Report - Project1

黄子骕 21307130013

RB-Tree

基本结构

- ***key值的选定***：由于此次数据的格式为 `word-(part of speech,frequency)`，因此很难得转换成数字的key值（容易产生冲突）。这里考虑到字符串本身是可以比较大小的，因此可以直接使用字符串作为key值，因为只有两字符串相同时，它们才是相等的。此时经过试验又发现，在本次给出的数据中，存在 *word* 相同，但 *part of speech* 不同的情况，因此这里将 *word* 与 *part of speech* 合并在一起作为key值，其对应的 *frequency* 作为value值。
- ***结点的设置***：令每个结点储存以下变量
  - left：左孩子指针
  - right：右孩子指针
  - parent：父结点指针
  - wordNpos：起与key值相同的作用，为一个字符串，内容为 *word* + " " + *part of speech*
  - value：一个string，保存 *frequency*

rbt类的实现

- ***辅助功能函数***
  - **rbt::get_root**：返回指向root结点的指针
  - **rbt::update_root**：更新指向root结点的指针
  - **rbt::left_rotate、rbt::right_rotate**：二者实现了左旋、右旋功能，其中特别注意处理当旋转相关结点为nullptr的情况
  - **rbt::test**：测试该树是否符合红黑树的所有性质，若符合则提示 `"Test passed!"`，若有不符合则会提示相应错误
  - **rbt::sub_test**：实现了用递归方法判断树中每条路径黑色结点数量是否相同、是否出现连续红色结点的性质，仅在 **rbt::test** 中被调用
  - **rbt::split**：传入一个string，并将该string分割为两部分：*wordNpos*、*frequency*，并按顺序存在一个vector中返回。用于处理文件中的数据。
  - **rbt::find_successor**：传入一个指向结点的指针，返回指向该结点的后继结点的指针。
- ***Insert功能相关函数***
  - **rbt::insert**：实现了插入结点的基本功能，传入一个指向结点的指针，若插入成功返回*true*，若插入失败返回*false*并提示 `"key xxx conflict!"`
    - 先按与普通二叉搜索树相同方法插入结点
    - 后进行调整：分两种case讨论（父结点是左/右孩子），每个case可以分为三个subcase
      - *subcase1*（父、叔叔结点都为红）：将父、祖父结点变色，并向上继续处理
      - *subcase2*（父结点红，叔叔结点黑，且当前结点与父结点异侧）：左旋/右旋（根据处于哪种case中来判断），转换为*subcase3*

        - *subcase3*（父结点红，叔叔结点黑，且当前结点与父结点同侧）：右旋/左旋（根据处于哪种case中来判断），并进行变色操作，结束调整。
    - **rbt::insert_by_command**：传入一个格式为 `"wordNpos frequency"` 格式的string，如 `"toughie N 26.0"`，并创建一个保存相应数据的结点，调用 **rbt::insert** 进行插入。若插入成功则提示 `"Successfully inserted the key-value pair: xxx"`
    - **rbt::insert_by_file**：传入一个string，内容为想要用来插入的文件的文件名（最好保证在与程序在同一文件夹下，否则需要额外添加路径）。对于相应文件的每一行（除表示attribute的第一行外），调用 **rbt::split** 处理数据，并创建保存该数据的结点，调用 **rbt::insert** 插入。若全部插入成功，则提示 `"Insert_by_file: xxx.txt completed!"`，若有 *key* 值冲突，则会对冲突数据提示 `"key xxx conflict!"`

- ***rbt::search***：传入一个格式为 `"word pos"` 格式的string，如 `"toughie N"`，实现了通过给定key值搜索对应结点并打印其value的功能，实现方法与普通二叉搜索树相同，通过不断比较key值来最终确定是否搜索到对应结点以及往左子树走还是往右子树走。若成功搜索到，则返回指向目标结点的指针并提示 `"Successfully found the wordNpos: xxx, the corresponding frequency is: xxx"`，若无目标key值，则返回 *nullptr* 并提示 `"Key: xxx missing!"`

- ***Delete功能相关函数***:

    - **rbt::delete_by_command**：传入一个格式为 `"word pos"` 格式的string，如 `"toughie N`，实现了删除给定key值的结点的功能
        - 首先调用 **rbt::search** 查找该节点，此时对于查找结果会有提示，若找到则继续删除，若无该结点则返回*false*，删除成功提示 `"Successfully deleted wordNpos: xxx!"`
        - 在找到相应结点的情况下，分三种case进行处理
            - *case1*：待删除结点有两个非空子结点，则找到后继结点，将后继结点作为待处理结点（必定至多只有一个孩子），转换为case2或case3
            - *case2*：待删除结点只有一个子结点，此时分只有左孩子与只有右孩子来讨论，在这个情况下直接删除并处理孩子即可
            - *case3*：待删除结点无子结点，此时分为以下两个subcase
                - *subcase1*：待删除结点为红色，则直接删除
                - *subcase2*：待删除结点为黑色，此时一定有兄弟结点，分为以下四个situation（分待删除结点为左/右孩子的情况讨论，以下以带删除结点为左孩子为例）
                    - *situation1*：兄弟结点为红色，则以父结点为轴，左旋并颜色调整
                    - *situation2*：兄弟结点为黑色，且两个子结点都为黑色（空也视为黑色），此时进行颜色调整并将父结点作为新的当前结点继续向上调整
                    - *situation3*：兄弟结点为黑色，且左子结点为红色，右子结点为黑色，此时以兄弟结点为轴右旋并颜色调整
                    - *situation4*：兄弟结点为黑色，且右子结点为红色，此时以父结点为轴左旋并颜色调整
                    - *调整的退出*:

situation	处理后的situation	说明
1	2 or 3 or 4	继续调整
2	1 or 2 or 3 or 4	若新的当前结点为红色，则变黑后直接退出；若为黑色则继续调整
3	4	继续调整

situation	处理后的situation	说明
4	/	退出

- - **rbt::delete_by_file**：传入一个string，内容为想要用来插入的文件的文件名（最好保证在与程序在同一文件夹下，否则需要额外添加路径）。对于相应文件的每一行（除表示attribute的第一行外），调用 **rbt::split** 处理数据，接着调用 **rbt::delete_by_command** 对相应key的结点进行删除，在过程中对于每一个结点的删除过程都会有提示

```
若删除正常：
"Successfully found the wordNpos: xxx, the corresponding frequency is: xxx"
"Successfully deleted wordNpos: xxx!"
若找不到该节点：
"Key: xxx  missing!"
```

- **rbt::update_by_command**：传入两个string，第一个string格式为`"word pos"`，如`"toughie N`，表示想要update的结点key值；第二个string格式为`"wordNpos frequency"`，如`"toughie N 26.0"`，表示想要更新的数据
  - 首先调用 **rbt::search** 查找是否存在key值相同结点，并提示查找结果
  - 若有，则删除原结点；若无则不作操作
  - 调用 **rbt::insert_by_command** 插入更新的结点

```
若已有key值相同的结点，且更新的结点信息不产生冲突则提示：
"Updating the key-value pair......"
"Successfully deleted wordNpos: xxx!"
"Successfully inserted the key-value pair:  xxx"
若无key值相同的结点，且更新的结点信息不产生冲突则提示：
"Inserting the key-value pair......"
"Successfully inserted the key-value pair:  xxx"
若更新结点的信息出现冲突，则提示错误（参照 rbt::insert_by_command 函数）
```

- **rbt::initialize**：传入一个string，内容为想要用来插入的文件的文件名（最好保证在与程序在同一文件夹下，否则需要额外添加路径）。对于相应文件的每一行（除表示attribute的第一行外），调用 **rbt::split** 处理数据并创建保存该数据的结点，调用 **rbt::insert** 插入。若全部插入成功，则提示`"Initialize_by_file: xxx.txt completed!"`，若行信息插入发生问题则会提示（参照 **rbt::insert**）

- **rbt::dump**：传入一个指向结点的指针，用中序遍历（由于利用了string做为key值，因此中序遍历时自动按照字典序）遍历整个以传入结点为根结点的树并打印每个结点的 *key-value*
  - 若要显示所有结点的的信息，用以下形式即可

```
tree.dump(tree.get_root());
```

B-Tree

基本结构

- ***key值的选定***：同 RBT
- ***key_value结构***：有两个成员，分别保存key与value
- ***结点的设置***：
    - **keys_num**：保存结点中key个数
    - **pairs**：保存结点中所有格key-value pair
    - **leaf**：记录当前结点是否为叶结点
    - **children**：保存结点中所有的指向孩子的指针

bt类的实现

- ***辅助功能函数***

    - **bt::get_root**：返回指向root结点的指针
    - **bt::update_root**：更新指向root结点的指针
    - **bt::split**：传入一个string，并将该string分割为两部分：*wordNpos*、*frequency*，并按顺序存在一个vector中返回。用于处理文件中的数据。
    - **bt::get_minimum_degree**：返回minimum_degree
    - **bt::find_successcor**：找到目标 key 值的后继
    - **bt::find_predecessor**：找到目标 key 值的前驱
    - **bt::test_internal_node**：检查目标 key 是否在当前结点中，若在则返回 true，若不在则返回 false

- ***Insert功能相关函数***

    - **bt::insert_nonfull**：递归寻找插入位置（一定插入在叶子结点上），若下一个搜索目标孩子已满则需要调用spilt_child进行分裂
    - **bt::split_child**：创造一个新的结点，并将原来的目标结点分裂为两个结点，且将中间key上移至当前结点
    - **bt::insert_by_command**：输入待插入的数据
        - 若为空树，则创建新结点并将其作为根结点
        - 查找树中是否有该key，若有则提示 conflict
        - 插入过程
            - 从根结点开始检查，若满则需调用 **bt::split_child**
            - 调用 **bt::insert_nonfull** 递归找到在叶结点中的合适位置进行插入（中途若有对应结点满的情况也需要调用 **bt::split_child** ）
    - **bt::insert_by_file**：输入文件路径并插入文件中的数据，即读取每一行数据并使用 **bt::insert_by_command**，与rbt中类似

- ***Delete功能相关函数***

    - **bt::_delete**：递归实现删除功能，分为3个case
        - 先在当前结点找到大于等于key值的最小元素（下一次寻找的位置或是目标本身位置）
        - case1：目标key在当前结点且当前结点为叶结点，直接删除
        - case2：目标key在当前结点且当前结点为内结点
            - case2a：左孩子至少有 t 个 key，则找其前驱并删除
            - case2b：右孩子至少有 t 个 key，则找其后继并删除

            - case2c：左右孩子都只有k t-1 个 key，则进行merge操作，将当前结点中对应的key下移并与左右孩子合并
        - case3：目标key不在当前结点
            - 若当前结点为叶结点，则说明无此key，进行提示
            - 若当前结点为内结点，则一定有一个包含目标key值的子树
                - case3a：目标结点只有 t-1 个 key，相邻兄弟至少有一个有至少 t 个key，此时向有多余 key 的兄弟借一个 key
                - case3b：目标结点只有 t-1 个 key，相邻兄弟都只有 t-1 个key，此时将当前结点中对应的 key 下移，并与相邻兄弟其中一个合并
        - 递归地对下一个处理的目标结点继续调用 **bt::_delete**
    - **bt::deleta_by_command**：传入key值，先搜索是否有该 key，若有则调用 **_delete** 进行删除并提示，若无则直接提示无该 key
    - **bt::delete_by_file**：输入文件路径并插删除树中包含于文件的数据，即读取每一行数据并使用 **bt::delete_by_command**，与rbt中类似

- ***bt::initialize***：实现初始化，传入文件路径，并对其中每一行的数据调用 **bt::insert_by_command** 进行插入

- ***search相关函数***

    - **bt::search**：传入目标 key 值进行查找，并对查找结果进行提示，其中功能通过调用 **bt::sub_search** 实现
    - **bt::sub_search**：对于每一个结点通过递归寻找，若找到或到达叶结点则返回

- ***bt::updata_by_command***：实现对于目标数据的更新，传入需要更新的目标 key 值以及新的数据，先查找是否有该 key，若有则将其删除并插入新数据，若无则直接插入新数据

- ***bt::dump***：通过递归遍历所有结点中的数据并且按照字典序打印数据

    - 若要显示所有结点的的信息，用以下形式即可

```
tree.dump(tree.get_root());
```