**上 海 大 学**

**2019-2020学年冬季学期**

**《数据结构（2）》实验报告**

实 验 组 号： 01

上 课 老 师： 沈 俊

小 组 成 绩：

小组成员成绩表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 学号 | 姓名 | 贡献因子 | 成绩 |
| 1 | 17120193 | 张赛 | 20 |  |
| 2 | 18120189 | 林艺珺 | 20 |  |
| 3 | 18120206 | 徐奕婷 | 20 |  |
| 4 | 18120212 | 叶菁 | 20 |  |
| 5 | 18120221 | 陈思文 | 20 |  |

注：小组所有成员的贡献因子之和为100.

计算机工程与科学学院

2020年03月10日

实验三 查找

一．实验内容

（一）设计性实验：B树的设计与实现

根据教材介绍的3-阶B树，完成其数据结构的设计，实现插入、删除、查找等算法，并进行测试验证和性能分析。

（二）综合性实验：电子词典

[问题描述] 英汉词典作为一个常用的学习工具，是我们经常要使用的。现在要求根据英汉词典的功能，设计并实现一个简易的电子英汉词典，其中功能包括：单词的查找、增加、删除、修改等工作。

[基本要求]

(1) 在计算机中建立可扩充词汇量的电子英汉词典，每个单词词条包括：英文单词、词性、拼读音标、中文（汉语拼音）等。

(2) 对单词和释义字符串长度的限定分别为不少于20个词条。

(3) 系统显示这样的菜单：

电子英汉词典

(1)初始化词典

(2)添加词汇

(3)删除词汇

(4)查找词汇

(5)修改词汇

(0)退出系统

其中“初始化词典”将从词典文件(diction.in)中读入词典中的所有单词词条，“添加词汇”可以在词典中加入新的词条；“删除词汇”根据输入的英文单词从词典中删除相关词条;“查找词汇”根据输入的英文单词从词典中查找相关词条，输出该单词的词性、拼读音标、中文(汉语拼音)等信息;“修改词汇”根据输入的英文单词从词典中查找相关词条， 并根据输入的单词内容，修改相关信息。词典信息更新后在退出系统时修改词典文件。

二．主要算法设计

（一）设计性实验：B树的设计与实现

1. 问题分析与算法设计

B树是一种被称为查找树的树，与之类似的有二叉排序树、平衡二叉树等等。它们共同的特点是都按照一定规律排序并存储数据。

B树的基本操作是插入和删除操作。在进行插入和删除操作时，需要用到查找操作，而查找操作的思路主要利用了B树中数据是有序存放的原理，可以快速找到要插入的位置（即失败结点）或者要删除的结点位置。

我们通过插入操作来建立B树。

1) 查找关键字

从根结点开始，向子树结点中的关键字查找。如果找到，返回SUCCESS；否则返回FAIL。函数参数采用引用传递，便于记录相应的位置信息。

2) 插入关键字

通过查找操作确定关键字应插入的结点位置。然后分两种情况讨论：

　　1、该结点中关键字个数小于3，直接插入即可。

　　2、该结点中关键字个数等于3，需要分裂当前结点。以中间关键字为界将结点一分为二，将分裂结点的后半部分存储到新生成的结点中，并把中间关键字插入到双亲结点中。

重复上述步骤。

最坏情况是一直分裂到根结点，此时需建立一个新的根结点，整个B树增加一层。

3) 删除关键字

Ｂ树中关键字的删除比插入更复杂。

这个过程可以分两步完成：

1.找出关键字k所在的结点。然后判断它是否为叶子结点。

2.若该结点为非叶结点，且为该结点中第kn个关键字，则从指针pointer[kn-1]所指的子树中找出最大关键字K，代替被删关键字的位置，然后在叶结点中删去K。

这样就把在非叶结点删除关键字k的问题转变成删除叶子结点中的关键字的问题了。

同样根据不同情况作相应的处理，共有两种可能情况：

（1）如果被删关键字所在结点的原关键字个数n>=2，说明删去该关键字后该结点仍满足B树的定义。那么从该结点中直接删去关键字即可。

（2）如果被删关键字所在结点的关键字个数n=1，说明删去该关键字后该结点将不满足B树的定义，此时需要调整。

调整过程同样分为两种情况：

1.如果与该结点相邻的右（左）兄弟结点中的关键字个数大于1

将右（左）兄弟结点中最小（大）关键字上移至双亲结点

将双亲结点中小（大）于该上移关键字的关键字下移至被删关键字所在结点中

2如果与该结点相邻的右（左）兄弟结点中的关键字个数均等于1

把要删除关键字的结点与其左（右）兄弟结点以及双亲结点中分割二者的关键字合并成一个结点，到该删除关键字结点的左（右）兄弟结点中去。

如果合并后双亲结点中关键字个数小于1，则对此双亲结点做同样处理。

最坏情况是一直合并到根结点，整个B树减少一层。

2. 函数功能设计

|  |  |
| --- | --- |
| BTree主要成员函数 | |
| BTree(ElemType es[], int n); | 构造函数 |
| ~BTree(); | 析构函数 |
| Status Find(ElemType k, int& kn, BTreeNode<ElemType>\*& cur); | 查找关键字k |
| void Insert(ElemType k); | 插入关键字k |
| void NonSplitInsert(ElemType k, int kn, BTreeNode<ElemType>\* cur); | 不分裂结点插入k |
| void Split(BTreeNode<ElemType>\*& cur); | 分裂结点 |
| BTreeNode<ElemType>\* Parent(BTreeNode<ElemType>\* cur); | 求双亲结点 |
| void Delete(ElemType k); | 删除关键字k |
| void DeleteKey(BTreeNode<ElemType>\*& cur, int kn); | 删除叶子节点中元素 |
| void TakeKey(BTreeNode<ElemType>\*& cur); | 删除后调整关键字 |
| int DisplayTree(); | 输出B树 |

（二）综合性实验：电子词典

1. 问题分析与算法设计

本题是查找的单元内容的实际应用。词典的设计基于课本BinarySortTree类模板，但根据词典特性有部分修改。

本次选用二叉排序树来构造词典，二叉排序树作为一种基础的二叉树的数据结构，能适应动态的插入和删除操作。它的查找思路与B树相似，利用了二叉排序树中数据是有序存放的原理，可以快速找到要插入的位置或者要删除的词条位置。在查找操作中，除了判断单词word是否一致，还要确认词性type是否也相同，这样的改动使词典的插入和删除操作更加灵活。

在文件输入输出方面，通过中序遍历将二叉排序树中存储的信息写入txt文件，保证各词条在文件内是有序的，并且每次在调用可能修改信息的执行语句后，立即更新文件，避免数据丢失。

以下为Dictionary类查找的成员函数代码。

代码2.1 Find函数

|  |
| --- |
| 1. WordNode \*Dictionary::Find(const string &word, WordNode \*&f) const 2. { 3. WordNode \*p = GetRoot(); 4. f = NULL; 5. while (p != NULL) { // 查找关键字为word的结点 6. if (word < p->word) { // 关键字更小,在左子树上进行查找 7. f = p; 8. p = p->leftChild; 9. } 10. else if (word == p->word) { 11. cout << "请输入需要查找的词性："; 12. string t; 13. cin >> t; 14. if (t.length() < 5) { 15. for (int i = 5; i > t.length(); i--) { 16. string b = " "; 17. t = b + t; 18. } 19. } 20. string k = word + t; 21. return FindInit(k, f); 22. } 23. else { // 关键字更大,在右子树上进行查找 24. f = p; 25. p = p->rightChild; 26. } 27. } 28. return p; 29. } 30. WordNode \*Dictionary::Find(const string &word) const 31. { 32. WordNode \*f; // 指向被查找结点的双亲 33. return Find(word, f); 34. } |

代码2.2 FindInit函数

|  |
| --- |
| 1. WordNode \*Dictionary::FindInit(const string &key, WordNode \*&f) const 2. { 3. WordNode \*p = GetRoot(); 4. f = NULL; 5. while (p != NULL && key != p->key) { // 查找关键字为key的结点 6. if (key < p->key) { // key更小,在左子树上进行查找 7. f = p; 8. p = p->leftChild; 9. } 10. else { // key更大,在右子树上进行查找 11. f = p; 12. p = p->rightChild; 13. } 14. } 15. return p; 16. } 17. WordNode \*Dictionary::FindInit(const string &key) const 18. { 19. WordNode \*f; // 指向被查找结点的双亲 20. return FindInit(key, f); 21. } |

2. 函数功能设计

|  |  |
| --- | --- |
| Dictionary类主要成员函数（即电子词典主要功能函数） | |
| Dictionary(WordNode \*r); | 建立以r为根的电子词典二叉排序树 |
| bool Insert(const string &w, string &t, const string &p, const string &c); | 插入单词词条 |
| bool Delete(const string &key); | 删除单词词条 |
| WordNode \*GetRoot() const; | 求电子词典二叉排序树的根 |
| bool IsEmpty() const; | 判断电子词典是否为空 |
| Status SetType(WordNode \*p, string &e); | 修改单词词性 |
| Status SetPron(WordNode \*p, const string &e); | 修改单词音标 |
| Status SetChn(WordNode \*p, const string &e); | 修改单词中文释义 |
| int NodeCount() const; | 单词数统计 |
| WordNode \*Find(const string &word) const; | 按照英文单词查找 |
| WordNode \*FindInit(const string &key) const; | 按照唯一关键字查找 |
| void Print() const; | 按字母a-z顺序输出电子词典 |
| void Destroy(WordNode \* &r); | 销毁以r为根的电子词典二叉排序树 |
| virtual ~Dictionary(); | 析构函数 |
| 文件输入输出函数 | |
| bool SaveHelp(ofstream &SaveFile, WordNode \*r); | 保存到文件帮助 |
| bool Save(char \*filename); | 保存到文件 |

三．主要数据组织

（一）设计性实验：B树的设计与实现

代码3.1

|  |
| --- |
| 1. **struct** BTreeNode 2. {//数据成员： 3. int num;//关键字个数 4. ElemType key[ORDER+1];//关键字数组 5. BTreeNode<ElemType> \*pointer[ORDER+1];//指向子树的指针数组 6. //构造函数： 7. BTreeNode(); 8. }; 9. **class** BTree 10. { 11. protected: 12. BTreeNode<ElemType> \*root;//指向B树根结点的指针 13. public: 14. BTree(ElemType es[], int n); //构造函数 15. ……//其他功能函数此处省略 16. }; |

（二）综合性实验：电子词典

1. 数据文件格式

数据文件默认名称为“XXXXXXXX.txt”，文件中有多条数据。每条数据其中包括了英文单词本身、该词条英文单词词性、国际音标和中文释义。格式如图3-1所示。

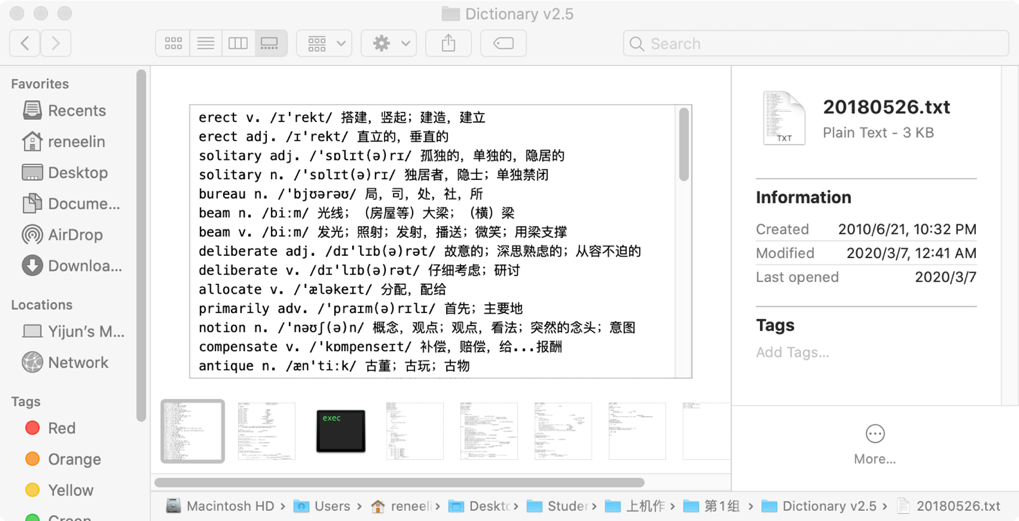


图3-1 输入信息

2. 数据结构设计

实验内容涉及到多条记录的处理，每条记录含有均含有string类的数据项，使用结构体存储信息较为方便，且二叉排序树中对于关键字有互不相同的要求，因此，构造新的string类关键字并作为结构体的信息存储，数据结构设计如下：

代码3.2

|  |
| --- |
| 1. #include <string> 2. using namespace std; 3. **struct** WordNode 4. { 5. string key; //根据单词及词性构造的关键字 6. string word; //英文单词 7. string type; //该词条英文单词词性 8. string pron; //国际音标 9. string chn; //中文释义 10. WordNode \*leftChild; // 左孩子指针域 11. WordNode \*rightChild; // 右孩子指针域 12. WordNode() //构造函数 13. { leftChild = rightChild = NULL;} 14. WordNode(const string &w, const string &t, const string &p, const string &c, WordNode \*lChild = NULL, WordNode \*rChild = NULL) 15. { 16. word = w; 17. type = t; 18. if (t.length() < 5) { 19. for (int i = 5; i > t.length(); i--) { 20. string b = " "; 21. type = b + type; 22. } 23. } 24. key = w + t; //通过填补词性至5位加上单词本身构造新的关键字 25. pron = p; 26. chn = c; 27. leftChild = lChild; 28. rightChild = rChild; 29. } 30. }; |

四．测试分析

（一）设计性实验：B树的设计与实现

在3阶B树的查找过程中，耗费的时间在结点内查找和从外存读取结点上。树的高度H为查找时最多读取外存的次数。就样例1来说，在等概率的情况下，平均读外存次数为(1+2\*3+3\*7)/11。当3阶B树是完全二叉树时，其读外存性能最差为log2n；当3阶B树每个结点的关键字个数均为2时，其读外存性能最好为log2（n/2）。由于读写外存的速度相较内存结点内查找慢得多，所以在每个结点关键字个数越大，树的高度越小的情况下，3阶B树的查找性能越好。

1. 构造函数验证

样例1 普通B树

|  |
| --- |
| 1. **int** es[] = {41,26,74,82,17,35,53,60,78,87,94}; 2. **int** n = 11; |

样例1 测试结果

A screenshot of a cell phone

Description automatically generated

图4-1-1(1) 样例1测试

样例2 结点关键字个数都为1的B树

|  |
| --- |
| 1. **int** es[] = {69,43,38,47,78,63,56}; 2. **int** n = 7; |

样例2 测试结果

A screenshot of a cell phone

Description automatically generated

图4-1-1(2) 样例2测试

样例3 结点关键字个数都为2的B树

|  |
| --- |
| 1. **int** es[] = {56,34,77,45,27,60,84,32,39,93,102,65,90,108,37,38,43,113,   121,100,110,73,63,75,112,130};   1. **int** n = 26; |

样例3 测试结果

A screenshot of a cell phone

Description automatically generated

图4-1-1(3) 样例3测试

样例4 单结点B树

|  |
| --- |
| 1. **int** es[] = {6}; 2. **int** n = 1; |

样例4 测试结果

A close up of a logo

Description automatically generated

图4-1-1(4) 样例4测试

样例5 空B树

|  |
| --- |
| 1. **int** es[] = {}; 2. **int** n = 0; |

样例5 测试结果

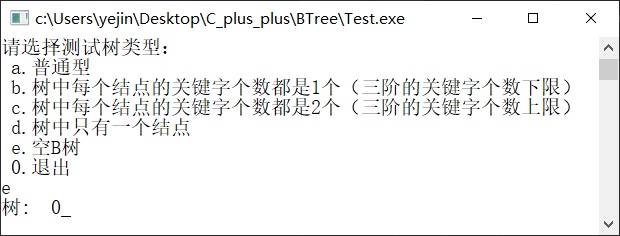


图4-1-1(5) 样例5测试

2.功能函数验证

以“1. 构造函数验证”中样例1为例进行功能函数验证

（1）查找关键字

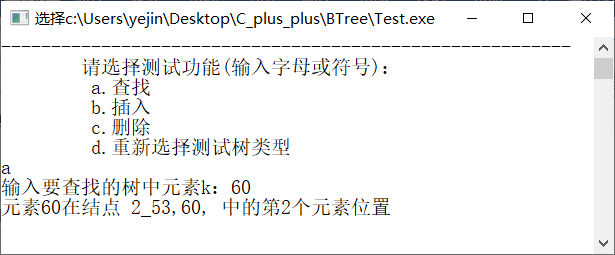


图4-1-2(1) 查找关键字

（2）查找不存在的关键字

A screenshot of a cell phone

Description automatically generated

图4-1-2(2) 查找不存在的关键字

（3）插入关键字

A screenshot of a cell phone

Description automatically generated

图4-1-2(3) 插入关键字

（4）插入已存在的关键字

A screenshot of a cell phone

Description automatically generated

图4-1-2(4) 插入已存在的关键字

（5）插入关键字到关键字个数为上限的结点中

A picture containing screenshot

Description automatically generated

图4-1-2(5) 插入关键字到关键字个数为上限的结点中

（6）删除关键字

A screenshot of a cell phone

Description automatically generated

图4-1-2(6) 删除关键字

（7）删除不存在的关键字

A screenshot of a cell phone

Description automatically generated

图4-1-2(7) 删除不存在的关键字

（8）删除关键字个数为下限的结点中的关键字

A screenshot of a cell phone

Description automatically generated

图4-1-2(8) 删除关键字个数为下限的结点中的关键字

（二）综合性实验：电子词典

[输入数据]

输入文件中每一行有四种数据，分别是英文单词、该词条单词词性、国际音标以及中文释义。

[输出数据]

输出文件中每一行有四种数据，分别是英文单词、该词条单词词性、国际音标以及中文释义。并按照单词a-z字母进行排序。

1. 构造电子词典验证

样例1 普通电子词典



图4-2-1(1) 电子词典样例1输入文件

样例1 测试结果

A close up of text on a white background

Description automatically generated

图4-2-1(2) 电子词典样例1终端初始化

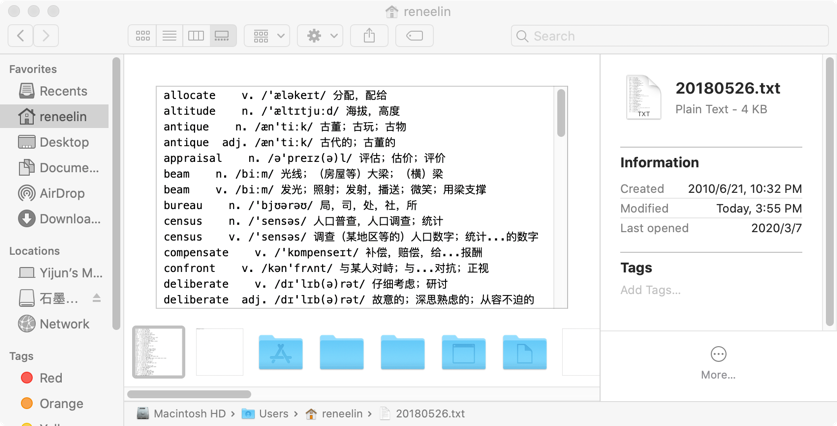


图4-2-1(3) 电子词典样例1输出文件

样例2 电子词典只有一个词条

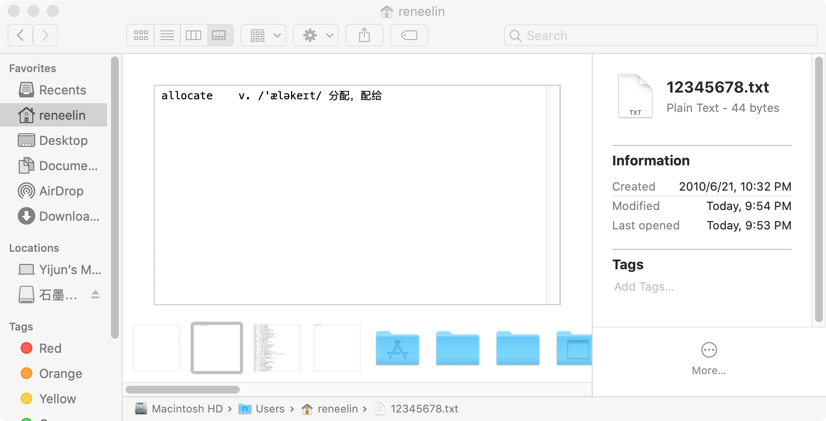


图4-2-2(1) 电子词典样例2输入文件

样例2 测试结果



图4-2-2(2) 电子词典样例2终端初始化

样例3 电子词典为空文件

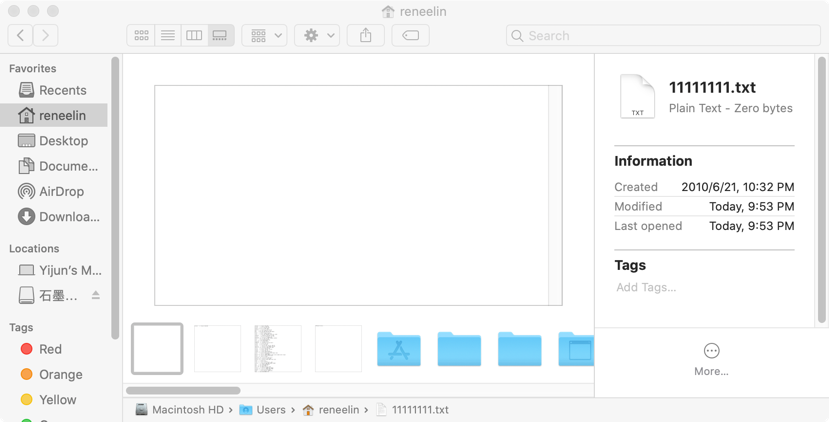


图4-2-3(1) 电子词典样例3输入文件

样例3 测试结果

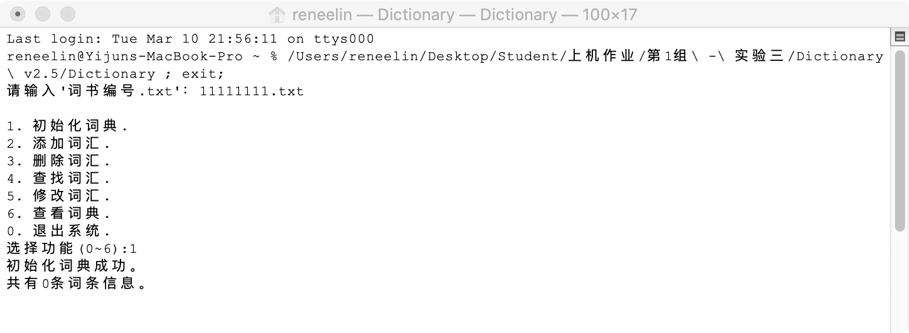


图4-2-3(2) 电子词典样例3输出文件

样例4 电子词典文件不存在

样例4 测试结果

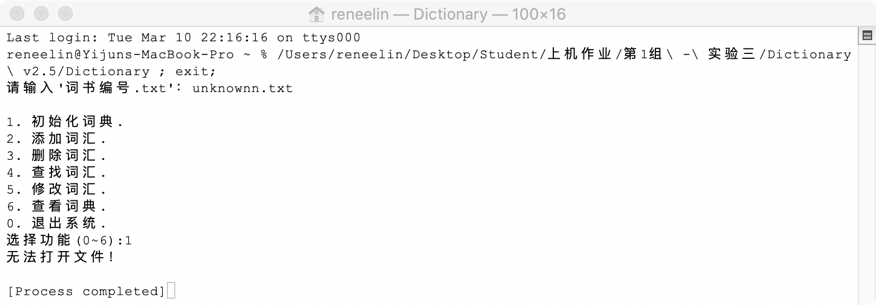


图4-2-4(1) 电子词典样例4终端初始化

样例5 普通电子词典（词条精简版）

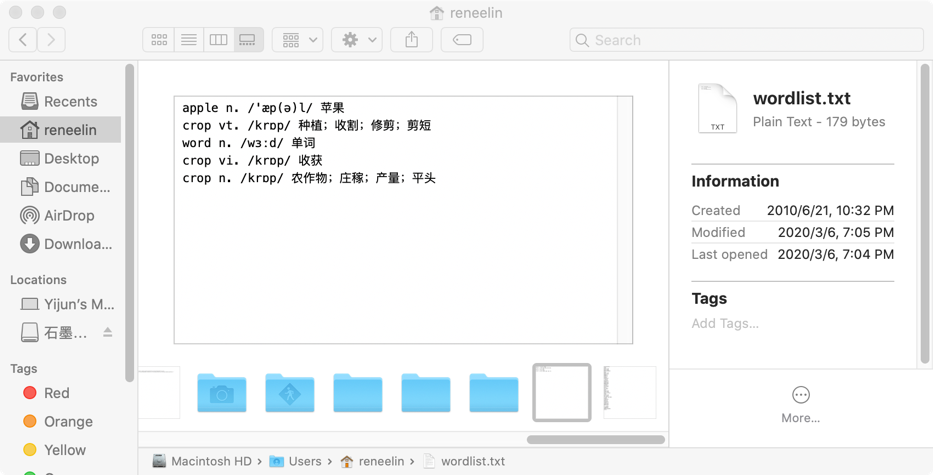


图4-2-5(1) 电子词典样例5输入文件

样例5 测试结果



图4-2-5(2) 电子词典样例5终端初始化

A screenshot of a cell phone

Description automatically generated

图4-2-5(3) 电子词典样例5输出文件

2. 电子词典功能验证

以“1. 构造电子词典验证”中的样例5作为以下功能测试案例。

(1) 添加词汇词条



图4-3-1 添加词汇词条终端

(2) 添加已存在的词汇词条

A screenshot of a social media post

Description automatically generated

图4-3-2 添加已存在的词汇词条终端

(3) 删除词汇词条

A screenshot of a social media post

Description automatically generated

图4-3-3 删除词汇词条终端

(4) 删除不存在的词汇词条



图4-3-4 删除不存在的词汇词条终端

(5) 查找词汇词条

A screenshot of a cell phone

Description automatically generated

图4-3-5 查找词汇词条终端

(6) 查找不存在的词汇词条

A screenshot of a social media post

Description automatically generated

图4-3-6 查找不存在的词汇词条终端

(7) 查找不存在的词汇词性词条

A screenshot of a cell phone

Description automatically generated

图4-3-7 查找不存在的词汇词性词条终端

(8) 修改词汇词条——词性

A screenshot of a cell phone

Description automatically generated

图4-3-8 修改词汇词条——词性终端

(9) 修改词汇词条——已存在的词性



图4-3-9 修改词汇词条——已存在的词性终端

(10) 修改词汇词条——国际音标

A screenshot of a social media post

Description automatically generated

图4-3-10 修改词汇词条——国际音标终端

(11) 修改词汇词条——中文释义

A screenshot of a cell phone

Description automatically generated

图4-3-11 修改词汇词条——中文释义终端

五．课程设计中遇到的问题和解决方法

（一）设计性实验：B树的设计与实现

B树在插入和删除元素时，如果结点关键字个数超出规定范围时，需要修改该结点及该结点兄弟结点或双亲结点，甚至会自下而上一路修改到根结点。所以设计中较为复杂的问题是插入时需要分裂结点或者删除时需要合并结点。

在设计B树结点的结构时，将结点的关键字和指针的储存空间都多留一个，分裂时以容纳双亲结点暂时超出关键字个数范围的关键字和指针。对于插入时分裂的结点，用FindKeyNumInParent()函数找到该结点在双亲结点中的位置，将分裂结点位于中间的关键字插到双亲结点的位置中，后半部分关键字移到新建的右兄弟结点。

删除时合并结点更为麻烦，有多种情况，一类是删除关键字后间接地向兄弟结点借关键字，另一类是删除关键字后向双亲结点借关键字，合并兄弟结点。再判断双亲结点是否要合并，从该结点向上直到没有结点需要合并。

在显示B树的功能上，借助队列层序遍历，使树能横向显示，计算出树的高度与宽度，保持横向结点间的间隔。

（二）综合性实验：电子词典

在设置数据结构的时候，介于同一个词汇含有多种词性的情况，为了减少词条长度和不准确性，将不同词性的同一词汇放在不同词条当中，并且在查询的时候分词性查询。由于二叉排序树对于关键字有互不相同的要求，因此在这种情况下不能将string类的单词本身作为关键字，而需要制造出一个独属每一个词条的关键字，就如同序列号一样。在上述数据结构中，已经具体描述了使用的方法。虽然我认为可能在这过程中，或许能有更简便的方法，但至少这样的方式解决了所出现的问题。

词典输出时，选择使用了C++的setfill与setw，但是出现了无法对齐的问题，目前尚未找到合适的解决办法，只得暂时搁置。想避免这种方式可以采用Qt等作为可视化界面化渠道进行改进。

六．实验总结

17120193 张赛

本次实验跨度较长，实验期间和组员沟通稍许有点不及时，故最后帮助组员完成程序内容中的测试样例和bug修复。在B树的实现后期需要对B树中存在时特殊节点情况进行测试，故使节点中关键字数量位于下限数量进行测试。后续再补充了单节点用例，排除在构造时初始化时的bug。

对于三月份开学以来的线上教学模式，一方面是感觉新奇另一方面是提醒自己要回到学习的状态中了。同时也很庆幸，当今网络的发达，可以提供充足的手段和工具为我们学生以及老师建立起沟通的平台。也让我们意识到，我们所学的知识是服务于大众的，在今后类似的特殊时期自然也会起到关键性作用。

18120189 林艺珺

在这次实验中我主要负责的电子词典的设计性实验部分。在放假之初，本着趁热打铁的想法，在放假第一周，依据书本上关于二叉排序树的内容，完成了大体框架和基础功能的编写，后续细节和查错工作，在新冠疫情和春节假期的双重夹击下一直拖延，直至近网上开学，才重新拾起。在重新查错的过程中，由于间隔时间长，会一下无法回忆自己当时的思路，但与此同时，有一个好处，清空思绪的时候会用一种全新的角度重新审视一些问题。于是，在重新阅读自己代码的同时，也思考出了一些新的解决问题的方式，在Debug的过程中，也更加有思绪、更加顺利。不过依然需要反思自己在这次实验过程中的拖延，未来职场中的项目可不会给自己这么长的时间。

现在是个很特殊的阶段，全国，或者说是全世界都在为抗击新型冠状病毒肺炎做出各自的努力。网课不是一种新的技术手段，但在这个初春，竟一破往日的习惯成为了主流的授课方式。虽说这个现状一定是暂时的，但在这过程中不禁让我们思考许多问题。网课的效率与线下授课存在不小差别，直播碍于技术手段无法及时或者高效进行课堂互动（比如我本周二回答问题的时候不可控的麦克风问题），但部分录播课的回看功能其实给予了更多的学习空间。网课的学习是新颖又回归“听”课本质的，希望自己能在网课的授课环境下也有一个较好的知识获取的效率与成果转化。

18120206 徐奕婷

本次实践让我了解并掌握了B树的插入和删除操作。

在尝试设计插入和删除函数时，大体思路如下：对于插入，一旦关键字个数溢出，那么结点就要分裂；对于删除，一旦关键字个数过少，那么结点就要合并。

总体是浅显易懂的。但我在考虑连续分裂/合并的情况时，在子树的过继问题上有些疏漏。以分裂操作为例，它的关键在新生成结点的双亲结点要指向分裂前结点的双亲结点，且过继的子树的双亲结点要重新定向到新生成的结点。简单来说就是，分裂/生成新结点时，要处理好其双亲结点与其子结点的指针。合并结点也是如此。

我一开始始终找不出bug，在学习了另一位同学的代码后，发现自己的程序在处理双亲结点的重定向时有错误，导致后续的递归操作崩盘。

这次艰辛的debug过程告诉我：写码前要捋清楚每一个逻辑点，任何一个点的不确定在环环相扣下就可能演变成复杂问题，对于后续测试找bug很不利。

最后总结一下，B树在算法中的地位至关重要，它在大量数据下的增删查效率很高，当然也存在缺陷——B树不适合范围查询。因为B树的查询是从根节点开始的，这导致在某个范围查询数据的性能较低。而从B树衍生出的B+树解决了该问题，所以后续我会更多的去深刻关于B+树的认识。

18120212 叶菁

3阶B树的设计大大提高了查找性能，插入删除功能的算法设计较复杂，由于有关键字个数的范围限制和B树的平衡性需求，需要分多种情况进行讨论，为这两个功能设计出许多辅助函数。树的显示兼顾了美观和实用，采用横向展开。菜单界面有二级菜单，有返回功能，打开一次程序能测试所有已有样例的所有功能。3阶B树的设计也使用于m阶B树的使用，增加阶数能减少查找次数，更能提高查找性能。

在疫情期间，我们小组使用微信讨论，和在校内交流没有区别。学习数据结构这门课则在老师的带领下使用线上直播的方式，有实时教学，ppt板书，师生间互动，大体上与线下教学没有区别，但线下授课方式更有课堂氛围，同学们坐在旁边，有激励作用，老师面对面讲课，学习效果也更好。

18120221 陈思文

这次因为疫情的原因，通过ZOOM在线学习，并通过QQ、微信来小组合作。在家学习，要克服惰性。远程协作的方式和原本的小组体验比较相似，只要网络状况良好，效率还是比较高的。实验中有一些困惑、难点，可以在群聊中和同学交流讨论。远程教学方面，因为有签到、上课互动、作业提交等形式，还是能够督促我学习。远程教学避免了换教室的烦恼，省出了很多自己的时间。但是容易分散注意力。希望以后可以录制直播相结合的方式提高学习效率。