数据结构 PJ1(红黑树&B 树)说明文档

运行说明

- 分成了两个文件 PJ1_RB_tree.cpp & PJ1_B_tree.cpp
- 每个文件内部按照类似“面测明细”中给出的 template 模板来写，main 函数中使用 switch case 的无限循环。输入 **0–7 代表 8 种要求的操作，输入 20 代表退出循环。**
- 另外，创建这两棵树的操作在两个类的构造函数中直接实现，而不是在成员函数中实现。word 代表 key 值，key 的比较通过 string 的比较函数实现（重载的< > =运算符）

RB–Tree

Init

主要是文件处理

从文件中读入：用以下代码一行一行读入文件中的内容

```C++
fstream file_in(path);
string l;
while (getline(file_in, l))
  line.push_back(l);
```

并且使用指针以及一些 string 的函数将每一行中的三个值分离出来（按照空格），这样可以将这三个参数传入 Insert 函数中进行操作。

Insert

成员函数重载了两种类型的 Insert

Insert by command

将三个数据传入到树中，word 是 key 值。插入的时候先按照 word 放到树的叶节点，并且设置为红色，再通过 fixup 函数调整。fixup 函数可以将树调整成符合红黑树要求的结果，主要通过左旋和右旋得到。

另外，如果找节点的时候发现 word 值相同则输出 conflict

Insert by file

这里直接使用了 Init 函数，二者的操作其实是一样的，都有文件读入以及调用 Insert 函数的过程。

Search

这里调用了 search_node 函数查找 word，search_node 函数返回的是一个节点指针，如果返回的指针是 NIL 则输出 missing，否则输出该指针指向的节点的值。

search_node 函数从根节点开始向下寻找 word，通过比较大小判断向左找还是向右找，最终找到所在节点或者 NIL

Update

与 search_node 函数类似的，也是找到 word 所在的节点，如果找到则更新 value(part of speech,frequency）值，如为 NIL 则调用 Insert 函数插入。

Dump

调用 print_by_order 函数

print_by_order 函数是中序输出红黑树中的所有节点，是一个递归函数（与 lab5 中对 BST 的中序输出函数类似）

Delete

成员函数重载了两种类型的 Delete

Delete by command

先调用了 search_node 函数，如果没找到就直接输出 missing 并且返回，否则找到需要删除的节点。

如果左右儿子有 NIL 则直接删除，否则将该节点的后继放到该位置。

如果删除了黑色节点，需要调用 delete_fixup 函数，该函数针对可能出现的 4 种情况对红黑树进行调整，从而符合红黑树的性质。

Delete by file

读入文件的方式和 Init 一致， 读出参数之后调用 Delete 函数

B-Tree

Init

与红黑树的 Init 相同，读入文件

Insert

Insert by command

先检查根节点是否是满的。如果根节点是满的则分裂（调用 split_child 函数），将一个值提升到新的根节点中。之后执行 insert_nonfull 函数

insert_nonfull 函数：递归地向 B 树中插入 word 值，如果遇到 word 相等的情况则输出 conflict

split_child 函数：将子节点分裂并且将其中的一个值提升到父节点中。

Insert by file

与红黑树的 Init 相同，也是调用 Init 函数

Search

调用了 search_node 函数，查找含关键词 word 的节点。

search_node 函数从根节点开始，通过比较传入的 word 和树中节点 word 的大小进行搜索。如果查找到则输出值并且返回，如果查找到叶节点依然没有发现相等，则输出 missing。

Update

先使用类似于 search_node 的方法查找 word 的位置，如果找到则更新 value 值，如果到叶节点仍未找到则调用 Insert 函数插入

Dump

调用 order 函数，递归地从根节点开始输出一个节点的值，直到根节点结束。调用及循环的顺序能够保证按照字典序输出。

Delete

Delete by command

先检查根节点是不是只有一个关键字，如果是，检查其两个子节点是不是只有 t–1 个，如果是则调用 merge_child 函数（将两个子节点合并，并且父节点下到子节点中），如果不是则调用 delete_nonone 函数。

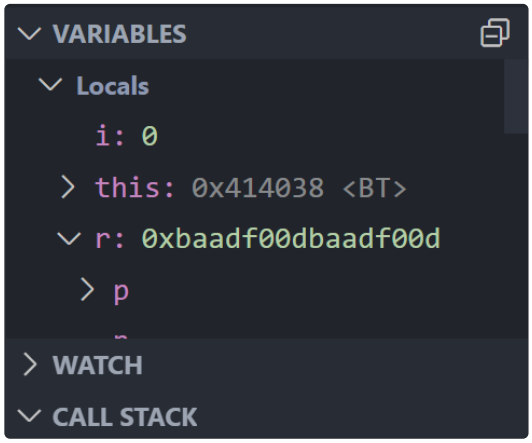
delete_nonone 函数：递归删除关键词 word，如果是叶节点则按顺序查找，如果没有则输出 missing，如果有就直接删除，并且所有关键词前移。如果是在内部节点中，要考虑 2a, 2b, 2c 的 3 种情况。其中还需要调用到查找前驱和后继的叶节点，将其放到该位置上。如果不在这个内部节点上，则需要继续找到一个至少有 t 个关键字的节点进行 delete_nonone 操作，其中有 3a, 3b 的两种情况。

Delete by file

读入文件的方式和 Init 一致， 读出参数之后调用 Delete 函数

总结

- 本次 PJ 的代码主要依据《Introduction to Algorithm》这本书上关于红黑树和 B 树的操作讲解，以及书上的伪代码，在自己打完这个 PJ 之后能够更加清晰地了解到这两种数据结构是怎样进行以上运算的。
- 在写 PJ 的过程中也遇到了很多问题（特别是 B 树），主要解决方法是：查看在 vscode 上调试程序中得到的参数值，按照参数值将这棵树画在纸上，并且一步步查看操作步骤，对比理想中正确的操作，发现 bug，调整操作。在这个过程中提升了自己的 debug 水平，以及对 vscode 的操作更加熟悉。
- debug 的过程中经常看到 0xbaadf00dbaadf00d，感到十分疑惑，查资料之后发现是指针未初始化所致，指向了“bad food”的未知地址。因此在使用指针的过程中应该需要多多注意初始化的问题以及边界条件。



- 依然需要指出的不足是：关于 free 内存的代码在本程序中有部分缺失（尽管可能不会对此次运行产生影响），这是一个关键的问题，很有必要解决；另外程序的封装存在一些问题，有些函数和成员变量可能不需要暴露，以及使用全局变量的问题，程序的封装性还可以提高。