第10章　应用经典二叉树编程

吴永辉

EMAIL：YHWU@FUDAN.EDU.CN

WECHAT：13817360465

ICPC ASIA PROGRAMMING CONTEST 1ST TRAINING COMMITTEE – CHAIR

- 本章给出如下经典二叉树的实验范例：
- 提高数据查找效率的二叉搜索树；优先队列的最佳存储结构二叉堆；以及在此基础上的兼具二叉搜索树和二叉堆性质的树堆；
- 用于算法分析和数据编码的霍夫曼树，以及多叉霍夫曼树；
- 在二叉搜索树的基础上，为进一步提高数据查找效率而派生出的AVL树、伸展树。

10.1 二叉搜索树的实验范例

10.1.3 THE ORDER OF A TREE

- 试题来源：2011 MULTI-UNIVERSITY TRAINING CONTEST 16 - HOST BY TJU
- 在线测试：HDOJ 3999

- 众所周知，二叉搜索树的形状与插入的键的顺序有很大关系。准确地说：
- 1. 在一棵空树上插入一个键*K*，然后树就变成一棵只有一个节点的树；
- 2. 在非空树中插入键*K*，如果小于根，则在其左子树上插入*K*；否则在其右子树上插入*K*。
- 我们称键的顺序为“树的顺序”，给出一棵树的树的顺序，请您找一个序列，使之能产生与给出的序列相同形状的二叉搜索树；同时，这一序列的字典序最小。

- **输入**
- 输入中有多个测试用例，每个测试用例的第一行给出一个整数N（$N \le 100,000$），表示节点的数目。第二行给出N个整数，从K_1到K_N，表示树的顺序。为了简明，从K_1到K_N是一个1到N的序列。
- **输出**
- 输出N个整数的一行，这是一个树的顺序，以最小的字典序生成相同形状的树。

试题解析

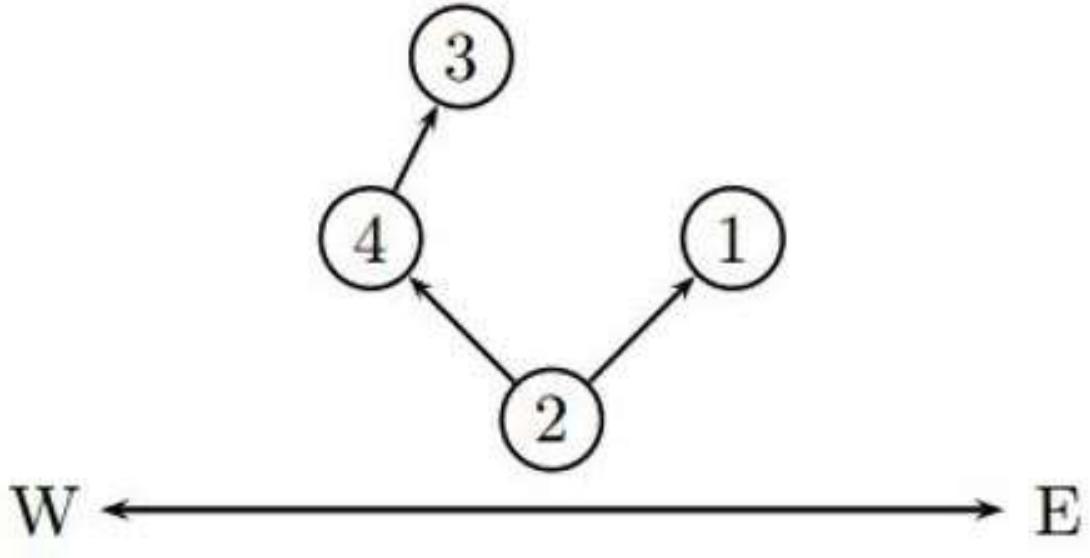
- 首先，根据输入键的顺序，构造对应的二叉搜索树；然后，前序遍历这棵二叉树，得出的树的顺序就是以最小的字典序生成的相同形状的树。
- 注意：为了使得顺序值之间用空格隔开，设首节点标志*FLAG*。递归前*FLAG*初始化为TRUE，随后改为FALSE。每搜索到一个节点，若*FLAG*为FALSE，则尾随一个空格，直至搜索至最后一个节点。

10.1.4 ELVEN POSTMAN

- 试题来源：2015 ACM/ICPC ASIA REGIONAL CHANGCHUN ONLINE
- 在线测试：HDOJ 5444

- 精灵是非常奇特的生物。正如大家所知，他们可以活很长时间；他们的魔法让人不能掉以轻心；此外，他们住在树上；等等。但是，精灵的有些事情您可能不知道：虽然他们能通过神奇的心灵感应，非常方便地传送讯息（就像电子邮件一样），但他们更喜欢其他更“传统”的方法。
- 所以，对于一个精灵邮递员，了解如何正确地把邮件送到树上的房间是很重要的。在精灵树的交叉处，最多两条分支，或者向东，或者向西。精灵树看起来非常像人类计算机科学家定义的二叉树。不仅如此，在对房间编号时，他们总是从最东边的位置向西对房间进行编号。东边的房间通常更受欢迎，也更贵，因为可以看日出，这在精灵文化中非常重要。
- 精灵通常把所有的房间号按顺序写在树根上，这样邮递员就知道如何投递邮件了。顺序如下：邮递员将直接访问最东边的房间，并记下沿途经过的每个房间。到达第一个房间后，它将去下一个没有访问过的最东边房间，并在途中记下每个未被访问过的房间；这样做直到所有房间都被访问过为止。
- 给出写在根上的序列，请您确定如何到达某个房间。

例如，序列 2, 1, 4, 3 写在如下的树的树根上。



- **输入**
- 首先给出一个整数T（$T \leq 10$），表示测试用例的数目。
- 对于每个测试用例，在一行上给出一个数字N（$N \leq 1000$），表示树上的房间数。然后的N个整数表示写在根上的房间号的序列，分别为$A_1, \ldots\ldots, A_N$，其中$A_1, \ldots\ldots, A_N \in \{1, \ldots, N\}$。
- 在接下来的一行，给出一个数字Q，表示要投递的邮件数。之后，给出 Q个整数$X_1, \ldots, X_Q$，表示每封邮件要送达的房间号。
- **输出**
- 对于每个查询，输出邮递员送邮件所需的移动序列（由E或W组成）。E表示邮递员向东分支走，而W表示向西分支走。如果目的地的房间在根上，就输出一个空行。
- 请注意，为了简便，我们设定邮递员总是从根开始，而不管他要去哪个房间。

试题解析

- 本题给出线性排列的树节点值，第一个数字是根节点值，如果后面的数不大于当前值，则往东（E）走，否则往西（W）走。显然，这样的线性排列为二叉搜索树的前序遍历，而其中序遍历则为1...N。
- 本题是在已知这棵二叉搜索树的前序遍历和中序遍历的基础上，查询根至某些节点所要走的路径。由此得出解题步骤：
- 输入写在根上的房间序列，并构造二叉搜索树。注意，插入节点时需记录插入时的路径；
- 依次处理Q个询问：每次询问按小（或相等）“东”、大“西”的规则，搜索由根至当前房间的路径。

10.2 二叉堆的实验范例

10.2.2 Binary Search Heap Construction

- 试题来源：ULM LOCAL 2004
- 在线测试：POJ 1785，ZOJ 2243

- 堆是这样的一种树，每个内节点被赋了一个优先级（一个数值），使得每个内节点的优先级小于其父节点的优先级。因此，根节点具有最大的优先级，这也是堆可以用于实现优先队列和排序的原因。
- 在一棵二叉树中的每个内节点有标号和优先级，如果相应于标号它是一棵二叉搜索树，相应于优先级它是一个堆，那么它就被称为树堆（TREAP）。给出一个标号-优先级对组成的集合，请构造一个包含了这些数据的树堆。

- **输入**
- 输入包含若干测试用例，每个测试用例首先给出整数N，本题设定$1 \le N \le 50000$；后面给出N对的字符串和整数$L_1/P_1, \ldots, L_N/P_N$，表示每个节点的标号和优先级；字符串是非空的，由小写字母组成；数字是非负的整数。最后的一个测试用例后面以一个0为结束。
- **输出**
- 对每个测试用例在一行中输出一个树堆，树堆给出节点的说明。树堆的形式为 (< LEFT SUB-TREAP >< LABEL >/< PRIORITY >< RIGHT SUB-TREAP >). 子树堆递归输出，如果是树叶就没有子树输出。

试题解析

本题要求构造一个大根堆，并中序遍历这个大根堆。输出格式是用括号表示法描述中序遍历。

由于标号和优先级的取值都是唯一的，每一个测试用例都有唯一的树堆解。因此，树堆可以用一种直截的方式构造：找到优先级最高的节点，作为树堆的根；然后，将剩余的节点划分成两个集合：标号比根小的节点和标号比根大的节点。递归地采用这一方法，基于第一个集合构造左子树，基于第二个集合构造右子树。如果这两个节点集都是空的，则只是一个叶节点，递归结束。

上述的方法基于列表来实现，就要用线性的时间找到具有最大优先级的节点，用线性时间划分节点集。在最坏情况下，运行时间为 $O(n^2)$，在 n 的值高达 50000 时，运行速度就很慢。

因此，要在划分节点集合的过程进行优化，以减少运行时间。在初始时，按节点的标号对节点进行排序（时间复杂度 $O(n*\log(n))$），这样，节点集就表示为一个区间，会有一个最大值和一个最小值。然后，对这个区间，用线性时间找到具有最大优先级的节点，以此将一个区间被划分为两个子区间。此后，对每个区间递归上述步骤：一个区间被划分为两个子区间。对于每个区间，用线性时间找到具有最大优先级的节点。

采用有序统计树（order-statistic tree），通过扩大元素列表的方法，在 log(n)时间内找到最大优先级。为此，构建一棵足够大的完全二叉树，在其底层的 n 个叶节点按标号递增顺序自左向右排序，并被标记了优先级，树的每个内节点则被标记在该节点下面的节点的优先级的最大值。这样的树以自底向上的方式在线性时间内构造。所以，就不用扫描整个区间来搜索最大优先级，而是可以使用存储在内节点中的子树的优先级最大值。因此，我们可以通过在有序统计树中由区间两端自底向上搜索在 log(n)时间内找到最大优先级。然后，采用自顶向下搜索，在区间内查找具有该优先级的元素。因此，算法的总运行时间是 O(n*log(n))。

试题要求构造一个大根堆，并中序遍历这个大根堆。输出格式是用括号表示法描述中序遍历。设堆为 *ost*。由于节点数的上限为 50000，因此将长度为 n 的初始数据作为堆的叶节点放入 $ost[2^{16}\cdot\cdot 2^{16}+n-1]$，建堆过程中形成的 n-1 个分支节点放入 $ost[2^{16}-n+1\cdot\cdot 2^{16}-1]$。计算过程如下：

①输入 n 个节点的标号和优先级，建立 p 表和 l 表，其中第 i 个输入节点的标号为 $l[i]$，优先级为 $p[i]$；

②创建堆的存储空间：按标号递增顺序建立 n 个节点的堆序号，放入 $ost[2^{16}\cdot\cdot 2^{16}+n-1]$

③自底向上构建初始堆：$ost[2^{16}..2^{16}+n-1]$为叶子，上推双亲层 $ost[2^{15}..\left\lfloor\frac{2^{16}+n-1}{2}\right\rfloor]$，其中 $ost[i]$为堆节点 $2*i+1$ 与 $2*(i+1)$间的大者的下标，即 $p[ost[i]]=\max\{p[ost[2*i+1]], p[ost[2*(i+1)]]\}$（$2^{15}\le i\le\left\lfloor\frac{2^{16}+n-1}{2}\right\rfloor$）。若 n 为奇数，则 $p[ost[\left\lfloor\frac{2^{16}+n-1}{2}\right\rfloor]]=p[2^{16}+n-1]$；按照上述方法上推倒数第 2 层，即 $ost[2^{15}..\left\lfloor\frac{2^{16}+n-1}{2}\right\rfloor]$的双亲层 $ost[2^{14}..\left\lfloor\frac{2^{16}+n-1}{4}\right\rfloor]$；……；依此类推，向上倒推 16 层时，即可得到大根 $ost[1]$。例如，按标号递增顺序建立 8 个节点的堆序号，放入 $ost[2^{16}..2^{16}+7]$。按照上述方法得到如下结构的初始堆（图 10.2-4）：

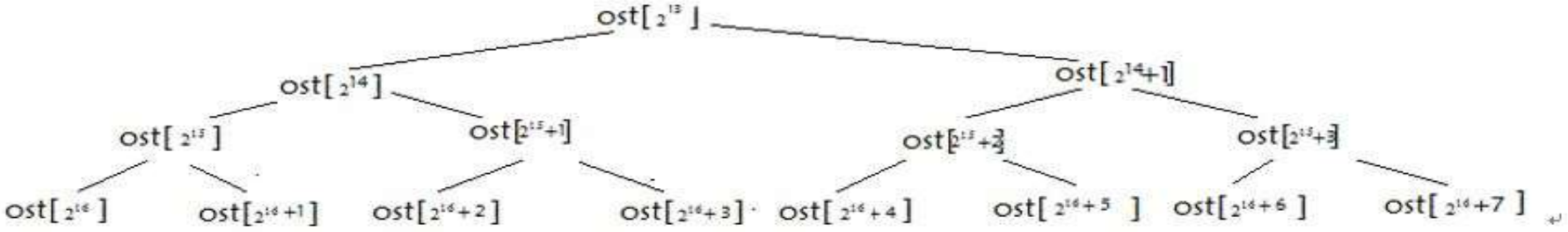


图 10.2-4

递归计算堆的括号表示法

通过递归过程 *recurse*(*f*, *t*)，计算和输出序号区间[*f*··*t*]对应子堆的括号表示法：

✧ 计算子根序号 *r*。首先计算区间两端优先级大者的下标 *r*，即 *p*[*ost*[*r*]]=max{*p*[*ost*[*f*]], *p*[*ost*[*t*]]}；然后按照自下而上顺序，依次搜索每层节点区间两端的节点 *f* 和 *t*：若 *f* 为左儿子(*f*%2==0)，且同层有优先级最大的右邻节点((*f*+1<*t*)&&(*p*[*ost*[*f*+1]]> *p*[*ost*[*r*]]))，则调整 *r*(*r*=*f*+1)；若 *t* 为右儿子(*t*%2 ==1)，且同层有优先级最大的左邻节点((*t*-1>*f*)&& (*p*[*ost*[*t*-1]] > *p*[*ost*[*r*]]))，则调整 *r*(*r*=*f*+1)。接下来继续上推双亲层（*f*= *f*/2，*t*=*t*/2），直至上推至根为止（*f*≤*t*）。

✧ 将 *r* 下调至合适位置：若 *r* 与左儿子的优先级相同(*ost*[*r*]==*ost*[*r**2])，则 *r* 下调至左儿子(*r* = *r**2)；若 r 与右儿子的优先级相同(*ost*[*r*] == *ost*[*r**2+1])，则 *r* 下调至右儿子（*r*= *r**2+1）；

✧ 输出'('，表示[*f*,*t*]对应堆的括号表示法开始；若 *r* 有左子树(*f*<*r*)，计算和输出左子树的括号表示法（*recurse*(*from*, *r*-1)）；输出子根 *r* 的标号（*l*[*ost*[*r*]])和优先级（*p*[*ost*[*r*]]）；计算和输出右子树的括号表示法（*recurse*(*r*+1, *t*)）。

显然，递归调用 *recurse*(2^{16}, 2^{16}+*n*-1)便可以得到问题解。

10.2.3 Decode the Tree

- 试题来源：ULM LOCAL 2001
- 在线测试：POJ 2568，ZOJ 1965

- 给出一棵树（也就是一个连通无回路图），树的节点用整数1, …, *N*编号。 树的PRUFER码构造如下：取具有最小的编号的叶节点（仅和一条边关联的节点），将该树叶和它所关联的边从图中删除，并记下该叶节点所关联的节点的编号。在获取的图中重复这一过程，直到只有一个节点留了下来。很明显，这个唯一留下的节点编号为*N*。被记下的*N*−1个数的序列被称为树的PRUFER码。
- 给出PRUFER码，请重构一棵树。树表示如下：
- T ::= "(" N S ")"
- S ::= " " T S | EMPTY
- N ::= NUMBER
- 即，树用括号把它们括起来，用数字表示其根节点的标识符，后面跟用一个空格分开的任意多的子树（也可能没有）。

图 10.2-5 中的树是样例输入中的第一行给出的测试用例。

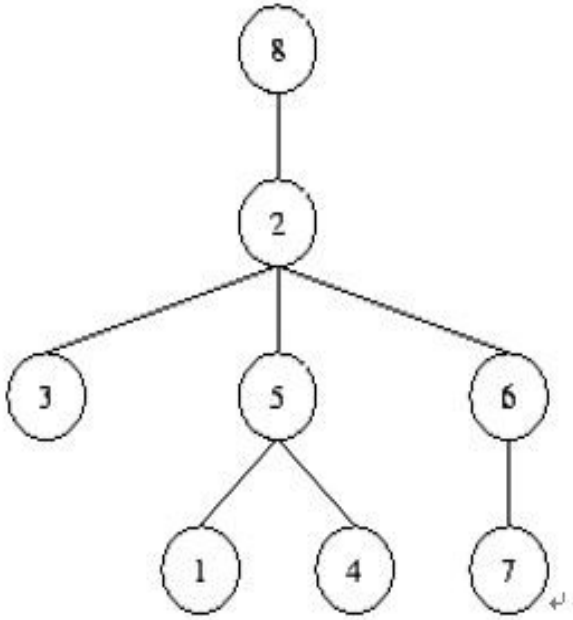


图 10.2-5

要注意的是，按上述定义，树的根也可能是树叶。这仅用于我们指定某个节点为树根的情况。通常，我们这里处理的树被称为“无根树”。

- **输入**
- 输入包含若干个测试用例，每个测试用例一行，给出一棵树的PRUFER码。给出用空格分开的*N*-1个数，输入以EOF结束。设定1≤*N*≤50。
- **输出**
- 对每个测试用例，输出一行，以上述的形式表示相应的树。表示一棵树有多种方式表示：选择你喜欢的一种。

试题解析

- 试题输入PRUFER码，要求构造对应的树，并输出其括号表示。本题的关键是如何构造与PRUFER码对应的树。
- 在PRUFER码中所有没有出现的节点编号都是树的叶节点编号（可能编号为*N*的节点除外）。根据PRUFER码的定义，该码中的第一个数与编号最小数的叶节点相邻；这样，就可以构成树的一条边。如果在PRUFER码中这个数字还会再次出现在PRUFER码中，则在此时，该数字所对应的节点还不是叶节点；而如果该数字不再出现，则它已经变成了一个叶节点：在这种情况下，将该节点加入到叶节点集合（优先队列）。重复同样的步骤*N*-2次，就得到其他边。

10.3 树堆的实验范例

- 10.3.1 树堆
- 10.3.2 可持久化树堆（非旋转树堆）

10.3.1 树堆

- **定义10.3.1.1（树堆(TREAP)）.** 树堆也被称为TREAP。对于一棵二叉搜索树，如果树节点带有一个随机附加域，使得这棵二叉搜索树也满足堆的性质，则这棵二叉搜索树被称为一棵树堆。

- 树堆节点*X*的属性通常包含两个属性：关键字值*KEY*[*X*]；优先级*PRIORITY*[*X*]，一个独立选取的随机数。
- 树堆的节点采用结构体存储，定义如下：
- STRUCT NODE
- {
- INT *KEY*; //关键字
- INT *PRIORITY*; //随机优先级
- NODE* *LEFT*; //左儿子节点
- NODE* *RIGHT*; //右儿子节点
- };

- 假设树堆所有节点的优先级是不同的，所有节点的关键字也是不同的。树堆的节点排列成让关键字遵循二叉搜索树性质，并且优先级遵循大根堆（或小根堆）顺序性质，即，
- 性质1：如果*V*是*U*的左孩子，则*KEY*[*V*]<*KEY*[*U*]；
- 性质2：如果*V*是*U*的右孩子，则*KEY*[*V*]>*KEY*[*U*]；
- 性质3：如果*V*是*U*的孩子，则*PRIORITY*[*V*]<*PRIORITY*[*U*]，或*PRIORITY*[*V*]>*PRIORITY*[*U*]。
- 其中，性质1和性质2为二叉搜索树性质，性质3为堆性质。所以，树堆（TREAP）具有二叉搜索树和堆的特征，即“TREAP=TREE+HEAP”。

例如，二叉搜索树节点的关键字集合为{A, B, E, G, H, K, I}，节点输入顺序不同，构成的二叉搜索树不同；但如果为每个节点附加了一个随机的优先级，得到的具有二叉搜索树和小根堆性质的树堆如图 10.3.1-1。

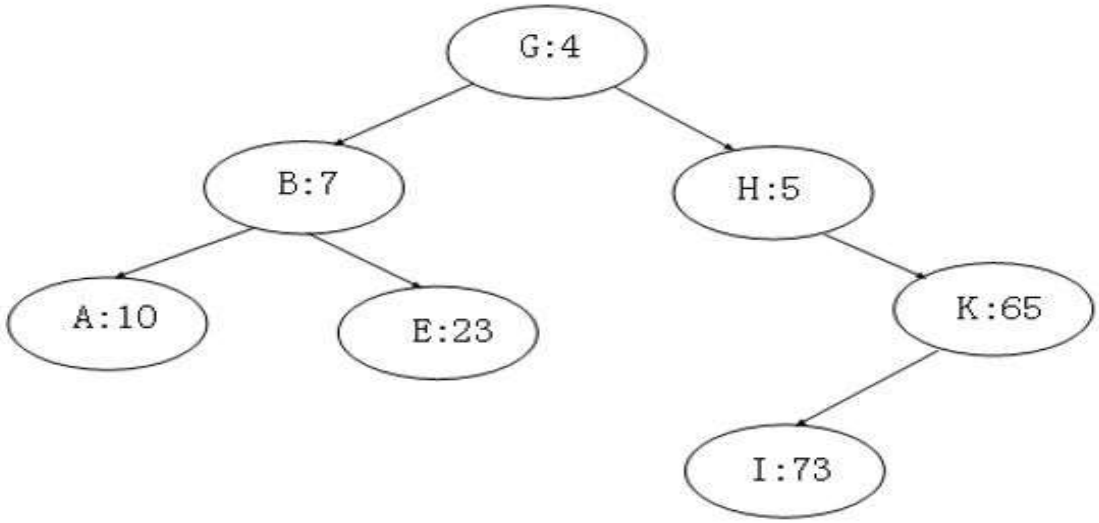


图 10.3.1-1

由图 10.3.1-1 可以看出，插入关联关键字的节点 x_1，x_2，…，x_n 到一棵树堆内，最终的树堆是将这些节点以优先级顺序插入一棵二叉搜索树而形成的，也就是说，以小根堆为例，如果 $priority[x_i]<priority[x_j]$，则树堆可以视为节点 x_i 在节点 x_j 之前被插入而形成的二叉搜索树。

- 相应于二叉搜索树，按照优先级顺序构建的树堆有如下改变：
- 树堆的形态不依赖节点的输入顺序
- 按照优先级顺序构建的树堆是唯一的。树堆的根节点是优先级最高的节点，其左儿子是左子树里优先级最高的节点，右儿子亦然。所以，构造树堆，可以视为先把所有节点按照优先级由高到低排序，然后依次以构造二叉搜索树的算法插入节点。所以，当节点的优先级数确定的时候，这棵树堆是唯一的。
- 2、树堆的操作更高效
- 基于随机给出的优先级构造的二叉搜索树的期望高度为O(LOGN)，因而树堆的期望高度亦是O(LOGN)，这可使二叉搜索树的任何操作变得更加高效。
- 3、树堆的编程比平衡二叉搜索树更简便
- 为了维护堆性质和二叉搜索树的性质，树堆采用旋转操作，但仅需要左旋转和右旋转两种，相应于AVL树、伸展树（在后面论述），树堆的编程要简单很多，这正是树堆的特色之一。

以小根堆为例，当节点 X 的优先级数小于节点 Y 的优先级数时，右旋转；当节点 Y 的优先级数小于节点 X 的优先级数时，左旋转；左、右旋转如图 10.3.1-2 所示。

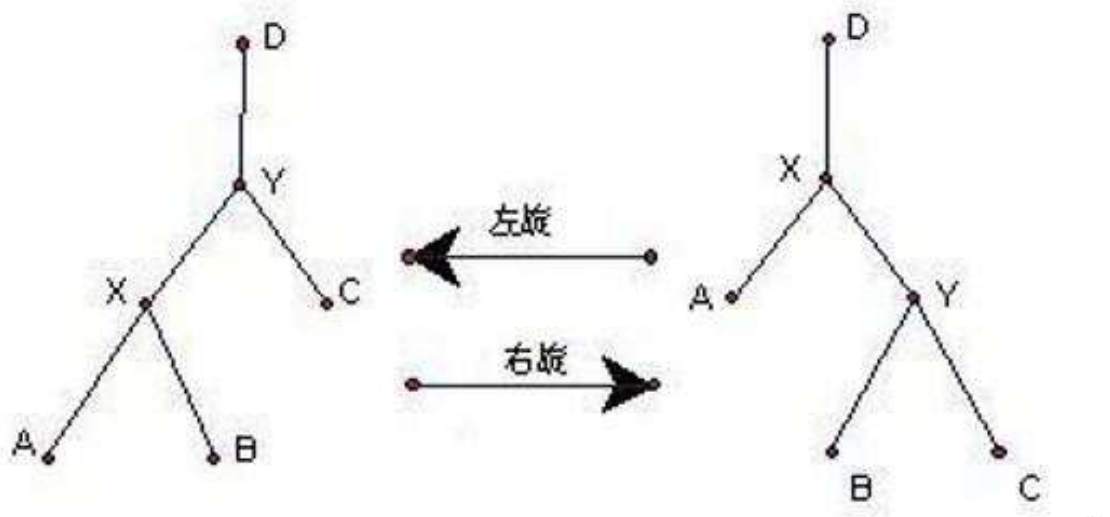


图 10.3.1-2

- 树堆有如下五种基本操作，其中分离和合并是最为重要的操作，因为树堆的许多操作在这两种操作的基础上展开：
- **（1）查找**
- 与一般的二叉搜索树查找一样。但是由于树堆的随机化结构，在树堆中查找的期望复杂度是O(LOG *N*)。
- **（2）插入**
- 首先，和二叉搜索树的插入一样，先把要插入的元素插入树堆，成为树堆的一个叶节点，然后，通过旋转来维护堆的性质。

- **（3）删除**
- 首先，和二叉搜索树的删除一样，找到相应的节点；然后，删除操作如下：
- （1）若该节点为叶节点（没有孩子节点），则直接删除；
- （2）若该节点仅有一个孩子节点，则将其孩子节点取代它；
- （3）否则，进行相应的旋转：以小根堆为例，每次找其优先级数最小的儿子，向与其相反的方向旋转，即左儿子优先级数最小，则右旋转；右儿子优先级数最小，则左旋转；直到该节点为上述情况之一，然后进行删除。

- （4）分离
- 要把一个树堆按大小分成两个树堆，前*K*个节点划分给树堆*A*，剩余节点划分给树堆*B*。则在要分开的位置加一个虚拟节点，关键字排序为第*K*+1，优先级为最高；将该虚拟节点插入，待该节点旋转至根节点时，则左右两个子树就是两个树堆了。时间复杂度是O(LOG*N*)。
- （5）合并
- 合并是指把两个树堆合并成一个树堆，其中第一个树堆的所有节点的关键字都必须小于第二个树堆中的所有节点的关键字。合并的过程和分离的过程相反，要加一个虚拟的根，把两棵树分别作为左右子树，然后对根节点做删除操作。

10.3.1.1 DOUBLE QUEUE

- 试题来源：ACM SOUTHEASTERN EUROPE 2007
- 在线测试：POJ 3481，UVA 3831

新成立的巴尔干投资集团银行（Balkan Investment Group Bank (BIG-Bank)）在布加勒斯特开设了一个新的办事处，配备了 IBM Romania 提供的现代计算环境，并使用了现代信息技术。与往常一样，银行的每个客户都用一个正整数 K 来标识，在客户到银行办理业务时，他或她会收到一个正整数优先级 P。银行的年轻管理人员的一项发明让银行服务系统的软件工程师非常吃惊。他们要打破传统，有时，服务台叫具有最低优先级的客户，而不是叫最高优先级的客户。因此，系统将接收以下类型的请求：

0	系统需要停止服务
1 K P	将客户 K 添加等待列表中，其优先级为 P
2	服务具有最高优先级的客户，并将其从等待名单中删除
3	服务具有最低优先级的客户，并将其从等待名单中删除

请您编写一个程序，帮助银行的软件工程师实现所要求的服务策略。

- **输入**
- 每行输入给出一个可能的请求；只有在最后一行给出停止请求（代码0）。本题设定，当有一条请求是在等待列表中加入一个新客户时（代码1），在列表中不会有其他的请求加入相同的客户或相同的优先级。标识符K 总是小于10^6，优先级P小于10^7。客户可以多次办理业务，并且每次的优先级可以不同。
- **输出**
- 对于代码为2或3的每个请求，您的程序要以标准输出的方式，在单独的一行中输出要服务的客户的标识符。如果发出请求时等待列表为空，则程序输出零（0）。

试题解析

- 加入一个新客户（代码1），则执行树堆的插入操作，插入节点的*V*值（顾客优先级，作为节点关键字值）和节点的*R*值（随机函数RAND获得，作为节点优先级）以及节点信息*INFO*（顾客编号）。
- 服务一个客户（代码为2或3），则是先*FIND_MAX*找到最大*V*值的节点信息*INFO*（代码为2），或*FIND_MIN*找到最小*V*值的节点信息*INFO*（代码为3），然后执行树堆的删除操作，删除该客户（*REMOVE*）即可。

10.3.2 可持久化树堆（非旋转树堆）

- 非旋转树堆的关键在于维护性质并不改变树的形态，所以也被称为可持久化树堆。分离和合并是非旋转树堆的重要操作。

- （1）分离
- 分离*SPLIT*(*X*, *K*)将以*X*为根的树堆的前*K*个节点划分给树堆*A*，剩余节点划分给树堆*B*，返回树堆*A*和*B*的根。分4种情况讨论：
- 若*K*=0，则树堆*A*空，树堆全部划入树堆*B*（图10.3.2-1(1)）
- 若树堆的规模不大于*K*，则树堆*B*空，树堆全部划入树堆*A*（图10.3.2-1(2)）；
- 在树堆的规模大于*K*的情况下，若树堆的左子树的规模不小于*K*，则对树堆的左子树做递归分离*SPLIT*(*X'*, *K*)，其中*X'*为左子树的根，将左子树中的前*K*个元素划分给树堆*A*，剩余元素划分给树堆*B*的左子树，树堆的右子树划分给树堆*B*的右子树（图10.3.2-1(3)）；
- 在树堆的规模不大于*K*的情况下，若树堆的左子树的规模小于*K*，则树堆的左子树作为给树堆*A*的左子树，对树堆的右子树做递归分离，将前（*K*-1-树堆的左子树规模）个节点作为树堆*A*的右子树，剩余节点划分给树堆*B*（图10.3.2-1(4)）；

- （2）合并
- 合并操作*MERGE*(*X*, *Y*) 将树堆*A*和树堆*B*合并成树堆*TREAP*，返回树堆*TREAP*的根，其中*X*是树堆*A*的根，*Y*是树堆*B*的根。同样递归实现，分析*X*和*Y*的优先级，设树堆*A*和树堆*B*是小根堆：
- 如果*X*的优先级数小于*Y*的优先级数，则先将*B*与*A*的右子树合并，合并结果作为树堆*TREAP*的右子树，树堆*TREAP*的左子树即为*A*的左子树，如图10.3.2-2(1)所示，即递归调用*MERGE*(*RC*(*X*), *Y*)，其中*RC*(*X*)是*X*的右孩子；否则，先将*A*与*B*的左子树合并，合并结果作为树堆*TREAP*的左子树，而树堆*TREAP*的右子树即为*B*的右子树（图10.3.2-2(2)），即递归调用*MERGE*(*X*, *LC*(*Y*))，其中*LC*(*Y*)是*Y*的左孩子。

10.3.2.1 VERSION CONTROLLED IDE

- 试题来源：ACM ICPC ASIA REGIONAL 2012:: HATYAI SITE
- 在线测试：UVA 12538

- 程序员使用版本控制系统来管理他们的项目中的文件，但是在这些系统中，只有当您手动提交文件时，该版本才被保存。您能实现一个IDE，在您插入或删除一个字符串的时候，自动保存一个新的版本吗？
- 在缓冲区中的位置从左到右从1开始进行编号。最初，缓冲区是空的，版本为第0个版本。然后你可以执行以下的3类指令（*VNOW*是执行命令之前的版本，*L*[*V*]是第*V*个版本在缓冲区的长度）：
- 1 *P S*：在位置*P*后插入字符串*S*（0≤*P*≤*L*[*VNOW*]，*P*=0表示插在缓冲区开始的位置之前）。*S*至少包含1个字母，至多包含100个字母。
- 2 *P C*：从位置*P*开始删除*C*个字符（*P*≥1；*P*+*C*≤*L*[*VNOW*]+1）。剩余的字符（如果有的话）向左移动，填入空格。
- 3 *V P C*：在第*V*个版本中(1≤*V*≤*VNOW*)，从位置*P*开始打印C个字符 (*P*≥1; *P*+*C*≤*L*[*V*]+1)。
- 第一条指令肯定是指令1（插入指令），每次执行了指令1或指令2之后，版本数增加1。

- 输入
- 仅有一个测试用例。首先给出一个整数N（$1 \le N \le 50{,}000$），表示指令的数目。
- 接下来的N行每行给出一条指令。所有插入的字符串的总长度不超过1,000,000。

- **输出**
- 按序对每条指令3，输出结果。
- 为了防止您预处理命令，输入还采取以下的处理方案：
- 每条类型1的指令变成 1 *P+D S*
- 每条类型2的指令变成 2 *P+D C+D*
- 每条类型3的指令变成 3 *V+D P+D C+D*
- 其中*D*是这条指令处理之前，您输出的小写字母'C'的数目。例如，在混淆之前，样例输入是：
- 6
- 1 0 ABCDEFGH
- 2 4 3
- 3 1 2 5
- 3 2 2 3
- 1 2 XY
- 3 3 2 4
- 您的程序读取下面给出的样例输入，而处理的真正的输入如上。

试题解析

- 简述题意：给三种操作
- 操作1：在*P*位置插入一个字符串；
- 操作2：从*P*位置开始删除长度为C的字符串；
- 对于前两个操作，每操作一次形成一个历史版本。
- 操作3：输出第*V*个历史版本中从*P*位置开始的长度为C的字符串。

10.4 哈夫曼树的实验范例

10.4.2 多叉哈夫曼树

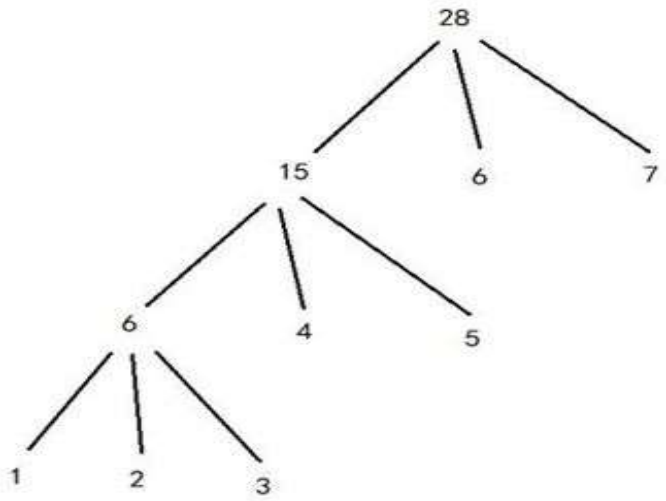
- 哈夫曼树也可以是K（K>2）叉的。K叉哈夫曼树是一棵满K叉树，每个节点要么是叶子节点，要么它有K个子节点，并且树的权最小。因此，构造K叉哈夫曼树的思想是每次选K个权重最小的元素来合成一个新的元素，该元素权值为这K个元素权值之和。但是，如果按照这个步骤，可能最后剩下的元素个数会小于K。

- 设给出M个节点，其权值为$W_1, W_2, \ldots\ldots, W_M$。构造以此$M$个节点为叶节点的$K$叉哈夫曼树的算法如下：
- 首先， 将给出的M个节点构成M棵K叉树的集合$F=\{T_1, T_2, \ldots\ldots, T_M\}$。其中，每棵$K$叉树$T_I$中只有一个权值为$W_I$的根节点，子树为空，$1 \le I \le M$；如果$(M-1)\%(K-1) \neq 0$，就要在集合$F$中增加$K-1-(M-1)\%(K-1)$个权值为0的“虚叶节点”；然后，重复做以下两步：
- （1）在F中选取根节点权值最小的K棵K叉树作为子树，构造一棵新的K叉树，并且置新的K叉树的根节点的权值为其子树根节点的权值之和；
- （2）在F中删除选取的这K棵K叉树，同时将新得到的K叉树加入F中；
- 重复(1)、(2)，直到在F中只含有一棵K叉树为止。这棵K叉树便是K叉哈夫曼树。

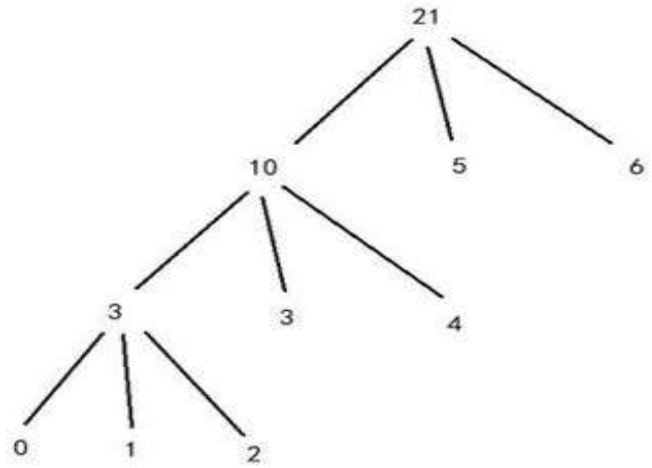
- 如果(*M*-1)%(*K*-1)=0，不必增加权值为0的“虚叶节点”，第一次选*K*个权重最小的节点构造一棵*K*叉树；而如果(*M*-1)%(*K*-1)≠0，第一次选(*M*-1)%(*K*-1)+1个权重最小的节点，并虚拟*K*-1-(*M*-1)%(*K*-1)个权值为0的“虚叶节点”，构造一棵*K*叉树。

例 14.2.1 构造序列 1, 2, 3, 4, 5, 6, 7 对应的三叉哈夫曼树，则 m=7，k=3。因为(m-1)%(k-1)=6%2=0，第一次选当前 3 个权重最小的节点 1, 2, 3；第二次选当前 3 个权重最小的节点 4, 5, 6；第三次，最后 3 个节点 6, 7, 15；因此得到 3 叉哈夫曼树如图 10.4.2-1(a)所示。

构造序列 1, 2, 3, 4, 5, 6 对应的三叉哈夫曼树，则 m=6，k=3。因为(m-1)%(k-1)=5%2=1≠0，第一次选当前 2 个权重最小的节点 1, 2，并虚拟 1 个权值为 0 的“虚叶节点”，构造一棵 k 叉树；第二次选当前 3 个权重最小的节点 3, 3, 4；第三次，最后 3 个节点 10, 5, 6；因此得到 3 叉哈夫曼树如图 10.4.2-1(b)。



(a)



(b)

图 10.4-2

10.4.2.1 SORT

- 试题来源：2016 ACM/ICPC ASIA REGIONAL QINGDAO ONLINE
- 在线测试：HDOJ 5884

- 最近，BOB刚刚学习了一种简单的排序算法：合并排序。现在，BOB接受了ALICE的任务。
- ALICE给BOB N个排序序列，第I个序列包含A_I个元素。BOB要合并所有这些序列。他要编一个程序，一次合并不超过K个序列。合并操作的耗费是这些序列的长度之和。而ALICE允许这个程序的耗费不超过T。因此，BOB想知道可以使程序及时完成的最小的耗费K。

输入

输入的第一行给出一个整数 t_0，表示测试用例的数量。接下来给出 t_0 个测试用例。对于每个测试用例，第一行由两个整数 N（$2 \le N \le 100000$）和 T（$\sum_{i=1}^{N} a_i < T < 2^{31}$）组成；下一行给出 N 个整数 $a_1, a_2, a_3, ..., a_N$（$\forall i$，$0 \le a_i \le 1000$）。

输出

对于每个测试用例，输出最小的 k。

试题解析

- 简述题意：对N个有序序列进行归并排序，每次可以选择不超过K个序列进行合并，合并代价为这些序列的长度和，总的合并代价不能超过T，问K最小是多少。
- 解题思路：通过二分法计算最小K，即设K的取值区间为$[2, N]$，每次取区间中位数MID，如果能够构造MID叉哈夫曼树，则在左子区间寻找更小K值；否则在右子区间寻找可行的K值。用两个队列来实现K叉哈夫曼树：表示N个有序序列的元素先进行排序，放在队列Q1中；并使用另外一个队列Q2来维护合并后的值，显然，队列Q2也是递增序列。每次取值时，从Q1和Q2两个队列的队头取小的值即可。
- 这里注意，对于N个数，构造K叉哈夫曼树，如果$(N-1)\%(K-1)\neq 0$，要虚拟$K-1-(N-1)\%(K-1)$个权值为0的叶节点。
- 解题的算法的时间复杂度为$O(NLOGN)$。

10.5 AVL树的实验范例

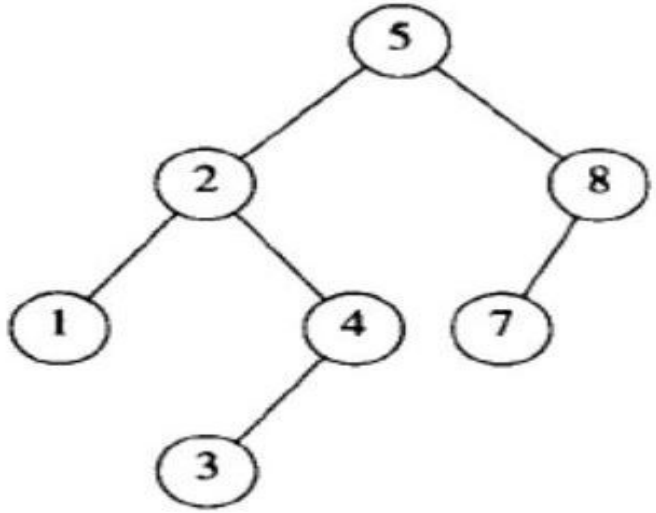
定义 10.5.1（平衡二叉树(Balanced Binary Tree)）. 平衡二叉树或者是一棵空二叉树，或者是具有以下性质的二叉树：它的左右两个子树的高度差的绝对值不超过 1，并且左右两个子树也都是平衡二叉树。

平衡二叉树的常用算法有红黑树、AVL、树堆、伸展树等。

平衡二叉搜索树（Self-Balancing binary search tree）又被称为 AVL 树，定义如下：

定义 10.5.2（AVL 树(Self-Balancing binary search tree)）. AVL 树是一棵二叉搜索树，且每个节点的左右子树的高度之差的绝对值（平衡因子）最多为 1。

例如，图 10.5-1 是一棵 AVL 树。



AVL 树的节点类型取结构体，一般包括数据域 *val*，以其为根的子树高度 *h*，平衡因子 *Bf*（左子树高度与右子树高度之差），左指针 *left* 和右指针 *right*。其中，以 *T* 为根的子树高度 *T->h=max(T->left->h, T->right->h)+1*，平衡因子

$$T->bf=\begin{cases} 0 & T->left->h==T->right->h \\ -(T->right->h) & T->left==null \\ T->left->h & T->right==null \\ (T->left->h)-(T->right->h) & \text{否则} \end{cases}$$

- AVL树是带了平衡功能的二叉搜索树。对于AVL树，在插入元素时，计算了每个节点的平衡因子是否大于1；如果大于，那么就进行相应的旋转操作以维持平衡性。在插入元素后，以插入的元素为起点，向上追溯，找寻到的第一个平衡因子大于1的节点，该节点称为不平衡起始节点。以不平衡起始节点向下给出插入节点的位置，则可以把不平衡性分为四种情况：
- （1）LL：插入一个新节点到不平衡起始节点的左子树的左子树，导致不平衡起始节点的平衡因子由1变为2；
- （2）RR：插入一个新节点到不平衡起始节点的右子树的右子树，导致不平衡起始节点的平衡因子由-1变为-2；
- （3）LR：插入一个新节点到不平衡起始节点的左子树的右子树，导致不平衡起始节点的平衡因子由1变为2；
- （4）RL：插入一个新节点到不平衡起始节点的右子树的左子树，导致不平衡起始节点的平衡因子由-1变为-2。

- AVL树有两种基本的旋转：
- （1）右旋转：将根节点旋转到其左孩子的右孩子位置；
- （2）左旋转：将根节点旋转到其右孩子的左孩子位置。
- 针对上述四种情况，可以通过旋转使AVL树变平衡，有四种旋转方式，分别为：右旋转，左旋转，左右旋转（先左后右），右左旋转（先右后左）。

对于 LL 情况，通过将不平衡起始节点右旋转使其平衡。如图 10.5-2 所示，旋转前 A 的平衡因子大于 1 且其左孩子 B 的平衡因子大于 0（(A->*bf*>1)&&(A->*left*->*bf*>0)）；旋转后，原 A（不平衡起始节点）的左孩子 B 成为父节点，A 成为其右孩子，而原 B 的右子树成为 A 的左子树。

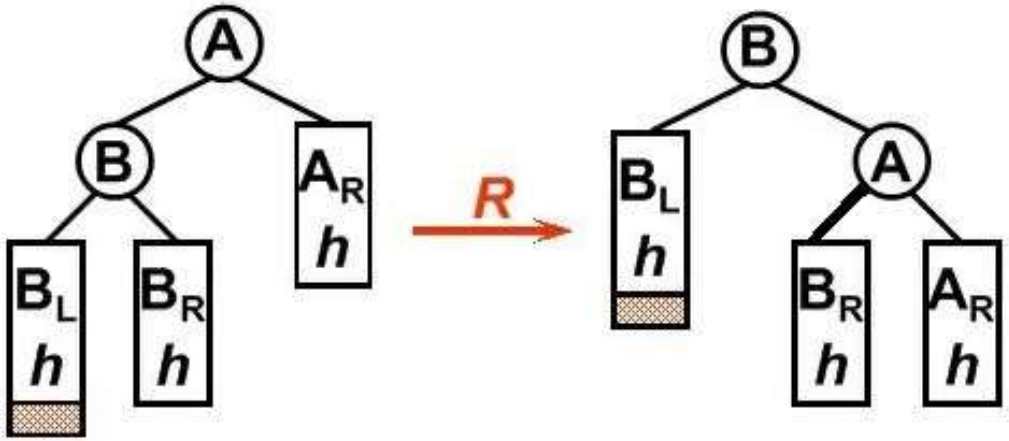


图 10.5-2

对于 RR 的情况，通过将不平衡起始节点左旋转使其平衡。如图 10.5-3 所示，旋转前 A 的平衡因子小于-1，右子树 B 的平衡因子小于 0（(A->*bf*<-1)&&(A->*right*->*bf*<0)）；旋转后，原 A（不平衡起始节点）的右孩子 B 成为父节点，A 成为其左孩子，而原 B 的左子树成为 A 的右子树。

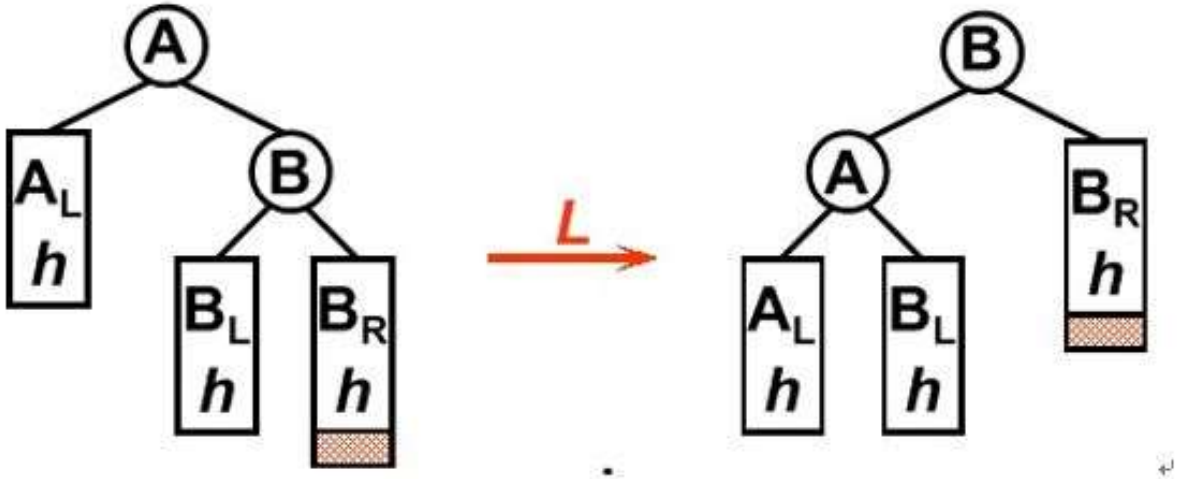


图 10.5-3

对于 LR 的情况，则需要进行左右旋转：先左旋转，再右旋转，使 AVL 树平衡。如图 10.5-4 所示，旋转前，A 的平衡因子大于 1，B 的平衡因子小于 0（(A->*bf*>1)&&(A->*left*->*bf*<0)）；在节点 B 按照 RR 型向左旋转一次之后，二叉树在节点 A（不平衡起始节点）仍然不能保持平衡，这时还需要再向右旋转一次。

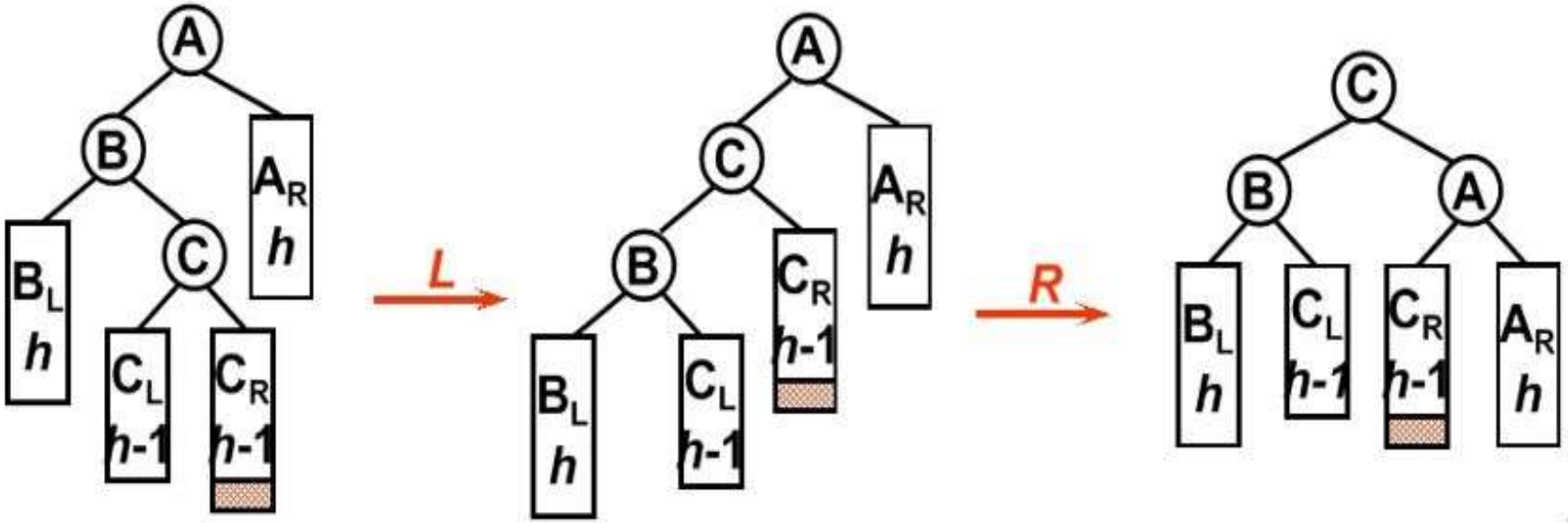


图 10.5-4

对于 RL 的情况，则需要进行右左旋转：先右旋转，再左旋转，如图 10.5-5 所示，旋转前 A 的平衡因子小于-1，右子树 B 的平衡因子大于 0（(A->*bf*<-1)&&(A->right->*bf*>0)）；先右后左旋转，旋转方向刚好同 LR 型相反。

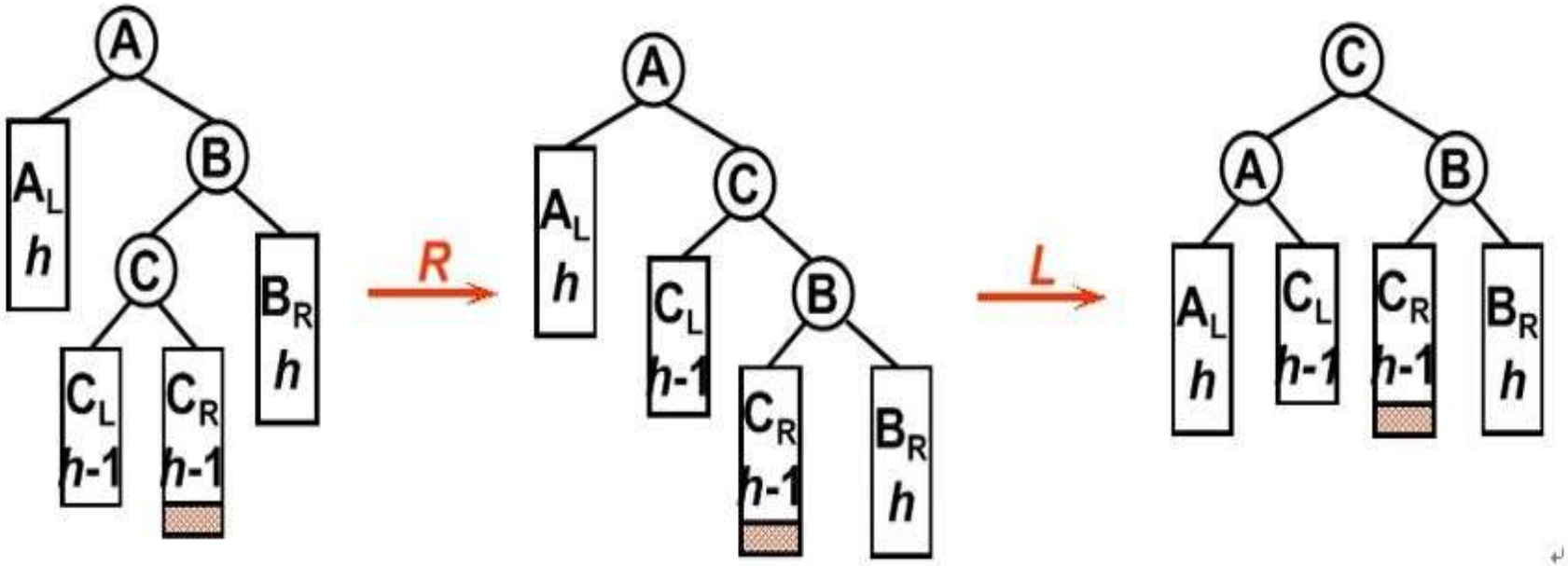


图 10.5-5

10.5.1 DOUBLE QUEUE

- 试题同10.3.1.1

试题解析

- 在【10.3 树堆的实验范例】中，基于树堆求解本题。在本节，本题求解基于AVL树。由于计算过程需经常借助高度或平衡因子，因此元素结构（即节点的数据域）包含*VAL*（顾客优先级，作为节点关键字值），*DATA*（顾客编号），*H*（以当前节点为根节点的子树的高度）和*BF*（平衡因子，即左子树高度与右子树高度之差。
- 加入一个新客户（代码1），则执行AVL树的插入操作。服务一个客户（代码为2或3），则是先找到最大或最小*VAL*值的节点，然后执行AVL树的删除操作。为了在增删操作后保持树的平衡性，可能需要进行左旋转、右旋转、先左后右旋转和先右后左旋转。

10.5.2 THE *K*TH GREAT NUMBER

- 试题来源：2011 ACM/ICPC ASIA DALIAN ONLINE CONTEST
- 在线测试：HDOJ 4006

- 小明和小宝在玩一个简单的数字游戏。在一轮中，小明可以选择写下一个数字，或者问小宝第*K*大的数字是什么。因为小明写的数字太多，小宝觉得头晕。现在，请您来帮小宝。

- **输入**
- 本题给出若干测试用例。对于每个测试用例，第一行给出两个正整数*N*，*K*，然后给出*N*行。如果小明选择写一个数字，就给出一个“I”，后面给出小明写下的那个数字。如果小明选择问小宝，就给出一个“Q”，您就要输出第*K*大的数字。
- **输出**
- 在一行中输出一个整数，表示一条询问要求的第*K*大的数字。

试题解析

本题以一棵 AVL 树实现。树的节点的数据域包含 *key*（小明写下的数字），*repeat*（*key* 的重复次数），*size*（以该节点为根的子树中数字的总数，即左儿子的 *size*+右儿子的 *size*+节点的 *repeat*）和 *h*（以该节点为根的子树的高度）。

对于“I”操作，在 AVL 树中插入一个数字 *x*。按照左小右大的顺序寻找插入位置。如果找到一个其 *key* 域值为 *x* 的节点 *T*（*T.key*==*x*），则节点 *T* 的 *repeat* 域值+1（++*T*->*repeat*）。

对于“Q”操作，则在 AVL 树中寻找第 *k* 大的数字，方法如下：

```
int selectKth(Node *rt, int k)                    //返回以 rt 为根的子树中第 k 大数字
{
    计算 rt 的左子树规模 lSize= rt->left->Size;
    if (k <= lSize) return selectKth(rt->left, k);   //第 k 的数字在 rt 的左子树中，递归搜索左子树中第 k 大的数
    else if (lSize + rt->repeat < k)                 //若第 k 大的数字在 rt 的右子树中，则递归搜索右子树中第（k-左子树规模-rt 的重复次数）大的数
        return selectKth(rt->right, k - lSize - rt->repeat);
    return rt->key;                                  //返回以 rt 为根的子树中第 k 大的数字
}
```

10.6 伸展树的实验范例

- **定义10.6.1（伸展树）.** 伸展树（SPLAY TREE），也被称为分裂树，是一种自调整的二叉搜索树。对于伸展树S中的每一个节点的键值X，其左子树中的每一个元素的键值都小于X，而其右子树中的每一个元素的键值都大于X。而且，可以沿着从该节点到树根之间的路径，通过一系列的旋转（伸展操作）可以把这个节点搬移到树根。
- 为简明起见，键值为X和Y的节点称为节点X和节点Y。

伸展操作 *splay* 是通过一系列旋转将伸展树 S 中节点 x 调整至树根。在调整的过程中，要分以下三种情况分别处理：

情况 1：节点 x 的父节点 y 是树根节点。如果节点 x 是节点 y 的左孩子，则进行一次 Zig（右旋转）操作；如果节点 x 是节点 y 的右孩子，则进行一次 Zag（左旋转）操作。经过旋转，节点 x 成为 S 的根节点，调整结束。Zig 和 Zag 操作如图 10.6-1 所示：

图 10.6-1

情况 2：节点 x 的父节点 y 不是树根节点，节点 y 的父节点为节点 z，并且节点 x 与节点 y 都是各自父节点的左孩子或者都是各自父节点的右孩子；则进行 Zig-Zig 操作或者 Zag-Zag 操作。Zig-Zig 如图 10.6-2 所示：

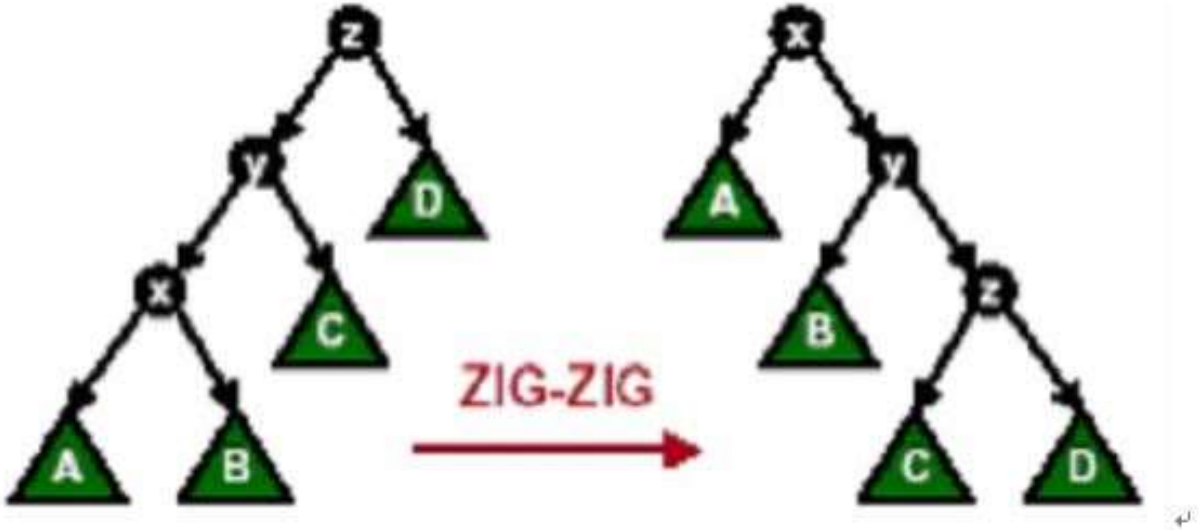


图 10.6-2

情况 3：节点 x 的父节点 y 不是树根节点，节点 y 的父节点为节点 z，并且节点 x 与节点 y 中一个是其父节点的左孩子而另一个是其父节点的右孩子。则进行 Zig-Zag 操作或者 Zag-Zig 操作。Zig-Zag 如图 10.6-3 所示：

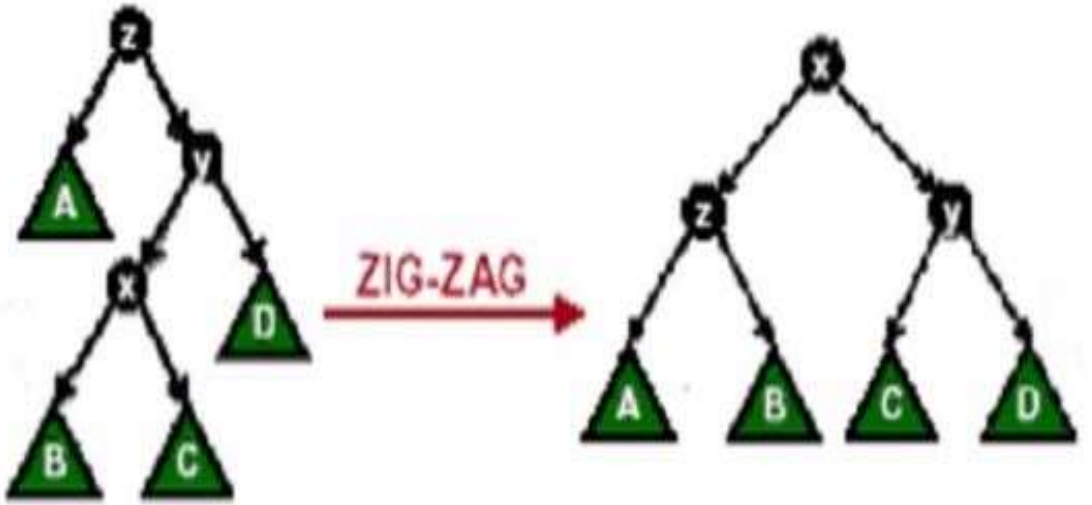


图 10.6-3

利用 splay 操作，可以在伸展树 S 上进行如下的伸展树基本操作：

（1）*merge*($s1, s2$)：将两个伸展树 $s1$ 与 $s2$ 合并成为一个伸展树，其中 $s1$ 中所有节点的键值都小于 $s2$ 中所有节点的键值。

首先，找到伸展树 $s1$ 中包含最大键值 x 的节点；然后，通过 *splay* 将该节点调整到伸展树 $s1$ 的树根位置（执行子程序 *splay*($s1$, $s1$->s)）；最后，将 $s2$ 作为节点 x 的右子树。这样，就得到了新的伸展树 s。如图 10.6-4 所示。

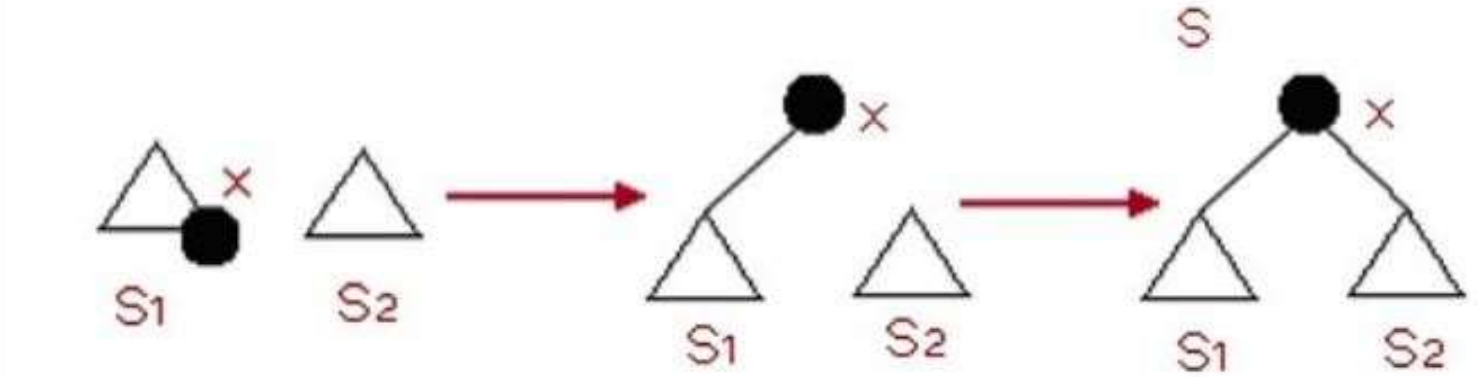


图 10.6-4

（2）*split*(*o*, *x*, *s*1, *s*2)：将伸展树 *o* 分离为两棵伸展树 *s*1 和 *s*2，其中 *s*1 中所有节点的数值均小于第 *x* 大的数，*s*2 中所有节点的数值均大于第 *x* 大的数。

首先，通过执行 *splay*(*o*, *x*)，将第 *x* 大的数旋转至 *o* 的根位置；然后，取其左子树为 *s*1（*s*1=*o*->*ch*[0]），取其右子树为 *s*2（*s*2=*o*->*ch*[1]），如图 10.6-5 所示：

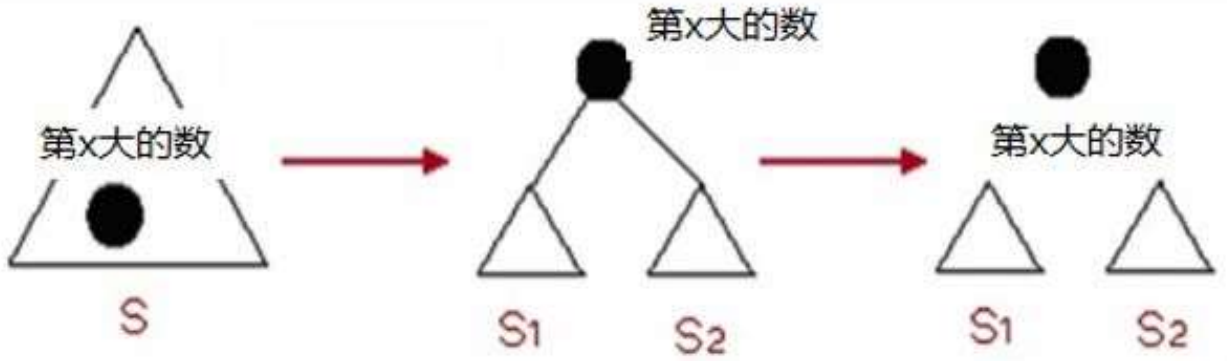


图 10.6-5

- （3）*DELETE*(*ROOT*, *X*)：将伸展树*ROOT*中包含第*X*大的数的节点从删除。
- 首先，从伸展树*ROOT*中分离出两棵子树：存储第1到第*X*-1大的数的子树*LEFT*和存储第*X*+1大到第*N*大的数的子树*RIGHT*；然后，通过执行*ROOT*=*MERGE*(*LEFT*, *RIGHT*) 合并伸展树*LEFT*和*RIGHT*，返回其根*ROOT*。

- （4）*INSERT*(*ROOT*, *X*, *V*)：将数值*V*插入伸展树*ROOT*中包含第*X*大的数的节点之后（第*X*大的数≤*V*<第*X*+1大的数）
- 首先，通过执行*SPLIT*(*ROOT*, *X*+1, *S*1, O)和*SPLIT*(*ROOT*, *X*, *M*, *S*2)，从伸展树*ROOT*中分离出子树*S*1（存储第1到第*X*大的元素）和子树*S*2（存储第*X*+1到第*N*大的元素）；然后构建包含数据值为*V*的节点，左右指针空，子树规模1的单根树*TT*，顺序合并*S*1、*TT*和*S*2，（*ROOT*=*MERGE*(*MERGE*(*S*1, *TT*), *S*2)），并将含数据*V*的插入节点旋转至根（*SPLAY*(*ROOT*, *X*+1)）。

10.6.1 SUPERMEMO

- 试题来源：POJ FOUNDER MONTHLY CONTEST – 2008.04.13, YAO JINYU
- 在线测试：POJ 3580

- 您的朋友JACKSON被邀请参加一个名为“SUPERMEMO”的电视节目，在节目中，参加者被告知要玩一个记忆游戏。首先，主持人告诉参加者一个数字序列，$\{A_1, A_2, \ldots A_N\}$。然后，主持人对数字序列进行一系列操作和查询，这些操作和查询包括：
- *ADD X Y D*：对子序列$\{A_X \ldots A_Y\}$中的每个数加上*D*。例如，对序列{1, 2, 3, 4, 5}执行"ADD 2 4 1"，结果是{1, 3, 4, 5, 5}。
- *REVERSE X Y*：对子序列$\{A_X \ldots A_Y\}$进行翻转。例如，对序列{1, 2, 3, 4, 5}执行"REVERSE 2 4"，结果是{1, 4, 3, 2, 5}。
- *REVOLVE X Y T*：对子序列$\{A_X \ldots A_Y\}$循环右转一位，转动*T*次。例如，对序列{1, 2, 3, 4, 5}执行"REVOLVE 2 4 2"，结果是{1, 3, 4, 2, 5}。
- *INSERT X P*：将*P*插在A_X后面。例如，对序列{1, 2, 3, 4, 5}执行"INSERT 2 4"，结果是{1, 2, 4, 3, 4, 5}。
- *DELETE X*：删除A_X。例如，对序列{1, 2, 3, 4, 5}执行"DELETE 2"，结果是{1, 3, 4, 5}。
- *MIN X Y*：查询子序列$\{A_X \ldots A_Y\}$中的最小值。例如，对序列{1, 2, 3, 4, 5}执行"MIN 2 4"，结果是2。
- 为了让节目更有趣，参加者有机会场外求助其他人，这就是说，JACKSON在回答问题感到困难时，他可能会打电话给你寻求帮助。请您观看电视节目，并写一个程序，对每个问题给出正确的答案，以便在JACKSON给您打电话时为他提供帮助。

- **输入**
- 第一行给出N ($N \leq 100000$)。接下来的N行描述序列。然后给出M ($M \leq 100000$)，表示操作和查询的数目。
- **输出**
- 对每个"*MIN*"查询，输出正确答案。

试题解析

- 本题求解基于一棵伸展树。
- 伸展树节点的数据域除了包含*V*（数据）以外，还包含懒惰标记*S*（子树规模），*MINN*（子树最小值），*FLIP*（翻转标志）和*ADD*（累加值）；懒惰标记*FLIP*和*ADD*在子程序*PUSHDOWN*中维护。设区间*X*~*Y*为序列中第*X*个元素至第*Y*个元素。
- ADD操作：为*X*~*Y*元素加一个*D*值。首先调用*SPLIT*切出*X*~*Y*元素。然后改变给切出的子树*ROOT*的*ROOT->ADD*，*ROOT->MIN*，*ROOT->V*。再调用*MERGE*切出的子树*ROOT*合并进原序列。
- REVERSE操作：把*X*~*Y*元素反转。首先用*SPLIT*切出*X*~*Y*元素，然后改变切出的子树*ROOT*的*ROOT->FLIP*标记。再调用*MERGE*将切出的子树*ROOT*合并进原序列。
- REVOLVE操作：把*X*~*Y*元素偏移*T*位。注意*T*可以为负。负向左，正向右。首先，对*T*进行修正。*T*=(*T*%(*R*-*L*+1)+(*R*-*L*+1))%(*R*-*L*+1))，这样正负方向就一致了，而且解决了没必要的偏移。这样问题就转化为把[*X*, *Y*]序列变成*LEFT*+[*Y*-*T*+1, *Y*]+[*X*, *Y*-*T*]+*RIGHT*，那么只要调用*SPLIT*切出[*X*, *Y*-*T*]就行了。注意*T*=0时要特判，不然等于切了个0空间。
- INSERT操作：在第*X*元素后插入*P*。首先调用*SPLIT*切出左段1~ *X*的元素，然后调用*MERGE*合并这个新的元素*P*，再*MERGE*右段*X*+1~*N*的元素。这里要注意，虽然伸展树是二叉搜索树，但是伸展树一旦为了维护原有序列顺序，则可以不再遵循BST的左≤中≤右原则。这里的INSERT就是，把*P*元素*MERGE*到右边是防止BST维护改变顺序（*MERGE*只对左子树进行BST维护）。
- DELETE操作：切出被删元素的左右两段，调用*MERGE*合并这两段即可。
- MIN操作：求*X*~*Y*元素最小值。依赖于*PUSHUP*(也就是*MAINTAIN*)，每次变动都要维护*ROOT-V*，*ROOT->LEFT*，*ROOT->RIGHT*三部分的最小值。首先切出[*X*, *Y*]，*ROOT->MINN*就是结果。
- BUILD操作，首先在0号位置加一个无穷大的前置节点，这样就可以调用*SPLIT*切出[*X*, *Y*]这个段了，如果没有前置节点，则应该这么切[1, *X*-1]，当*X*=1时就会出错。BUILD(0, *N*)。这样0号点就被放在了最左边。BUILD的时候为了保持原序列顺序，根据位置进行二分BUILD。在最初把伸展树的高度给压下去。因为伸展树的平衡性能实在太差，不能 BUILD成链。

10.6.2 TUNNEL WARFARE

- 试题来源：POJ MONTHLY--2006.07.30, UPDOG
- 在线测试：POJ 2892

- 抗日战争时期，中国军民在华北平原的广大地区广泛地开展地道战。一般情况下，通过地道，将村庄相连成一行。除了两头，每个村庄都与相邻的两个村庄相连。
- 日军会经常对一些村庄发动扫荡，捣毁其中的部分的地道。八路军指挥官要求了解地道和村庄的最新连接状态。如果一些村庄被隔离了，就要立即恢复连接。

- **输入**
- 输入的第一行给出两个正整数*N*和*M*（$N, M \leq 50{,}000$），表示村庄和事件的数量。接下来的*M*行每一行描述一个事件。
- 有三种不同的事件，以如下不同的格式描述：
- D *X*：第*X*个村庄被扫荡。
- Q *X*：八路军指挥官要知道和第*X*个村庄直接或间接相连的村庄的数量，包括它自己。
- R：最近被扫荡的那个村庄被重建。
- **输出**
- 对于每次八路军指挥官的请求，在一行中输出回答。

试题解析

- 本题求解基于一棵伸展树。由于试题要求重建最近被扫荡的村庄，因此需要设立一个栈，存储被扫荡的村庄。
- 初始的时候，建一棵伸展树，插入N+1和0。然后依次处理每个事件：
- 如果第X个村庄被扫荡，则X入栈，并把X插入伸展树中；
- 如果查询第X个村庄，则如果X被扫荡，则输出0；否则，输出X的后继-前驱-1（X左边被扫荡的最近点和右边被扫荡的最近点之间的数字个数）。
- 如果重建最近被扫荡那个村，则栈顶元素出栈，并从伸展树中移出。