Data Structure

HW2

0416037 李家安

這次的作業是做一個 threaded binary tree 裡面的 insertion、deletion、inorder_run、跟 reverseorder_run，其中我認為 deletion 應該是最難的部分。

Threaded binary tree 主要就是一個有隊前後維護的 binary tree，除了兩個 children 的 node 外，其他 node 會記錄前一個數字或後一個數字，也就是若此 node 無右子樹，將會把右邊的 pointer 指向比他大最小的數字，反之左子樹亦然，另外必須記錄此 pointer 指向的是子樹還是前(後)一個數字，因此在做 insertion 時，最後，在整棵樹上有兩個 node 是沒有數值的，分別是 head 跟 tail，此兩 node 將會方便 traverse 以及插入刪除時的維護。

我在做 insertion 時，先以 binary search 的方式找出要將此數值放在的位置，將前一個 node 的 pointer 連到新開的 node 上，然後維護前一個 node 的 is_threadr/is_threadl，最後再把 num + 1 就結束 insertion 了。

在做 inorder_run 跟 reverseorder_run 時，以 inorder_run 為例，只要從 head 一直向右跑到 tail 即可，但是要注意的是，如果向右走有子樹，我們必須先找到子樹最左邊的 node，也就是數值最接近原來的 node 的 node，才不會先找到比較大的 node 才跑回比較小的 node。而 reverseorder_run 只是從 tail 向左走到 head，中間亦小心檢查子樹的部分即可。

最後在做 deletion 時，先以 binary search 尋找要刪除的 node 記為 now，如果不在 tree 裡面，就直接結束這個 function，如果找到了話，在尋找時順便紀錄 previous 代表誰指向 now，之後檢查 now 是否為 leaf，若是，則將 previous 指向他的 pointer 指向他的下一個，把 now 刪除後，維護 head 跟 tail 並跳出 function；
若非 leaf，檢查右子樹是否存在，若存在，在右子樹下尋找最左邊者，記為 change，同時維護 pri 代表指向 change 的 node，將 change 的數字與 now 對調，刪除 change，並把 change 的子樹補上 pri，若 change 沒有子樹，亦將 change 指向的下一個 node 補上，只是此時要小心維護 pri 的 is_threadr，最後維護 head 跟 tail 及 num – 1 就完成 delete 的動作了；
而若不存在右子樹，反之存在左子樹，則將對做相反著做即可。

Result:

```
[calee0219@linux2 ~ ] ./a.out test1.txt
Change! Change myself into a cute mahou shoujo!!
The path: 2 3 5 6
Back! Back to the original life!!
The reverse path: 6 5 3 2
[calee0219@linux2 ~ ] ./a.out test2.txt
Change! Change myself into a cute mahou shoujo!!
The path: 1 2 3 4 5 6 7 8 9 10 11 12
Back! Back to the original life!!
The reverse path: 12
[calee0219@linux2 ~ ] ./a.out test3.txt
Change! Change myself into a cute mahou shoujo!!
The path: 1 2 3 4 5 6 7 8 9 10 11 12
Back! Back to the original life!!
The reverse path: 1
[calee0219@linux2 ~ ] ./a.out test4.txt
Change! Change myself into a cute mahou shoujo!!
The path: 14 16 23 24 29
Back! Back to the original life!!
The reverse path: 29 24 23 16 14
[calee0219@linux2 ~ ]
```