Assignment 3 : CSL 201
Topic : B-Trees

The following describes the outline of a template class implementing B-trees:

```
template <class T>
class BTreeNode {
friend class BTree<T>;
// your code here
}

template <class T>
class BTree {
private:
BTreeNode<T> *root;
// your code here
}
```

Your objective is to add the following functions to the template class for B-trees. Each function should run in $O(\log_t n)$ time, where n is the number of nodes in the B-tree, and $t \geq 2$ is the order of the B-tree. The only exception is **ThreeWayJoin** which should run in $O(1 + |h(T_1) - h(T_2)|)$ time, where $h(T_1)$ and $h(T_2)$ denote the heights of B-trees T_1 and T_2, respectively.

1. **Search.** Searches for a key in the B-tree. Returns a pointer to the key if it is present, otherwise returns **NULL**.

   ```
   T* search(T x) {};
   ```

2. **Insert.** Inserts a key into B-tree. If the key is already present, prints a "Key already present" message.

   ```
   void insert(T x) {};
   ```

3. **Delete.** Deletes the item with a particular key from the B-tree. If the key is not present, prints a "Key not present" message.

   ```
   void delete(T x) {};
   ```

4. **k-th Smallest.** Given an integer k, returns the kth smallest element in the B-tree.

   ```
   T kthSmallest(int k) {};
   ```

   Note that to implement this function in $O(\log_t n)$ time, you will have to augment the data structure by adding extra fields in each B-tree node.

5. **Three Way Join.** Given a key k and two B-trees T_1 and T_2 such that all keys in tree T_1 are $< k$ and all keys in tree T_2 are $> k$, returns a B-tree T containing all keys in $T_1 \bigcup \{k\} \bigcup T_2$. Note that T_1, T_2, and T are B-trees of the same order t.

```
BTree& threeWayJoin(T k, BTree& t1, BTree& t2) {};
```

   Note that to achieve this operation in $O(1 + |h(T_1) - h(T_2)|)$ time, you need to maintain the height of each B-tree node i.e., the length of any path to a leaf in the node's subtree.

6. **Split.** Given a key k and a B-tree T on which the function is called, makes two B-trees T_1 and T_2 of the same order as T such that T_1 contains all keys in tree T which are $< k$, and T_2 contains all keys in T which are $> k$.

```
void split(T k, BTree& t1, BTree& t2) {};
```

   Use the height field in each BTreeNode along with the function **three-WayJoin** to achieve this operation in $O(\log_t n)$ time.

1 Input Format

Your algorithm will maintain a sequence of B-trees $T_0, T_1, T_2, \cdots$ and so on. Initially, all the trees are empty. You can assume that there will never be more than 10000 trees in the program.

The first line of input will contain a single integer t greater than or equal to 2, which denotes the order of B-trees. All B-trees will be of the same order t. You can assume that the value of t never exceeds 1000.

The second line of the input will contain a single number n, which denotes the number of tree operations. Each line after that will contain a single instruction which can be an insert, a delete, a split, a join, or finding the k-th smallest element.

An insert instruction consists of the keyword "insert" followed by the number i of the B-tree T_i in which insertion is to be made. After this is a single number n, which denotes the number of integers to be inserted. This is followed by a sequence of n integers. Note that we allow batch insertions, i.e., a single insert instruction can be used to insert any number of items into a tree.

For example, to insert 10 numbers $45, 13, 11, 77, 2, 9, 1, 66, 5, 22$ into B-tree number 4, we will give the instruction:

```
insert 4 \\ insert operation on tree number 4
10 \\ number of integers to be inserted
45 13 11 77 2 9 1 66 5 22 \\ the list of integers to be inserted
```

A deletion instruction is similar to an insert instruction, except that we use the keyword "delete". For example, the following instruction deletes the set of numbers $7, 32, 41, 15, 16, 19, 22$ from B-tree 10:

```
delete 10 \\ delete operation on tree number 10
7 \\ number of integers to be deleted
7 32 41 15 16 19 22 \\ list of integers to be deleted
```

The instruction of a three-way join, consists of the keyword "join" followed by the number x of the first B-tree, the key k, and the number y of the second B-tree. This is followed by the number z of the B-tree which is obtained by joining T_x, k, and T_y. After the join, B-trees x and y will be empty, and B-tree z will contain all the keys in $T_x \bigcup \{k\} \bigcup T_y$. You can assume that all keys in the first B-tree are less than k, and all keys in the second B-tree are greater than k.

For example, the instruction to join trees 5 and 8 with intermediate key 77 into tree 11 is:

```
join 5 77 8 11
```

The instruction for a split, consists of the keyword "split" followed by the number x of the B-tree to be split, the splitting key k, and the numbers y and z of the target B-trees. After the split, B-tree y consists of all keys in B-tree x that were less than k, B-tree z consists of all keys in B-tree x which were greater than k, and B-tree x itself becomes empty.

Thus, the instruction to split tree 5 about key 61 with target trees 11 and 17 is:

```
split 5 61 11 17
```

The instruction for k-th smallest consists of the keyword "select" followed by the B-tree number and the rank of the key to be returned. To find the 11th smallest key in B-tree 7, we will write the following:

```
select 7 11
```

The output of "select" instruction is a single number on a line by itself, which is the value of the k-th smallest key in the corresponding B-tree.

Note that all other instructions except "select" have no output.

We are giving six input test files $in0, in1, in2, \cdots, in5$ in the tar archive "input-assgn3.tar.gz". The description of each test case along with the correct output is given in the file "test_cases".

2 Example

This example is illustrative and hence the B-trees considered here consist of small number of keys.

Suppose your program is given the following input:

```
2
7
insert 0 5 1 3 5 7 9
insert 1 5 11 14 15 16 17
delete 1 2 14 16
join 0 10 1 2
split 2 7 3 4
select 3 2
select 4 4
```

The first number tells us that $t = 2$, and hence all B-trees will have order 2 i.e., each node of the B-tree has either $2, 3$, or 4 children. The second number is equal to 7, and tells us that there are 7 instructions in the input. The first instruction inserts 5 numbers into B-tree 0. After this, $T_0 = (1\ 3\ 5\ 7\ 9)$. The second instruction inserts 5 numbers into B-tree 1. After this, $T_1 = (11\ 14\ 15\ 16\ 17)$. The third instruction deletes two numbers 14 and 16 from B-tree 1. After this, $T_1 = (11\ 15\ 17)$. The fourth instruction joins B-trees 0 and 1 along with intermediate key 10 into B-tree 2. After this instruction B-trees 0 and 1 are empty, and B-tree 2 contains the keys (1 3 5 7 9 10 11 15 17). The fifth instruction splits B-tree 2 about key 7 into B-trees 3 and 4. After this instruction, B-trees $0, 1$, and 2 are empty, and $T_3 = (1\ 3\ 5)$ and $T_4 = (9\ 10\ 11\ 15\ 17)$. The sixth instruction asks for the 2nd smallest key in B-tree 3. The output of this instruction is the number 3 on a line by itself. The seventh instruction asks for the 4th smallest key in B-tree 4. The output of this instruction is the number 15 on a line by itself.

The final output of the program consists of just two numbers, which are the outcome of the two "select" instructions:

```
3
15
```

Note that all instructions except "select" have no output.