PJ1--Translator

第三方库

PyQt5--->用于ui

使用说明

安装第三方库后，直接运行 **luncher.py** 即可运行ui
analysistest.py ,是一个简单测试脚本，测试其在import三个文件的效率 多次运行后 去均值 计入csv文件

Analysis

基于多次测试后，结果存储于data-B.csv data-BR.csv

type	initilize	delete	insert
b	0.005864500009920448	0.00011140003334730864	0.000242199981585145
br	0.0112496999790892	0.00024060002760961652	0.0002460000105202198

由上方数据可知，b树在测试样例中总体效率更高,由于b树的删除功能未能完全实现，删除时间仅供参考

原因分析：

- B树和红黑树的各项操作复杂度均为O(lg n),难以从数量级角度分析
- B树在3000数据量下，树高低于红黑树，所以B树在查询，删除，插入等操作时更多的是去做比较操作，而不是寻址；红黑树需要更多次数的寻址，所以所需时间较长
- B树的各个节点具有更大的容纳量，不容易破坏树的平衡性，需要调整平衡的次数小于红黑树需要调整平衡的次数
- 而后续小规模插入操作，二者的差距缩小，说明b树在数据量大的情况下，更占优势，小数据量时，需要更多调整平衡的操作；红黑树在数据规模变大时，需要调整的次数增多，效率下降

UI设计

基于OOP思想，采用前后端分离的设计思路，将程序主体分为前端UI，后端两种树的Model，以及连接前后端的control
前端响应事件通过control来传递到Model，后端信息再传递回前端UI

controller

```
#初始化方法
def __init__(self):
#当ui界面的树的类型选择发生变化调用此
def changetype(self,n):
#响应import按钮，将遍历结果写入树内的result，返回result
def importfrom(self,_path):
#响应translate按钮，返回翻译结果
def search(self,key):
#响应delete按钮
def delete(self,key):
#响应add按钮
def add(self,key,value):
#响应submit按钮，将搜索结果树内的result，返回result
def searchByRange(self,_from,to):
```

model

为两棵树
基本功能见 B_Tree.py BR_Tree.py

文档中要求的检查

B树插入检查

```
def insert(self, k):
        if self.searchByEng(k)==None:
            return
```

B树删除检查
B树删除，即是在先寻找，如果到了叶节点还是没有找到，会自动退出

BR树插入检查

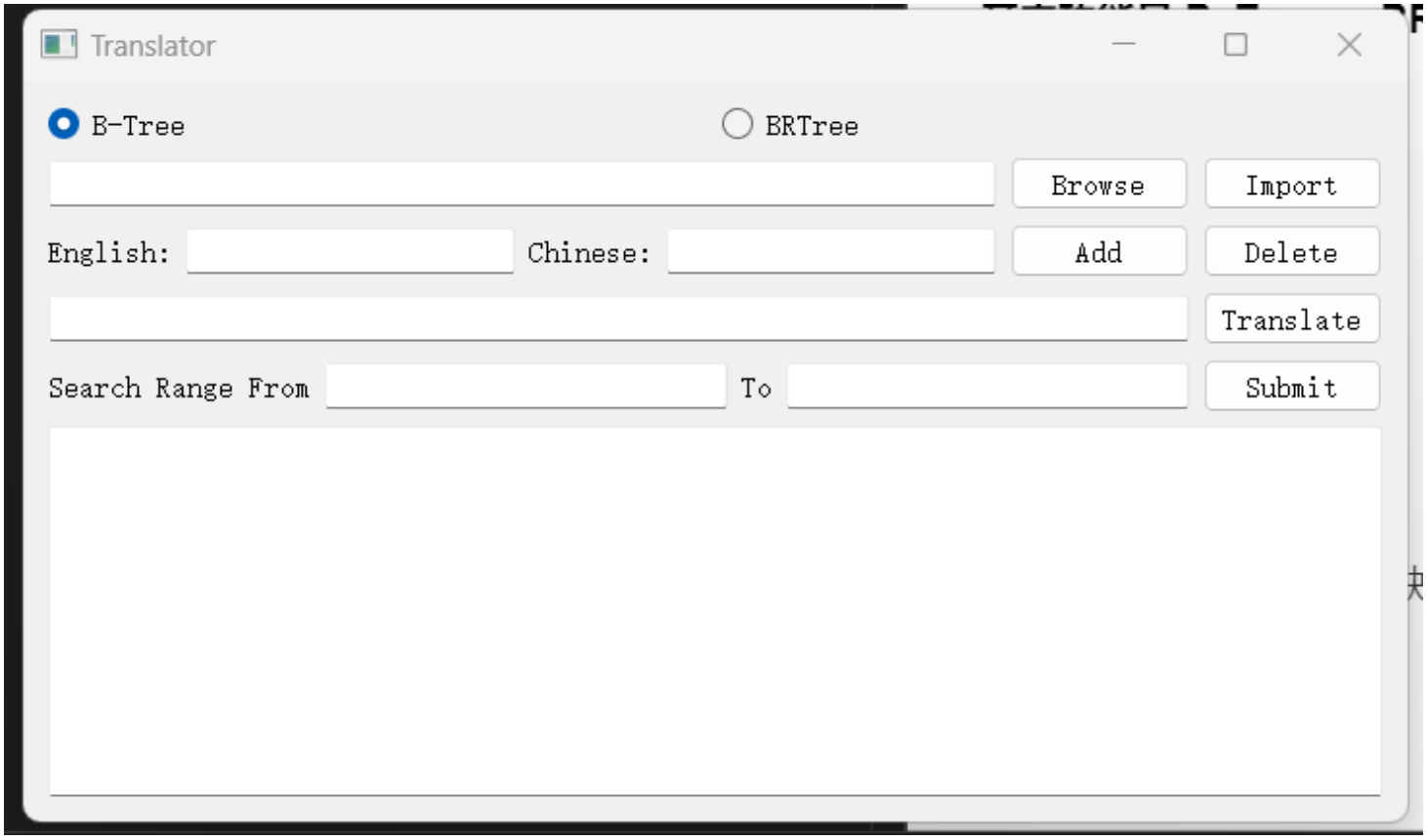
```
def insert(self, keys):

        key=keys[0]
        value=key[1]
        if self.search(key)==self.nil:
            return
```

BR树删除检查

```
        node_to_delete = self.search(key)
        if node_to_delete == self.nil:
            return
```

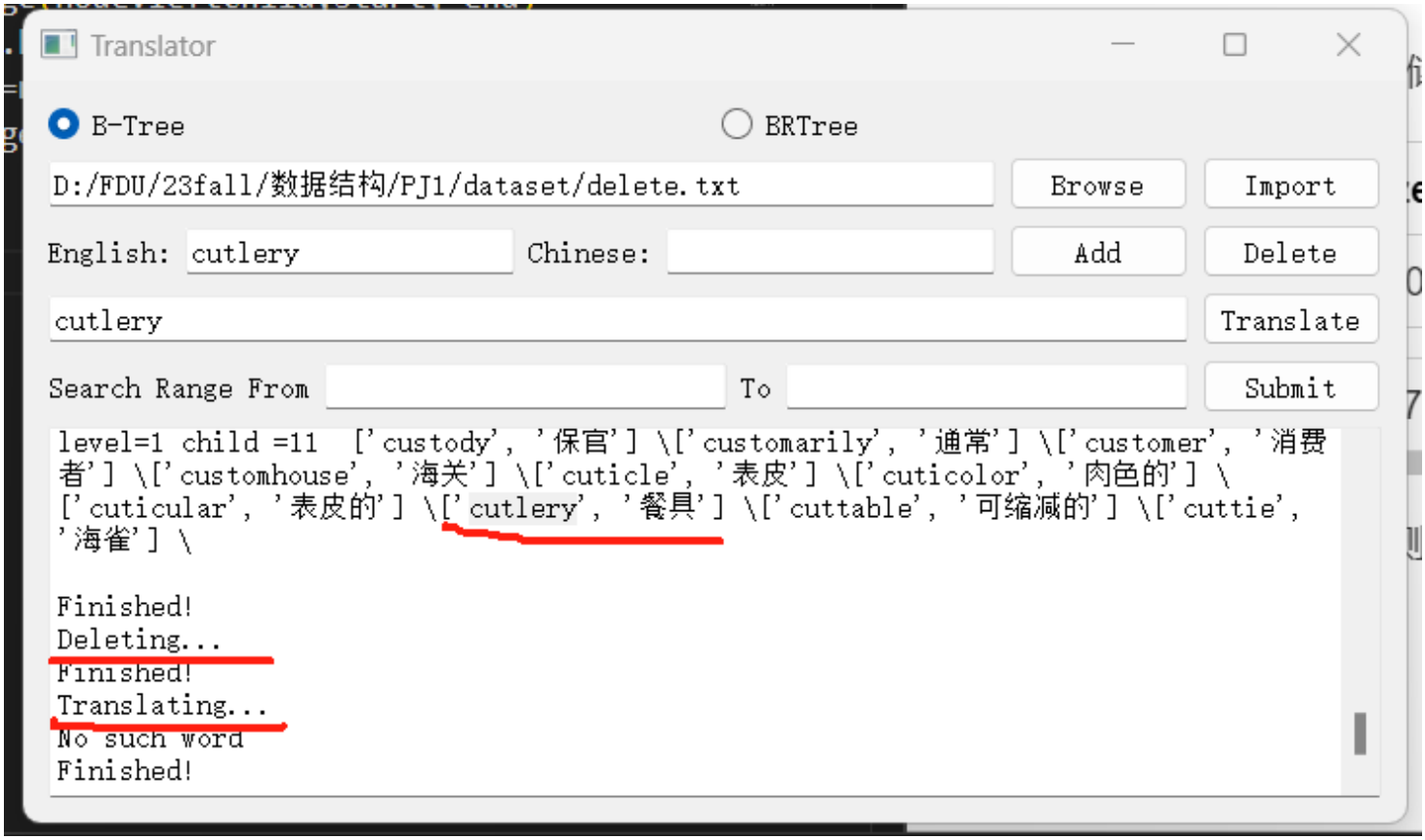
UI



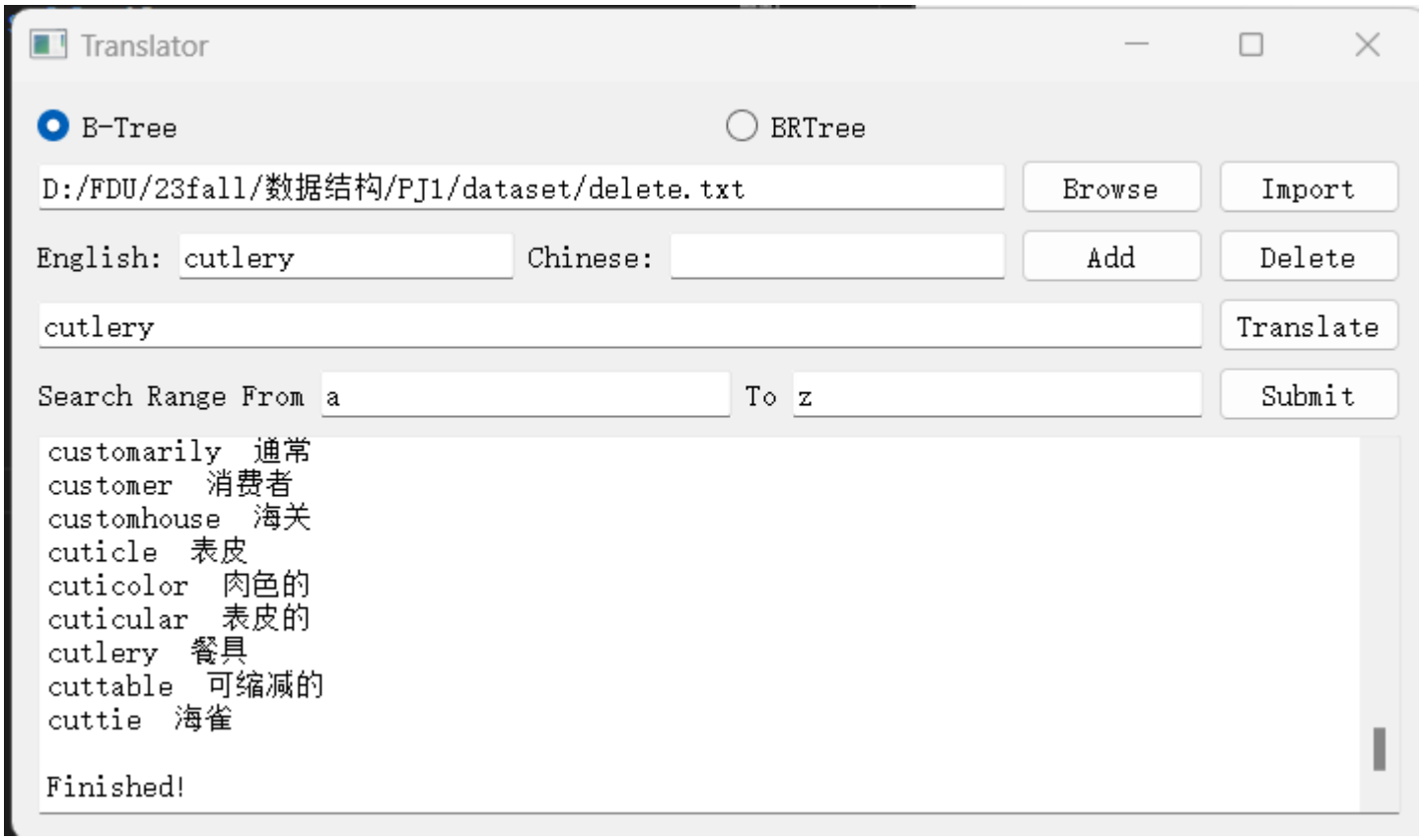
结果展示

b树初始化

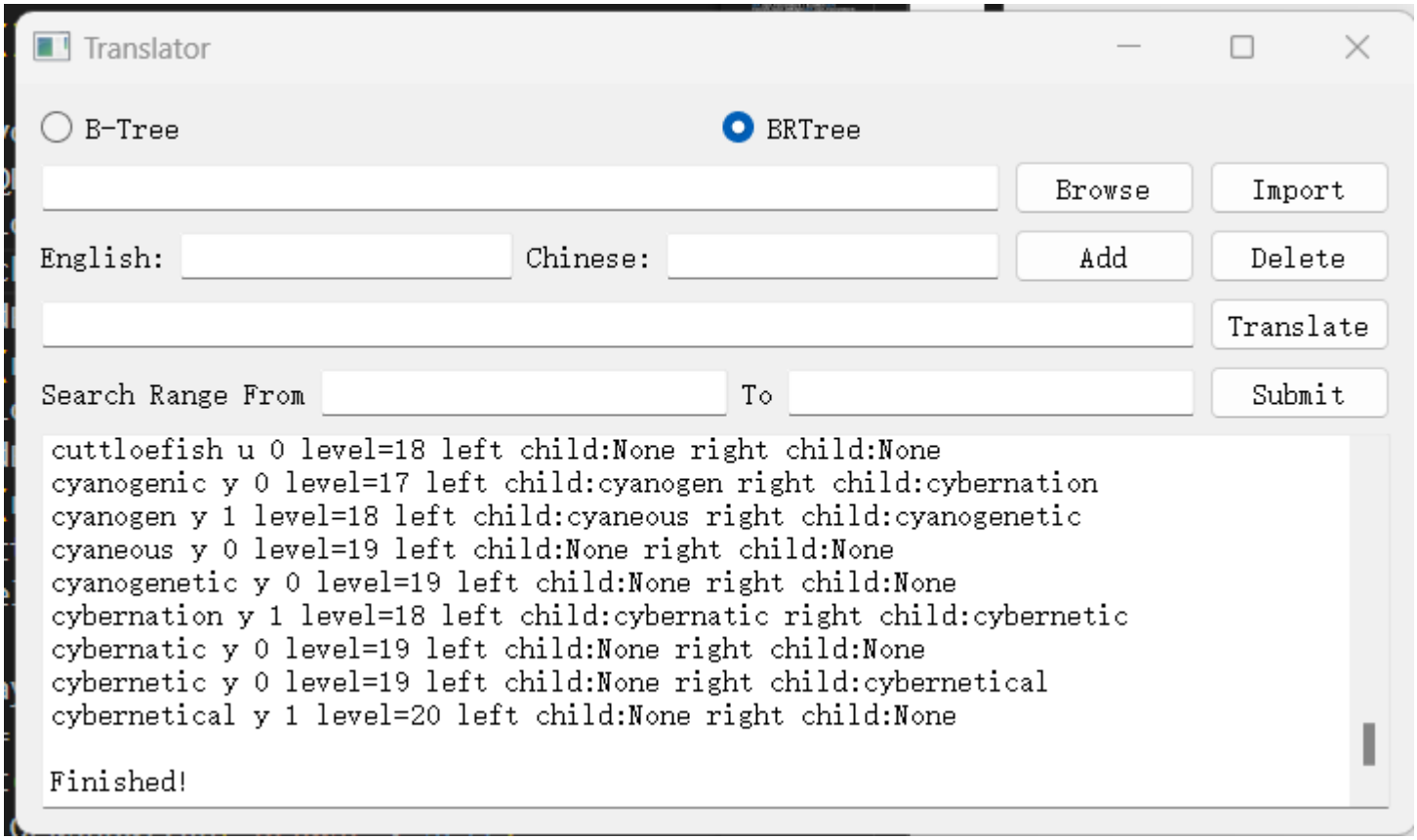
b树删除后无法查询到单词



range查询

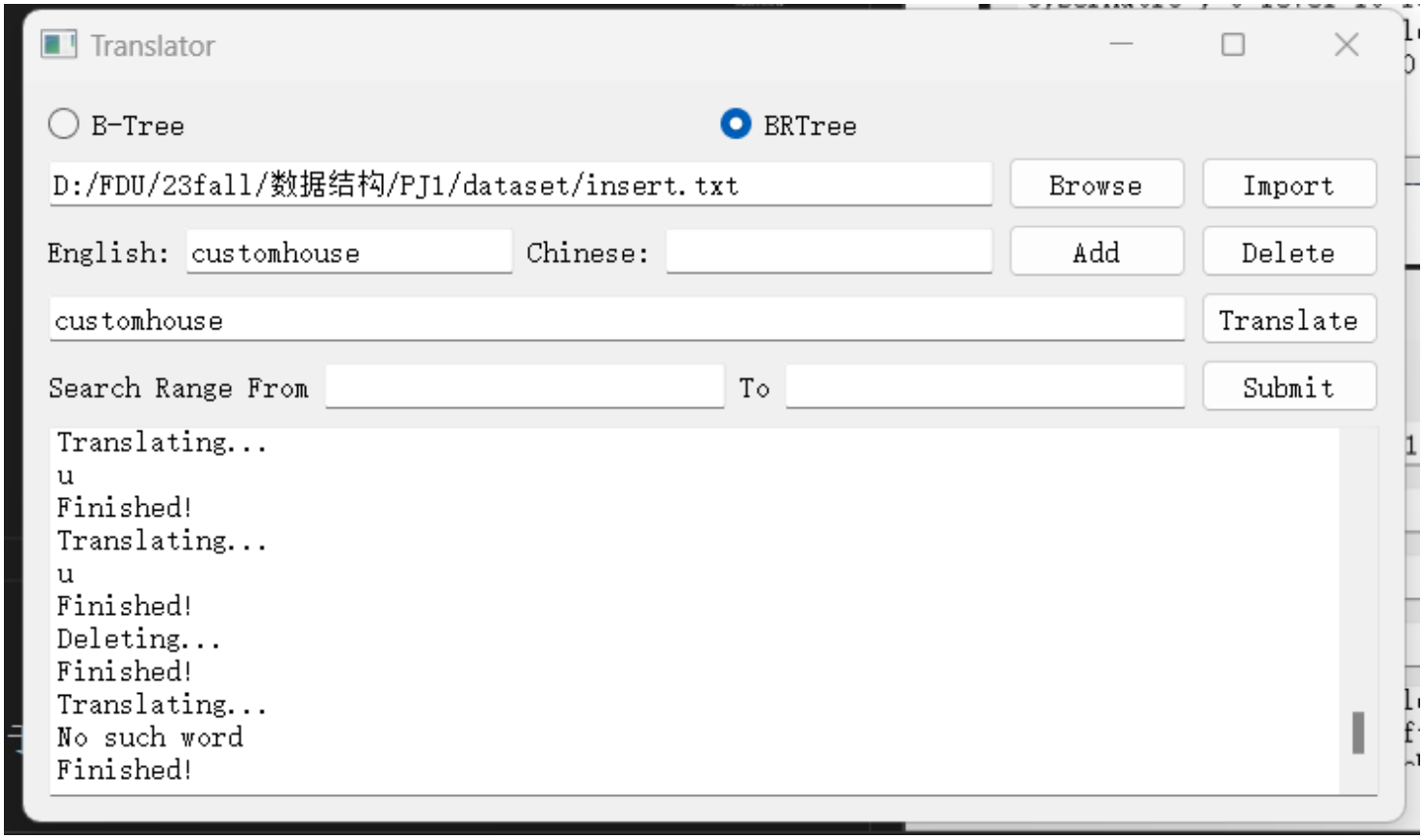


br树初始化

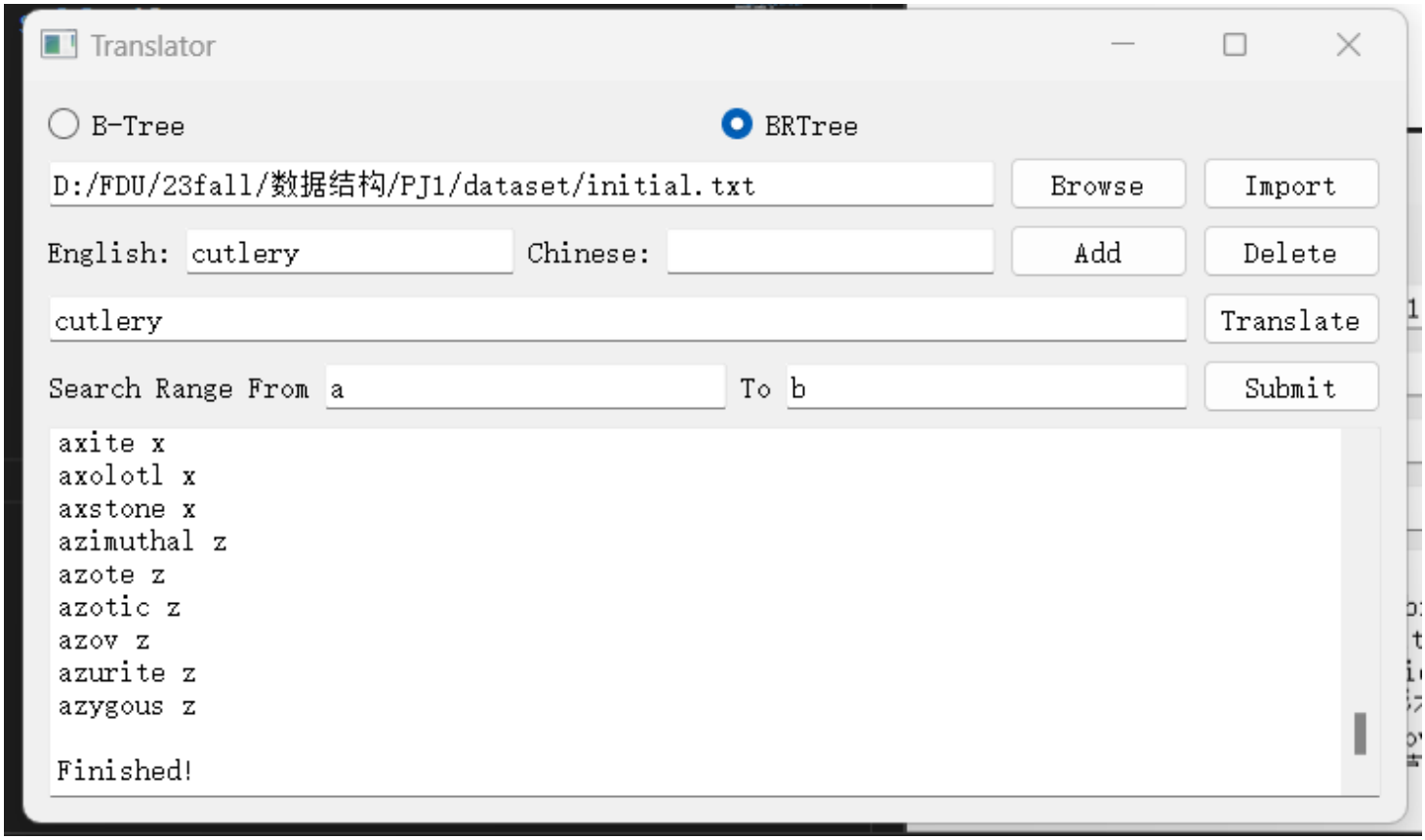


br树删除

br树删除后查询



br range 查询



缺陷

- b树的删除功能仍有缺陷，但由于递归次数过多，无法精确定位bug，放弃修改，未完全实现删除功能，在import delete.txt后，仍然有单词能被查询
- ui界面没有错误提示，遇到error会直接退出