Red-Black-Tree

1). Every node: value, color, leftChild, rightChild, father.

2). Main member variable: root.

3). Main methods:

Private: successor(), leftRotate(), rightRotare(), insertNode(), insertFixup(), deleteNode(), deleteFixup(), searchNode().

Public: insert(), delete(), search(), show().

4). How to insert or delete?

RB-tree insertion

Types		Operation
*z's father is **left** child*	***Case* 1L:** *z's uncle is red.*	*Change color.*
	***Case* 2L:** *z's uncle is black and z is **right** child.*	***Left** rotation, p(z).*
	***Case* 3L:** *z's uncle is black and z is **left** child.*	***Right** rotation, p(p(z)).*
*z's father is **right** child*	***Case* 1R:** *z's uncle is red.*	*Change color.*
	***Case* 2R:** *z's uncle is black and z is **left** child.*	***Right** rotation, p(z).*
	***Case* 3R:** *z's uncle is black and z is **right** child.*	***Left** rotation, p(p(z)).*

RB-tree deletion

Types		Operation
*z is **left** child*	***Case* 1L:** *x's sibling w is red.*	***Left** rotation, p(x).*
	***Case* 2L:** *x's sibling w is black and both of w's children are black.*	*Change color.*
	***Case* 3L:** *x's sibling w is black, and w's **left** children is red, and w's **right** child is black.*	***Right** rotation, w.*
	***Case* 4L:** *x's sibling w is black, and w's **right** child is red.*	***Left** rotation, p(x).*
*z is **right** child*	***Case* 1R:** *x's sibling w is red.*	***Right** rotation, p(x).*
	***Case* 2R:** *x's sibling w is black and both of w's children are black.*	*Change color.*
	***Case* 3R:** *x's sibling w is black, and w's **right** children is red, and w's **left** child is black.*	***Left** rotation, w.*
	***Case* 4R:** *x's sibling w is black, and w's **left** child is red.*	***Right** rotation, p(x).*

5). About the algorithms.

The insert and delete node are same as the binary search tree. The key of this algorithms is fix up the node to make the tree keep his red and black character. And the key of the fix up is the decisions which is above in the pictures. And the key to implement the decisions is the rotate methods. So, we get these, the problems are solved.

6). My description.

All my member variables and methods are private expect several public methods above. So, users can change the Red-Black-Tree only by these public methods, and these methods get my permit. So, it is safe and clearly.

B-Tree

1). Every node: isLeaf, num, value[], child[]. (num is the number of elements in array value[])

2). Main member variable: root, t.

3). Main methods:

Private: insertEntry(), insertNonFull(), splitChild(), deleteEntry(), mergeChild(), mergeSibling(), searchEntry(), deleteKey()

Public: insert(), delete(), search(), show()

4). How to insert or delete?

Insert:

Start at the root, recursion to insert. If the current node is full, then split it and recursion, else recursion directly. Until we find the leaf, insert the key directly. And then, recursion is over.

Delete:

Start at the root, recursion to delete. If the key is in the current node and current node is leaf, delete the key directly(case 1). If the key is in the current node and current node is not leaf and his nearest child is enough to reduce, then copy his precursor or successor to himself and delete his precursor or successor(case 2a,2b). If his nearest child is not enough to reduce, then merge his nearest two children and himself in one node and delete himself, then recursion(case 2c). If the key is not in the current node. We start at the root recursion down until find the node contain key. In this process, when we find the point(child[i]) we should recursion. If itself is enough to reduce, recursion directly. Else if it's nearest sibling is enough to reduce, catch one element from it's sibling, then recursion(case 3a). If it's two nearest sibling is not enough to reduce, then merge his right sibling and himself and one suitable element of his father, then recursion(case 3b).

5). About the algorithms.

The key of this algorithm is split and merge. We split, this is to make sure every node has at most 2t-1 keys. We merge, this is to make sure every node has at least t-1 keys. These two operate can make sure B-Tree keep his character. So, the problems are solved.

6). My description.

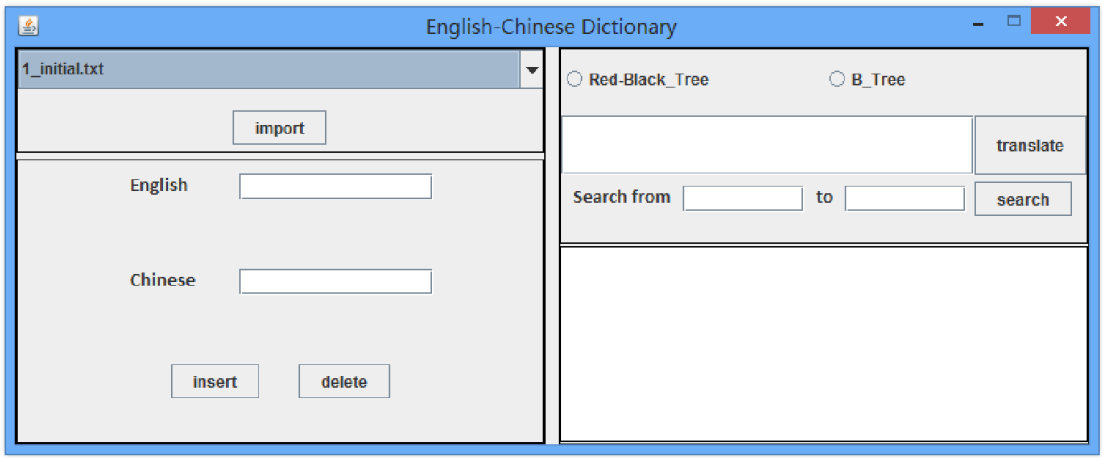
All my member variables and methods are private expect several public methods above. So, users can change the B-Tree only by these public methods, and these methods get my permit. So, it is safe and clearly.

GUI

1). The whole program has only one object for RedBlackTree and one object for BTree. We can only do insert, delete, search and show methods on the two objects because only these methods are public. And there is no other way to make change on our data in the tree.

2). When we choose the 1_initial.txt and import. The two objects are generated. And every time we import 1_initial.txt, the tree will be reset.

3). The UI.



The UI is simple, clearly, flattening and looks friendly. It is easy to use.

Bonus

I do no bonus on my UI, but I have bonus on my design for the software. I use the thinking of MVC model for my program. There are two class RedBlackTree and BTree, they are Model. Class MainView is View. And the class MyListener and Controllor are controller. In this way, three part is divided. And I can easily maintain the software.