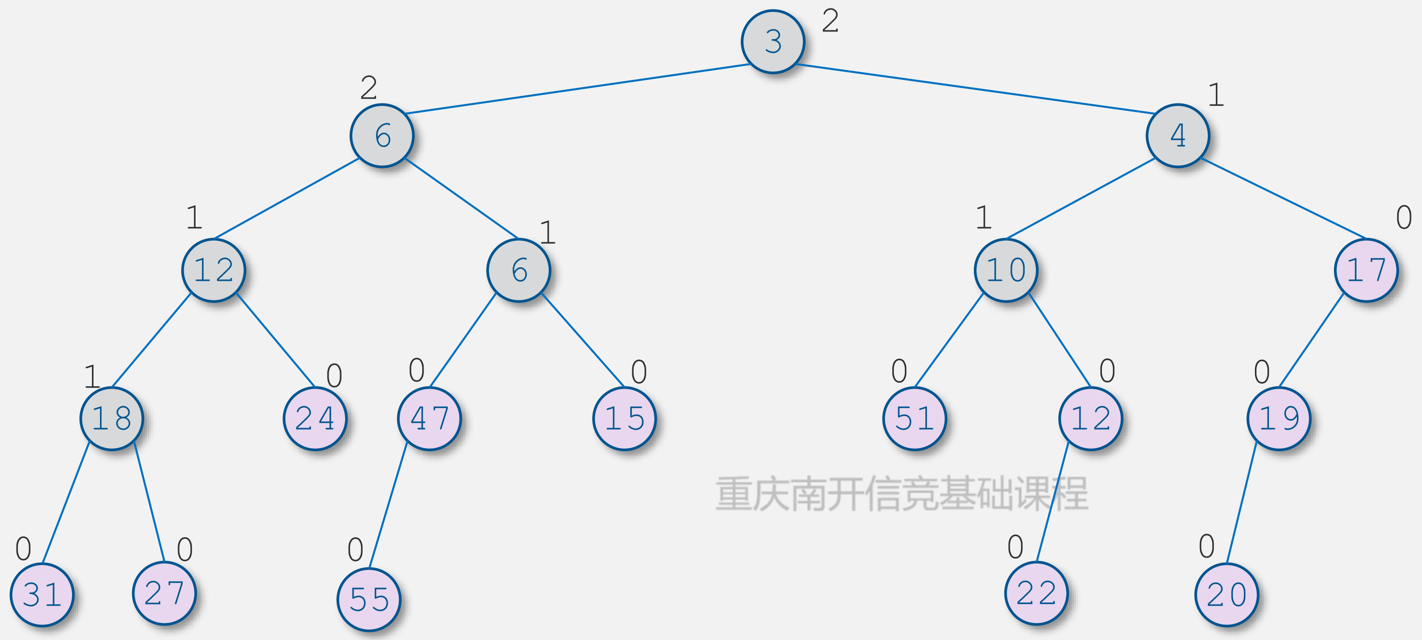
左偏树中，节点i的**空距**(disNull(i))为节点i到它的子树中最近一个外节点所经过的边数。

左偏树中，**任意节点的左儿子的空距不小于右儿子的空距**（左偏性质）。

由左偏性质可知，一个节点的空距等于以该节点为根的子树**最右路径**的长度(最右路径即该结点一直向右儿子走，到达外节点的距离)。

disNull(i)=disNull(Right[i])+1　//可以认为空节点的空距为-1

重庆南开信竞基础课程
重庆南开信竞基础课程



重庆南开信竞基础课程
重庆南开信竞基础课程

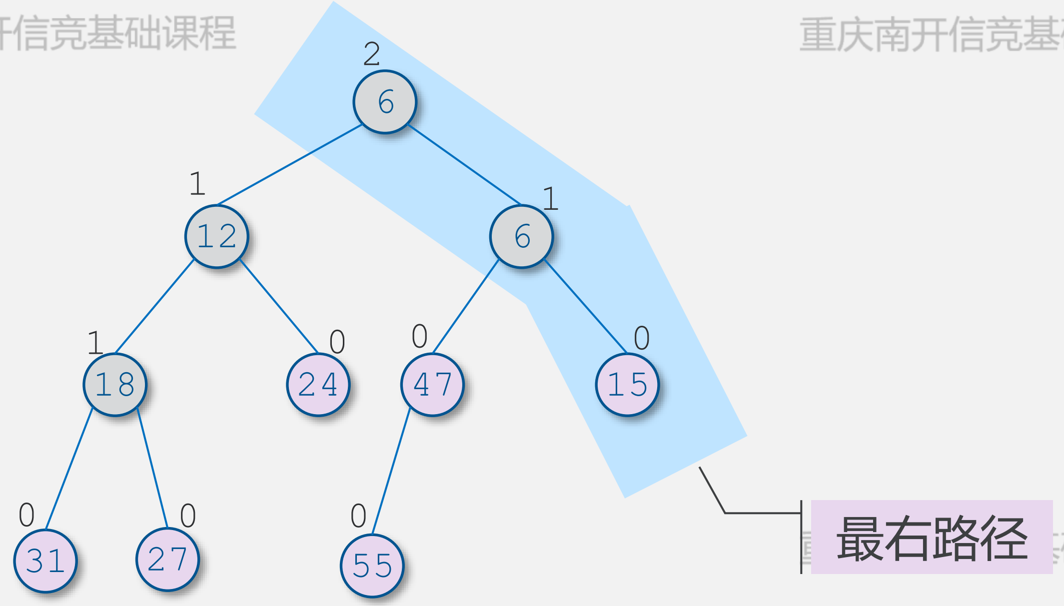
左偏树的性质

性质1：最右路径长度为k的左偏树必然至少有$2^{k+1}-1$个节点

一棵最右路径为k的左偏树，至少是一棵有k+1层的满二叉树。k+1层满二叉树的节点个数为$2^{k+1}-1$。所以节点数至少为$2^{k+1}-1$

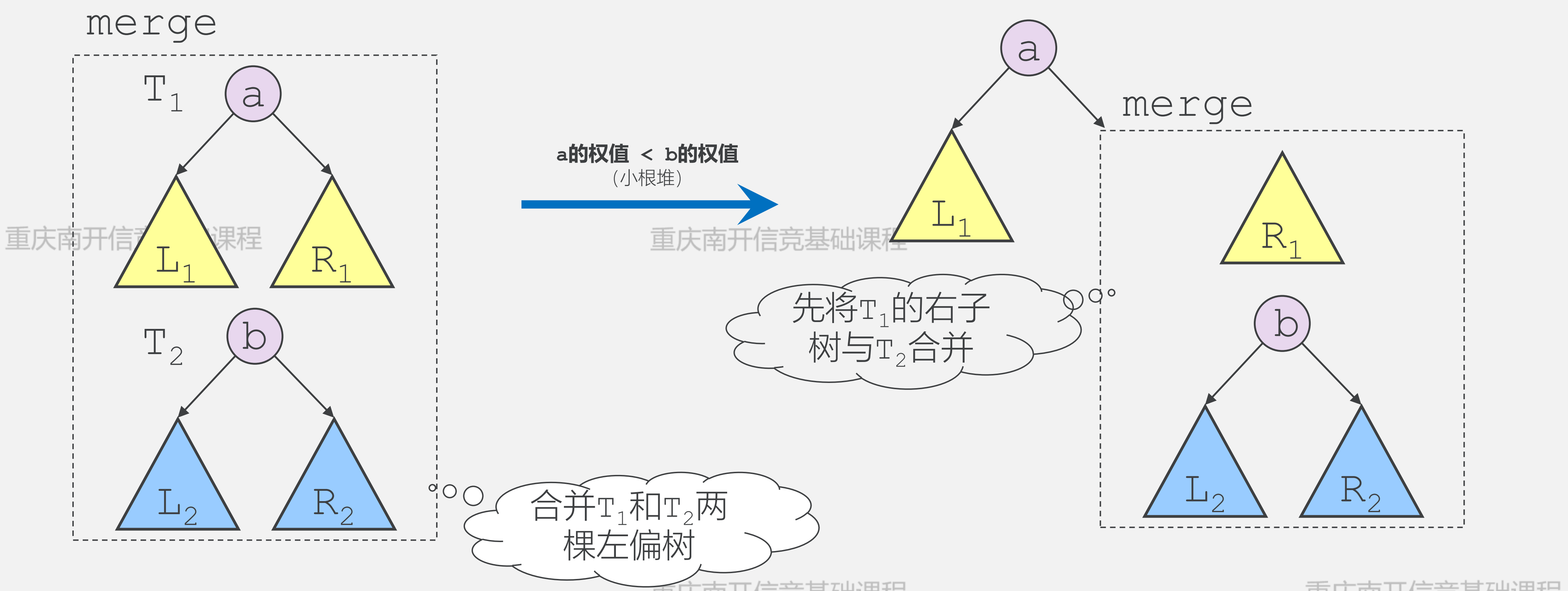
性质2：一棵左偏树有N个节点，则该左偏树的最右路径长度不超过$\lfloor \log^{(N+1)}-1 \rfloor$。

设路径长度为k,有性质1可知$N>=2^{k+1}-1$,则$k<=\log^{(N+1)}-1$



左偏树的关键操作——合并(小根堆)

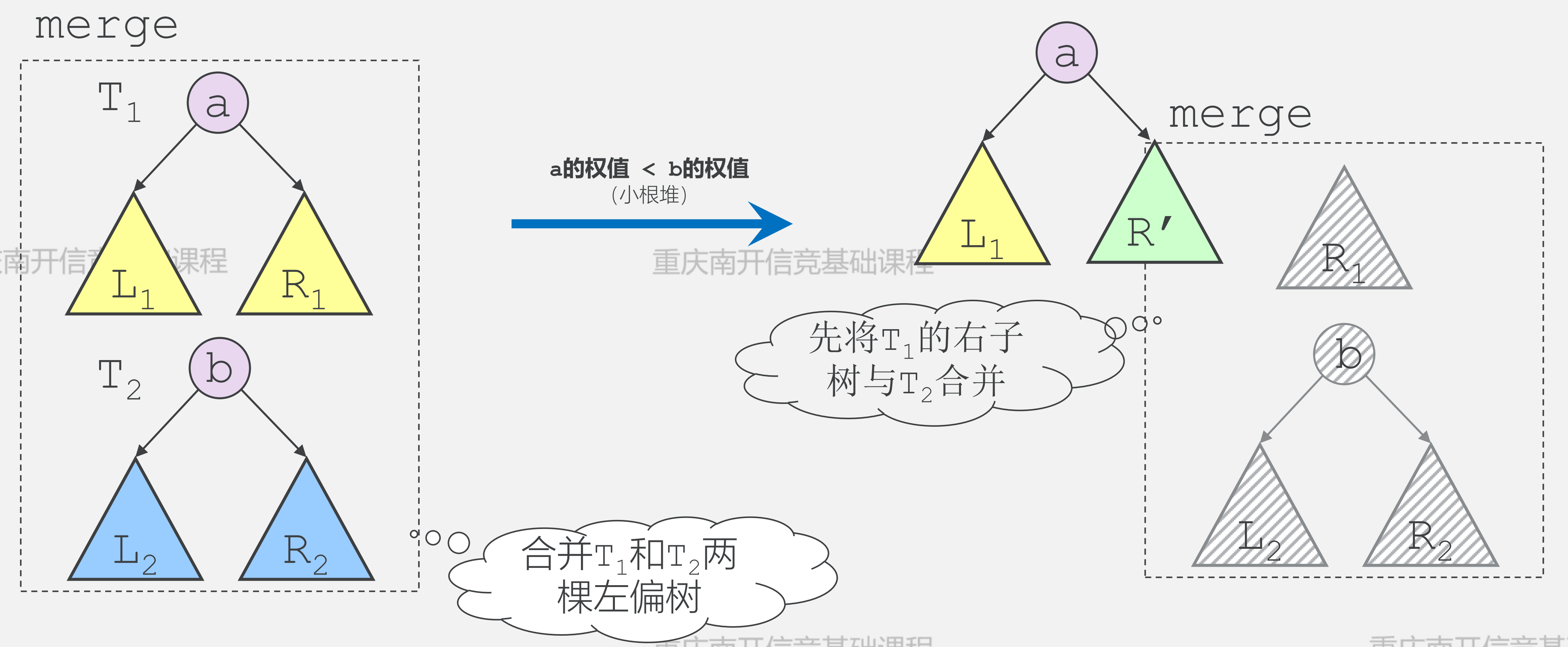
合并操作是递归进行的



左偏树的关键操作——合并

重庆南开信竞基础课程

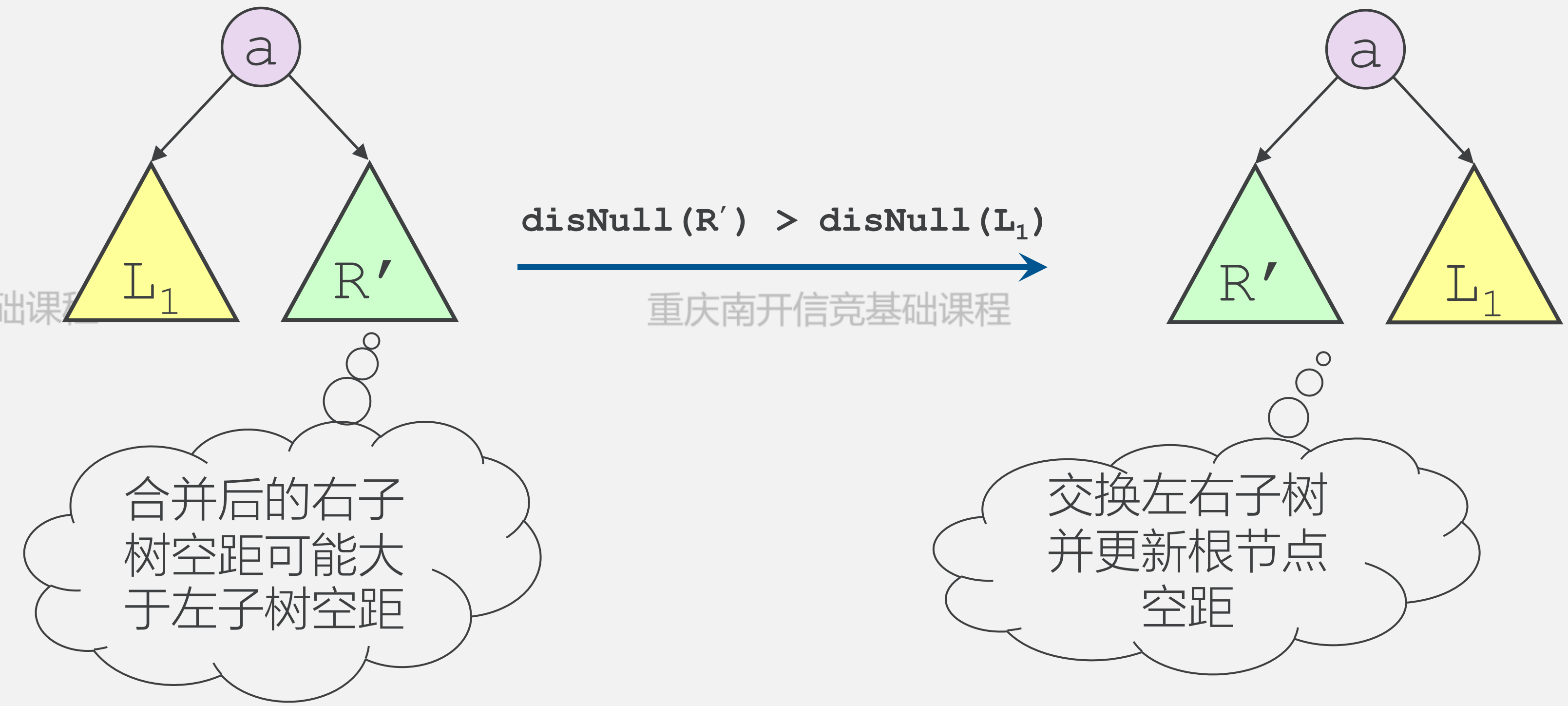
合并操作是递归进行的



重庆南开信竞基础课程
重庆南开信竞基础课程

左偏树的关键操作——合并

合并操作是递归进行的



左偏树的关键操作——合并

重庆南开信竞基础课程

合并操作参考代码：

```
int Merge(int A, int B)
{
    if(A==0)return B;
    if(B==0)return A;
    if(Value[B]<Value[A])swap(A, B);
    Right[A]=Merge(Right[A], B);
    if(disNull[Right[A]]>disNull[Left[A]])swap(Left[A],Right[A]);
    if(Right[A]==0)disNull[A]=0;
    else disNull[A]=disNull[Right[A]]+1;
    return A;
}
```

合并操作都是一直沿着两棵左偏树的最右路径进行的。

一棵N个节点的左偏树，最右路径上最多有 $\lfloor \log(N+1) \rfloor$个节点。

因此，合并节点数为N1和N2的两棵树的操作的时间复杂度为：O(logN1 + logN2)=O(logN)

重庆南开信竞基础课程
重庆南开信竞基础课程

左偏树操作：插入一个新节点

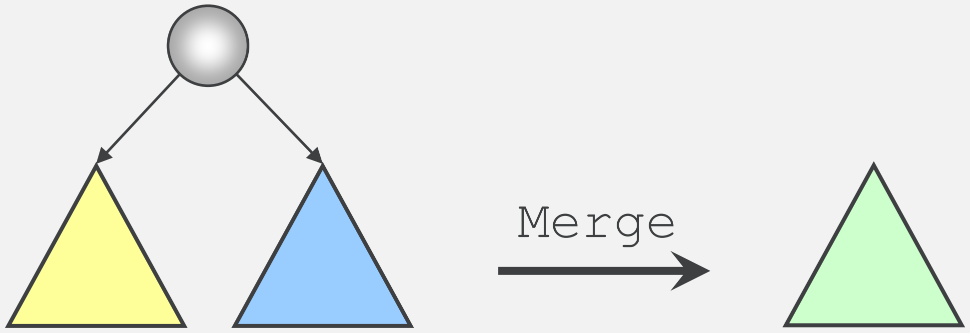
Insert —— 插入一个新节点

- 把待插入节点作为一棵单节点左偏树
- 合并两棵左偏树
- 时间复杂度：O(logN)

左偏树操作：删除最小(大)节点

删除最小节点

- 删除根节点
- 合并左右子树
- 时间复杂度：O(logN)



左偏树操作：删除一个已知(位置)节点

这里的已知，是已知这个节点的在左偏树中的位置，而不是值。

删除一个已知节点不可能像删除最小节点那样简单，因为合并他的左右子树后，新的子树的距离有可能会改变，这样会引起一串连锁反应。所以，将左右子树合并之后，需要一直沿着树链向上调整。

合并的复杂度是O(logN)，向上最多O(logN)次。所以删除一个已知节点的复杂度是O(logN)。

建议操作：没必要真的删除，标记一下，待它到根时再删除即可。因为一般来说只会在根用到它。

左偏树操作：建树

重庆南开信竞基础课程

将N个节点构成一棵左偏树。利用插入操作，一个个插入。O(NlogN)

也可以用以下几个步骤：

1. 建立一个队列，将每个节点看作一个节点数为1的左偏树加入队列。
2. 每次取出队头的两棵左偏树，将它们合并，并将合并后的新左偏树加入队列。
3. 重复第2步，直到队列为空。

```
void Build()
{
    queue<int> q;
    for(int i=1;i<=n;i++)q.push(i);
    int x,y;
    while(q.size())
    {
        x=q.front();q.pop();
        y=q.front();q.pop();
        q.push(Merge(x,y));
    }
}
```

时间复杂度为O(n)，证明：设$n=2^k$

前$n/2$次合并的是两棵有1个节点的左偏树；

接下来的$n/4$次合并的是两棵有2个节点的左偏树；

接下来的$n/8$次合并的是两棵4个节点的左偏树；

……

接下来的$n/2^i$次合并的是两棵2^{i-1}个节点的左偏树；

合并两棵2^i的左偏树的时间为$O(\log_2 2^i)=O(i)$

总的时间消耗：

$=(n/2)*O(1) + (n/4)*O(2) + (n/8)*O(3)+\ldots+(n/2^k)*O(k)$

$=O(\sum(n*i/2^i))$ 条件 $1<=i<=k$

$=O(n*\sum(i/2^i)) = O(n*(2-(k+2)/(2^k)))=O(n)$

左偏树操作：建树

$\frac{n}{2}*O(1)+\frac{n}{4}*O(2)+\frac{n}{8}*O(3)+\cdots\ldots+\frac{n}{2^k}*O(k)=O\left(\sum\left(n*\frac{i}{2^i}\right)\right)=O\left(n*\sum\left(\frac{i}{2^i}\right)\right)=\ O\left(n*\left(2-\frac{k+2}{2^k}\right)\right)=O(n)$　条件 $1<=i<=k$

$$s1=\frac{1}{2}+\frac{2}{4}+\frac{3}{8}+\cdots+\frac{k}{2^k}$$

令$s2=2*s1=2*\left(\frac{1}{2}+\frac{2}{4}+\frac{3}{8}+\cdots+\frac{k}{2^k}\right)=1+\frac{2}{2}+\frac{3}{4}+\cdots+\frac{k}{2^{k-1}}$

$$s1=s2-s1=1+\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\cdots+\frac{1}{2^{k-1}}-\frac{k}{2^k}$$

$$s3=1+\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\cdots+\frac{1}{2^{k-1}}$$

令$s4=2*s3=2*\left(1+\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\cdots+\frac{1}{2^{k-1}}\right)=2+1+\frac{1}{2}+\frac{1}{4}+\cdots+\frac{1}{2^{k-2}}$

$$s3=s4-s3=2-\frac{1}{2^{k-1}}$$

$$s1=s3-\frac{k}{2^k}=2-\frac{k+2}{2^k}<2$$

左偏树操作：建树

$\frac{n}{2} * O(1) + \frac{n}{4} * O(2) + \frac{n}{8} * O(3) + \cdots \ldots + \frac{n}{2^k} * O(k) = O\left(\sum\left(n * \frac{i}{2^i}\right)\right) = O\left(n * \sum\left(\frac{i}{2^i}\right)\right) = O\left(n * \left(2 - \frac{k+2}{2^k}\right)\right) = O(n)$ 条件 $1 <= i <= k$

$$s1 = \frac{1}{2} + \frac{2}{4} + \frac{3}{8} + \cdots + \frac{k}{2^k}$$

令$s2 = 2 * s1 = 2 * \left(\frac{1}{2} + \frac{2}{4} + \frac{3}{8} + \cdots + \frac{k}{2^k}\right) = 1 + \frac{2}{2} + \frac{3}{4} + \cdots + \frac{k}{2^{k-1}}$

$$s1 = s2 - s1 = 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots + \frac{1}{2^{k-1}} - \frac{k}{2^k}$$

$$s3 = 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots + \frac{1}{2^{k-1}}$$

令$s4 = 2 * s3 = 2 * \left(1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots + \frac{1}{2^{k-1}}\right) = 2 + 1 + \frac{1}{2} + \frac{1}{4} + \cdots + \frac{1}{2^{k-2}}$

$$s3 = s4 - s3 = 2 - \frac{1}{2^{k-1}}$$

$$s1 = s3 - \frac{k}{2^k} = 2 - \frac{k+2}{2^k} < 2$$

左偏树小结

操作	左偏树
取最小节点	O(1)
插入	O(logN)
删除最小节点	O(logN)
合并	O(logN)

左偏树的特点：

- 时空效率高
- 编程复杂度低

左偏树的应用：

- 可并堆
- 优先队列

斜堆

重庆南开信竞基础课程

斜堆(Skew Heap)是一种可并堆(Mergeable Heap)

它支持优先队列的三个基本操作（插入，删最值，取最值），还支持合并操作。

斜堆是一棵堆有序(Heap Ordered)二叉树。

斜堆的每个结点的左子树和右子树都是斜堆。

重庆南开信竞基础课程
重庆南开信竞基础课程

斜堆(Skew Heap)是精减版的左偏树。但是它不满足左偏性质。斜堆根本就没有“距离”这个概念——它不需要记录任何一个节点的空距。

重庆南开信竞基础课程
重庆南开信竞基础课程

斜堆的合并操作

合并操作： merge(A, B)

将以A，B为根结点的两个斜堆合并，再返回合并后的新斜堆的根结点。

以小根堆为例：

假设A.key <= B.key 就将A的右子树与B合并（递归进行）当做A的新的右子树。

然后直接交换A的左右子树。

跟左偏树一样，其他插入、删除等操作都是建立在合并操作的基础上。

斜堆的合并操作


```
int Merge(int A,int B)
{
    if(A == 0) return B;
    if(B == 0) return A;
    if(Value[A] > Value[B]) swap(A,B);
    Right[A] = Merge(Right[A],B);
    swap(Right[A],Left[A]);
    return A;
}
```

斜堆和左偏树的合并操作都是沿着最右路径进行，经过合并后，新堆的最右路径长度必然增加，这会影响后续合并操作的效率。

左偏树在进行合并时，会检查最右节点的空距是否超过左节点，并通过交换左右子树，使整棵树的最右路径长度维持在很小的规模。

斜堆不记录节点的距离，采取简单直接的办法：从下往上，沿着合并的路径，在每个节点处都交换左右子树。

由于没有对斜堆的右子树的深度做限制，因此最坏情况下复杂度为O(N)

但是，其均摊复杂度为O(logN)，且有证明不超过为O(4logN)

具体证明参见《Data Structure and Problem Solving Using Java Second Edition》（Mark Allen Weiss 著，电子工业出版社出版）中的784页的Theorem 23.2

第3节：典型例题

习题：sequence NKOJ8989

给定一个整数序列 $A_1, A_2, \ldots, A_n$，
求一个**不下降**序列 $B_1 \le B_2 \le \ldots \le B_n$，
使得下列式子的值最小：

$$|A_1 - B_1| + |A_2 - B_2| + \ldots + |A_n - B_n|$$

数据规模：$1 \le n \le 10^6$，$0 \le A_i \le 2*10^9$

习题：sequence NKOJ8989

简化问题：

问题1：只求一个数字B，使得式子的值最小

$|A_1-B| + |A_2-B| + \dots + |A_n-B|$

显然B是数列A的中位数

问题2：求两个数字$B_1 \le B_2$，使得式子的值最小

$|A_1-B_1| + |A_2-B_1| + \dots + |A_k-B_1| + |A_{k+1}-B_2| + |A_{k+2}-B_2| + \dots + |A_n-B_2|$

将数列A分为[1,k]和[k+1,n]两段，B_1为[1,k]的中位数，B_2为[k+1,n]的中位数

问题3：求m个数字$B_1 \le B_2 \le \dots \le B_m$，使得式子的值最小

$|A_1-B_1| + |A_2-B_1| + \dots + |A_k-B_i| + |A_{k+1}-B_i| + \dots + |A_n-B_m|$

将数列A分为m段，B_i为第i段的中位数

习题：sequence NKOJ8989

解题分析：

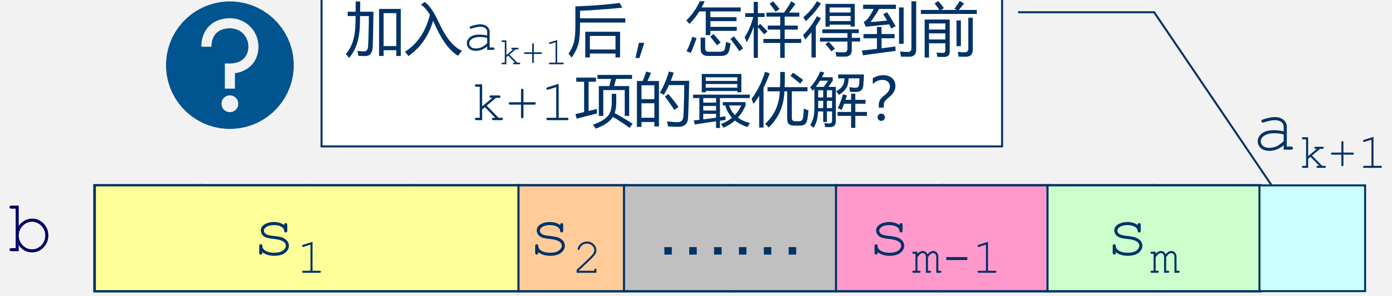
假设数列 $A_1, A_2, \ldots, A_k$ 的最优解为 $B_1 \le B_2 \le \ldots \le B_k$

合并$\{B_i\}$中相同的项，得到m个数 $S_1 < S_2 < \ldots < S_m$

m个数字对应了将A数列前k项划分成m段

S_i为数列A中在第i个段内的数字的中位数。

加入a_{k+1}后，怎样得到前k+1项的最优解？



习题：sequence NKOJ8989

解题分析：

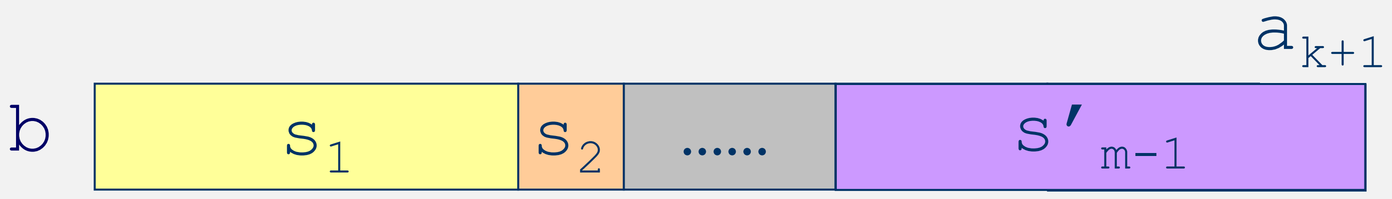
若$a_{k+1}>s_m$，

直接令$s_{m+1}=a_{k+1}$，得到前$k+1$项的最优解；

否则，将a_{k+1}并入第m个区间，并更新s_m

不断检查最后两个区间的解s_{m-1}和s_m，

若$s_{m-1}\ge s_m$，合并最后两个区间，并令新区间的解为该区间内的中位数。



习题：sequence NKOJ8989

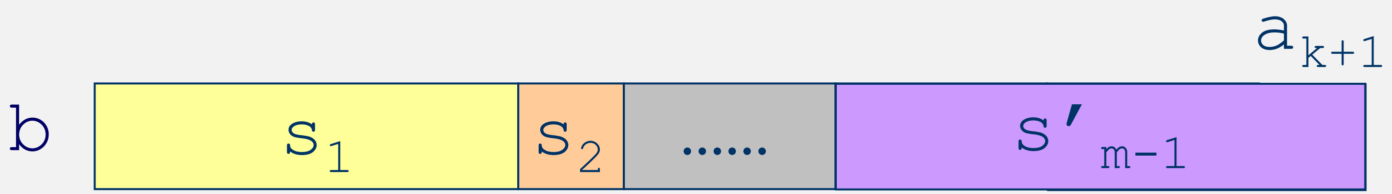
具体操作:

为每个s_i建立一个可并堆，当前A的前k项已经分成了m段。

读入a_{k+1}，令$s_{m+1}=a_{k+1}$，即将新读入的数字当作只有一个节点的堆，m++；

不断检查最后两个区间，反复执行下列操作:

1. 若$s_m>s_{m-1}$，已得到前k+1项的最优解，跳出；
2. 若$s_{m-1}\ge s_m$，将s_m并入第s_{m-1}对应的集合，并更新该集合的中位数s_{m-1}，m--；



习题：sequence NKOJ8989

具体操作：

为每个s_i建立一个可并堆，当前A的前k项已经分成了m段。

读入a_{k+1}，令$s_{m+1}=a_{k+1}$，即将新读入的数字当作只有一个节点的堆，m++；

不断检查最后两个区间，反复执行下列操作：

1. 若$s_m>s_{m-1}$，已得到前k+1项的最优解，跳出；
2. 若$s_{m-1}\geq s_m$，将s_m并入第s_{m-1}对应的集合，并更新该集合的中位数s_{m-1}，m--；

操作2的实现：维护中位数：

可并堆为大根堆

当堆内元素个数大于区间长度的一半时删除堆顶元素，则堆中的元素一定是该区间内较小的一半元素，堆顶元素即为该区间的中位数。

每个堆只需要记录所维护区间最小的一半数字。合并两个区间后，若此时堆中元素个数超过一半，就弹出多余的数字。

若序列A的中位数位M_A，序列B的中位数为M_B，且$M_A>=M_B$。合并两个序列后，新序列的中位数$M_C<=M_A$

奋斗吧少年

巨大的成功需要付出巨大的代价

no sacrifice, no success