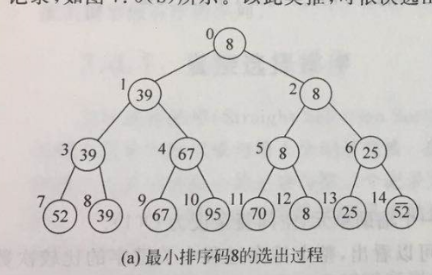
**树型选择排序又称为竞标赛排序。**

**基本思想**

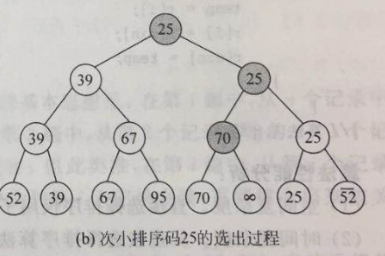
首先针对n个记录进行两两比较，比较的结果是把关键字较小者作为优胜者上升到父节点，得到n/2（向上取整）个比较的优胜者，作为第一步比较的结果保留下来；然后对这n/2（向上取整）个记录在进行关键字比较，如此重复，直到选出一个关键字值最小的记录为止。

这个过程可以用一个含有n个叶子节点的完全二叉树来表示，称为这种树为优胜者。

树中的每一个非叶子节点中的关键字均等于其左右孩子节点中较小的关键字，其根节点中的关键字就是所有叶子节点中的最小的关键字。



要求求出次小关键字记录，只需要将叶子节点中最小的关键字值改为“无穷”，然后从该叶子节点开始与其左（或右）兄弟的关键字进行比较，用较小关键字修改父节点的关键字，用此方法，从下到上修改该叶子节点到根节点路径上的所有节点的关键字值，则可得到次小关键字记录。



以此类推，可以依次选出从小到打的所有关键字。

如果开始记录个数不是2的k次幂，则将叶子节点数补足到2k个，这样，胜者数就为一棵满二叉树。

**胜者树的节点结构**

1.date 数据域

2.index 节点在满二叉树中的序号

3.active 此节点是否参加选择，1参加，0不参加

public class TreeNode { //胜者树的节点结构

public int date; //数据域

public int index; //结点在满二叉树中的序号

public int active; //是否参与选举

}

**胜者树的结构**

采用顺序存储结构

TreeNode[] tree; //胜者树节点数组

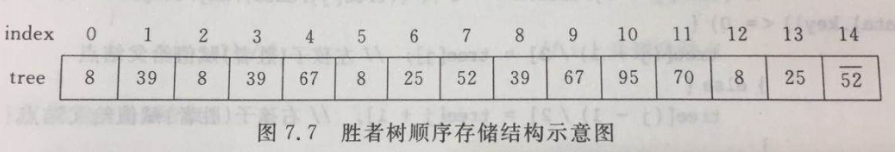
胜者树中的结点按满二叉树的结点编号顺序依次存放在顺序表tree中，胜者树中的叶子结点即为待排序结点。

在树形选择排序算法中引入变量

leafSize胜者树中的叶子结点个数，leafSize=n,n为待排序记录的长度

TreeSize胜者树的结点总数，TreeSize=2n-1

loadindex叶子结点在顺序表tree中的起始存放位置，Loadindex=n-1



**步骤**

1.变量初始化，令待排序的结点个数为n，则leafSize=n，TreeSize=2n-1，loadindex=n-1

2.将n个待排序结点复制到胜者树的n个叶子结点中，将nums[]依次赋值到tree[loadindex,loadindex+1,…,TreeSzie-1]中

3.构造胜者树：将n个叶子结点的关键字进行两两比较，得到n/2个关键字值较小的结点，保留下来，再将n/2个结点得关键字进行两两比较，得到n/4个较小关键字值的结点，保留下来，依次类推，最后得到的根结点为最小关键字值的结点为止。

4.调整胜者树：先将根节点保留在原数组nums中，再把具有根节点值所对应的叶子结点的值改为最大值，然后从该叶子结点开始，和其左（或右）兄弟的值进行比较，修改从该叶子结点到根的路径上各结点的值，直到根结点。

5.重复步骤4，直到得到n个结点为止

**算法性能分析**

**1.空间复杂度**

树形选择排序虽然减少了排序时间，但是使用了较多的附加存储空间。对于含有n个记录的顺序表，当n=2k时，需要使用**2n-1**个结点来存放胜者树；当2k-1<n<2k的时候同样也要使用2\*2k-1个结点。

**2.时间复杂度**

树形选择排序过程可以用一棵满二叉树来表示，高度为(向下取整) +1 。除了第一次选择具有最小的关键字值的记录，需要进行n-1次关键字比较外，调整胜者树所需要的关键字比较次数为O()，总的关键字比较次数为O(n)，由于结点的移动次数不超过比较次数，所以时间复杂度为**O(n)**。

**稳定性：稳定**

**代码**：

public static void tournamentSort(int[] nums) { //树形选择排序

TreeNode[] tree; //胜者树结点数组

int leafSize = 1; //胜者树的叶子结点数

while (leafSize < nums.length)//生成叶子结点

leafSize \*= 2;//得到胜者树叶子结点(外结点)的个数，该个数必须是2的幂数

int TreeSize = 2 \* leafSize - 1; //胜者树的所有节点数

int loadindex = leafSize - 1; //叶子结点（外结点）存放的起始位置

tree = new TreeNode[TreeSize];

int k=0;

for (int i = loadindex; i < TreeSize; i++) {//把待排序结点复制到胜者树的叶子结点中

tree[i]=new TreeNode();

tree[i].index=i;

if(k<nums.length){ // 把nums的所有数据都加入到胜者树中，从loadindex开始

tree[i].date=nums[k];

k++;

}

else//若胜者树后边还有叶子结点，赋值为空结点，或不参与比赛。

tree[i].active=0;

}

//开始查找关键字值最小的结点，一趟排序

int i = loadindex; //从第一个叶子结点开始

while (i > 0) {

int j = i;

while (j < 2 \* i) { //处理各对比赛者，树结点到TreeSize=2\*leafSize-1=2\*i-1,所以j<2\*i

if (tree[j + 1].active == 0 || tree[j].date <= tree[j + 1].date)//j+1为空结点或j的关键字小于j+1的关键字

tree[(j - 1) / 2] = tree[j];//当前的父节点赋值为较小的j位置的关键字

else

tree[(j - 1) / 2] = tree[j + 1];//否则，父结点赋值为j+1的关键字

j += 2;//下一对比赛者

}

i = (i - 1) / 2;//处理上层结点

}

//处理剩余的n-1个记录

for (i = 0; i < nums.length - 1; i++) {

nums[i] = tree[0].date; //将胜者树的根也就是当前最小关键字存入数组nums

tree[tree[0].index].active = 0;//将记录对应外结点不参见比赛

updateTree(tree, tree[0].index);//调整胜者树

}

nums[nums.length - 1] = tree[0].date;

}

public static void updateTree(TreeNode[] tree, int index) {//调整胜者树

//index是当前最小关键字的下标

//将当前最小关键字调至无穷比较

//也可以跟最小关键字相比较的那个结点无条件胜出

if (index % 2 == 0)//当下标为偶数的时候，对手为左结点

tree[(index - 1) / 2] = tree[index - 1];//左结点自动胜出

else //奇数是，对手为右结点

tree[(index - 1) / 2] = tree[index + 1];//右结点自动胜出

int i = (index - 1) / 2; //上升到父节点

int j;//记录当前结点的对手结点

while (i > 0) { //直至到i=0，根节点选出当前最小的关键字

if (i % 2 == 0)

j = i - 1;

else

j = i + 1;

if (tree[i].active == 0 || tree[j].active == 0) {

if (tree[i].active == 0) //当i为空结点的时候

tree[(i - 1) / 2] = tree[j];//j自动获胜

else //当j为空结点的时候

tree[(i - 1) / 2] = tree[i];//i自动获胜

} else {

if (tree[i].date <= tree[j].date) //当i的关键字小

tree[(i - 1) / 2] = tree[i]; //i获胜

else//j的关键字小

tree[(i - 1) / 2] = tree[j];//j获胜

}

i = (i - 1) / 2; //上升到父节点

}

}