**山东大学计算机科学与技术学院  
数据结构与算法课程设计报告**

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：202000120101 | 姓名： 尹国泰 | | 班级： 20.2 |
| 上机学时： 4 | | 日期：2022.5.20 | |
| 课程设计题目：单词拼写检查 | | | |
| 软件环境：VSCode,windows10操作系统 | | | |
| 报告内容：  1.需求描述  1.1 问题描述  现在有一些英语单词需要做拼写检查，在进行拼写检查之前首先建立一本词典。需要检查的单词，有的是词典中的单词，有的与词典中的单词相似，编写程序发现词典中与给定单词相同或相似的单词。  单词A与单词B相似的情况有三种：  (1)删除单词A的一个字母后得到单词B；  (2)用任意一个字母替换单词A的一个字母后得到单词B；  (3)在单词A的任意位置增加一个字母后得到单词B。  1.2 基本要求  （1）定义并实现字典ADT，基本操作根据应用需要设定。  （2）尽可能高效地实现单词的拼写检查。  1.3 输入说明  输入界面设计  可通过点击对应按钮从文件向词典中添加单词、向词典中添加单词、删除单词、清空词典。通过右侧的输入框输入要查找的单词。    输入样例  通过”dic1.txt”文件将单词导入到词典    1.4 输出说明  输出界面设计  导入词典后会显示词典中的单词，在搜索框中输入要查找的单词可以找到相同和相似词在右侧列出，并显示数量。可以通过底部按钮导出词典、导出匹配词、导出词典树结构。    输出样例  对于输入样例查询单词‘a’，可以找到右侧的相同和相似词，并显示有1个相同词，12个相似词。通过导出匹配词功能导出的文件如右侧所示。    2.分析与设计  2.1 问题分析   * 将题意概括一下，就是找出一个单词在字典中相同或相似的匹配词。倘若只有相同词的匹配，Tire字典树便可解决，但对于相似词，插入删除替换的位置是任意的，要找出来几乎要遍历整个字典，复杂度是无法接受的。 * 关键在于如何去匹配相似的单词。要解决此问题，可以使用一种BK树的结构，或者称为Burkhard-Keller树，这是一种基于树的数据结构，被设计于快速查找**近似字符串匹配**，比方说拼写纠错，或模糊查找，当搜索”aeek”时能返回”seek”和”peek”。 * 为此需要处理以下任务： * （1）输入界面模块：输入词典中要添加和删除的单词、输入要匹配的单词 * （2）输出界面模块：将查找到的所给单词的相同词和相似词进行输出显示、显示词典中的单词、导出词典和匹配词、导出词典树结构 * （3）词典单词增删模块：依据输入，向词典中添加和删除单词 * （4）查询匹配词模块：依据建立的BK树，查找所给单词的相同词和相似词   2.2 主程序设计  通过QT平台实现图形化界面，在主函数中创建Widget类对象w，并调用w的show()函数     * 在Widget类构造函数中将各按钮的信号与槽函数相连接。     2.3 设计思路  （1）输入界面模块  对入输入界面中的各个按钮和输入框，定义以下槽函数用于实现按钮功能，在构造函数中将以下槽函数与各自对应的按钮以及信号用QT库函数connect连接  void **pushButon\_ChooseDicFile\_clicked**();//从文件向词典中添加单词  void **lineEdit\_QueryWord\_changed**();//输入要查找的单词  void **on\_pushButton\_AddDic\_clicked**();//向词典中添加单词  void **on\_pushButton\_ClearDic\_clicked**();//清空词典  void **on\_pushButton\_EraseDic\_clicked**();//从词典中删除单词  （2）输出界面模块  界面中的输出按钮有导出词典、导出匹配词、导出词典树结构3个，为其设计以下槽函数  void **on\_pushButton\_DicOut\_clicked**();//导出词典  void **on\_pushButton\_WordOut\_clicked**();//导出匹配词  void **on\_pushButton\_BKTreeOut\_clicked**();//导出BK树结构  此外还有词典单词数、相同词数、相似词数、合计匹配词数4个LCD板输出以及词典、相同和相似词2个listView对象的输出，这些输出在输入按钮的槽函数中进行更新  （3）词典单词增删模块  建立BKTree类用于处理词典，在BKTree中建立以下函数用于单词的增删  void **clear**(){if(root!=NULL) erase(*root*); root=NULL; };  清空BKTree中的单词，即删除整棵BKTree，实现一个内部函数**erase**(BKTreeNode\* t)递归删除t为根的子树，调用erase(root)来删除整棵树  void **insert**(const string &word);  向BKTree中添加一个单词word，调用一个内部函数**insert**(root, word)递归找到插入位置并插入，关于这个类内函数的设计详见2.5算法设计及分析部分  void **erase**(const string &word);  从BKTree中删除一个单词word，首先找到word所在的结点cur，实现一个内部函数**erasechd**(BKTreeNode\* t,vector<string> &vec)递归删除以t为根的子树并将删除的单词存到vec中，  调用erasechd（cur，vec）之后，将vec中记录的子树单词依次用insert再次插入到树中。  （4）查询匹配词模块  在BKTree中设计下面这一个函数用于查找匹配词  vector<string> **query**(const string &word,int dist);  这个函数将BKTree中与word编辑距离为dist的单词全部找出并以一个vector数组的形式返回结果。  字符串A到B的编辑距离(LevenShtein Distance)是指，只用插入、删除和替换三种操作，最少需要多少步可以把A变成B，对于本题而言只需在外部调用query(word,1)即可。通过调用一个类内函数void **query**(root, word,1, vec)用于查找，关于编辑距离的说明以及这个内部函数的设计详见2.5算法设计及分析部分  2.4 数据及数据类(型)定义  BK树的结点采用如下结构体存储，存储了结点上的单词、map的第二维是结点子结点指针，map第一维是对应与每个子结点的编辑距离    在BKTree类中只需存储根节点指针    2.5.算法设计及分析  BKTree的核心部分在于以下三个类内的函数，其他函数均为基本的树结构遍历函数以及外部调用函数，此部分主要分析以下这三个函数    2.5.1 int **distance**(string s1,string s2); //返回s1与s2的编辑距离  （1）编辑距离  字符串A到B的编辑距离(LevenShtein Distance)是指，只用插入、删除和替换三种操作，最少需要多少步可以把A变成B。  本题目中  A与B相同：A与B的编辑距离为0  A与B相似：A与B的编辑距离为1  本题转化为，给出一个字典，对于每个单词，**找到字典中与之编辑距离为0或1的匹配词**  （2）编辑距离的性质  d(x,y) = 0 当且仅当 x=y  （编辑距离为0 <==> 字符串相等）  d(x,y) = d(y,x)  （从x变到y的最少步数就是从y变到x的最少步数）  d(x,y) + d(y,z) >= d(x,z)  （从x变到z所需的步数不会超过x先变成y再变成z的步数）  （3）求解字符串A[1…n]与B[1…m]间的编辑距离   * 采用动态规划求解 * 状态：D[ i ][ j ]表示A[1…i]与B[1…j]的编辑距离 * 初始条件：D[0][0]=0,D[0][j]=j, D[i][0]=i * 转移方程：D[ i ][ j ] = min ( D[ i-1 ][ j ] + 1,   D[ i ][ j-1 ] + 1,  D[i-1 ][ j-1 ] + (A[i]==B[j]?0:1) );  目标：D[n][m]  时间复杂度：O(nm)  2.5.2 void **insert**(BKTreeNode\* t,const string &word);  //向t为根的树中加入一个单词word,递归实现  （1）算法流程  计算t->word与word的编辑距离D  如果t不存在标号为D的子节点，为t新建一个标号为D子节点，并将子节点的值设置为word；  如果t存在标号为D的子节点c，递归运行insert(c,word);  **要注意插入过程中有相同词，即D=0，不重复插入。**  （2）建立的BK树结构分析  对于树中每个结点u，u的标号为D的分支的子树中所有单词，与u结点中的单词编辑距离均为D  2.5.3 void **query**(BKTreeNode\* t,const string &word,int dist,vector<string> &vec); //从t为根的树中查询与word的编辑距离小于等于dist的所有单词,存到vec中,递归实现  （1）算法流程  计算t->word与word的编辑距离D  如果D<=dist, t->word统计到vec中  找到所有t的标号在[ max(D-dist,1) , D+dist ]范围内的子节点ci，对每个ci递归运行query(ci,word,dist,vec);  （2）正确性  根据编辑距离的性质d(x,y) + d(y,z) >= d(x,z)  设w为进行匹配的单词，t当前根结点单词，c为子树节点单词  如果c能够与w匹配，有d(w,c)<=dist，可以得到  d(t,c)<=d(t,w)+d(w,c)<=D+dist  d(t,c)>=d(t,w)-d(w,c)>=D-dist  t与c是不同的单词，有d(t,c)>=1  因此得到d(t,c)，即可能匹配的子节点标号的范围是[ max(D-dist,1) , D+dist ]  3. 测试  （1）初始界面    （2）从文件向词典中添加单词  点击对应按钮，选择dic2.txt文件，点击打开    添加完毕后跳出添加完毕提示框，在词典中列出了添加的单词，词典单词数显示为914    （3）输入要查找的单词  在右侧输入框输入单词ab，列出了相同和相似词，以及数量    （4）向词典中添加一个单词    abc成功添加到词典，词典单词数也变为915，并且右边的匹配词自动进行了更新    （5）从词典中删除单词    ab成功删除，词典单词数变为914，并且右边的匹配词自动进行了更新，ab不再存在，右边相同词数量也变为0.    （6）清空词典    （7）导出词典  重新将dic1.txt中的单词导入到词典中    导出词典到outdic.txt      (8)导出匹配词        （9）导出词典树结构      导出成功，树中显示了每个结点上存储的单词以及每个分支的编辑距离标号    4. 分析与探讨  BKTree的优势在何处？  单词的长度一都是比较短的，判断两个单词是否为相同或相似词（也就是求编辑距离）的时间复杂度是很低的，而词典中单词的数量是庞大的，如果要降低时间复杂度应当从减少要判断的单词数量这方面来下手，BKTree就是在单词编辑距离匹配前提下减少了要比较的单词数量。  以下为对BKTree时间复杂度的简单分析    由于m远小于n，时间复杂度主要关注n的大小，多数情况下BKTree的结点分布是比较平均的，要比较的单词可以从n的级别降低到logn的级别，效率有极大的提升。  5. 附录：实现源代码  5.1 BKTree.h   |  | | --- | | #include<bits/stdc++.h>  using namespace std;  struct **BKTreeNode**{  string word;  map<int,BKTreeNode\*> chd;  **BKTreeNode**(const string &\_word)  :word(\_word){}  };  class **BKTree**{  private:  BKTreeNode\* root;  int **distance**(string s1,string s2);  //返回s1与s2的编辑距离  void **insert**(BKTreeNode\* t,const string &word);  //向t为根的树中加入一个单词word,递归实现  void **query**(BKTreeNode\* t,const string &word,int dist,vector<string> &vec);  //从t为根的树中查询与word的编辑距离小于等于dist的所有单词,存到vec中,递归实现  void **erase**(BKTreeNode\* t);  void **erasechd**(BKTreeNode\* t,vector<string> &vec);  void **output**(BKTreeNode\* t,ofstream &out);  public:  **BKTree**(){root=NULL; };  ~**BKTree**(){if(root!=NULL) erase(*root*); };    void **clear**(){if(root!=NULL) erase(*root*); root=NULL; };  void **insert**(const string &word); //外部调用接口  void **erase**(const string &word);  vector<string> **query**(const string &word,int dist); //外部调用接口  void **bKTreeOut**(string outpath);  }; |   5.2 BKTree.cpp   |  | | --- | | #include "BKTree.h"  void BKTree::**erase**(BKTreeNode\* t){  for(auto i:t->chd) erase(*i.second*);  delete t;  }  int BKTree::**distance**(string s1,string s2){  if(s1.size()==0) return s2.size();  if(s2.size()==0) return s1.size();  int D[s1.size()+1][s2.size()+1];  for(int i=1;i<=(int)s1.size();i++) D[i][0]=i;  for(int j=1;j<=(int)s2.size();j++) D[0][j]=j;  D[0][0]=0;  for(int i=1;i<=(int)s1.size();i++){  for(int j=1;j<=(int)s2.size();j++){  D[i][j]=min(D[i-1][j]+1,D[i][j-1]+1);  D[i][j]=min(D[i][j],D[i-1][j-1]+(s1[i-1]==s2[j-1]?0:1));  }  }  return D[s1.size()][s2.size()];  }  void BKTree::**insert**(BKTreeNode\* t,const string &word){  int D=distance(t->word,word);  if(D==0) return;  if(t->chd.count(D)) insert(*t->chd[D]*,word);  else{  BKTreeNode\* newt=new BKTreeNode(word);  t->chd[D]=newt;  }  }  void BKTree::**insert**(const string &word){ //外部调用接口  if(root==NULL){  root=new BKTreeNode(word);  }else insert(*root*,word);  }  void BKTree::**query**(BKTreeNode\* t,const string &word,int dist,vector<string> &vec){  int D=distance(t->word,word);  if(D<=dist) vec.push\_back(t->word);  for(int i=max(D-dist,1);i<=D+dist;i++){  if(t->chd.count(i))  query(*t->chd[i]*,word,dist,*vec*);  }  }  vector<string> BKTree::**query**(const string &word,int dist){ //外部调用接口  vector<string> vec;  if(root!=NULL) query(*root*,word,dist,*vec*);  return vec;  }  void BKTree::**erasechd**(BKTreeNode\* t,vector<string> &vec){  for(auto i:t->chd){  vec.push\_back(i.second->word);  erasechd(*i.second*,*vec*);  }  delete t;  }  void BKTree::**erase**(const string &word){  if(root==NULL) return;  BKTreeNode\* t=root,\*pre=NULL;  int D=distance(t->word,word);  while(D){  if(t->chd.count(D)) pre=t,t=t->chd[D];  else return;  D=distance(t->word,word);  }  if(pre!=NULL){//删除的不是根  pre->chd.erase(distance(t->word,pre->word));  }else{//删除的是根  root=NULL;  }  vector<string> vec;  erasechd(*t*,*vec*);  for(auto i:vec) insert(i);  }  void BKTree::**output**(BKTreeNode\* t,ofstream &out){  for(auto i:t->chd){  out<<t->word<<" -> "<<i.second->word<<" [label=\""<<i.first<<"\"];\n";  output(*i.second*,*out*);  }  }  void BKTree::**bKTreeOut**(string outpath){  ofstream out(outpath);  out<<"digraph G {"<<'\n';  if(root!=NULL) output(*root*,*out*);  out<<"}"<<endl;  out.close();  } |   5.3 main.cpp   |  | | --- | | #include "widget.h"  #include <QApplication>  int main(int argc, char \*argv[])  {  QApplication a(*argc*, *argv*);  Widget w;  w.show();  return a.exec();  } |   5.4 widget.h   |  | | --- | | #ifndef WIDGET\_H  #define WIDGET\_H  #include "BKTree.h"  #include <QWidget>  #include <QDir>  #include <QFileDialog>  #include <QStringList>  #include <QStringListModel>  #include <QLineEdit>  #include <QListView>  #include <QLCDNumber>  #include <QInputDialog>  #include <QMessageBox>  QT\_BEGIN\_NAMESPACE  namespace **Ui** { class **Widget**; }  QT\_END\_NAMESPACE  class **Widget** : public QWidget  {  Q\_OBJECT  public:  **Widget**(QWidget \*parent = nullptr);  ~***Widget***();  void **pushButon\_ChooseDicFile\_clicked**();//从文件向词典中添加单词  void **lineEdit\_QueryWord\_changed**();//输入要查找的单词  void **on\_pushButton\_AddDic\_clicked**();//向词典中添加单词  void **on\_pushButton\_ClearDic\_clicked**();//清空词典  void **on\_pushButton\_EraseDic\_clicked**();//从词典中删除单词  private slots:  void **on\_pushButton\_DicOut\_clicked**();//导出词典  void **on\_pushButton\_WordOut\_clicked**();//导出匹配词  void **on\_pushButton\_BKTreeOut\_clicked**();//导出BK树结构  private:  Ui::Widget \*ui;  BKTree tree;  set<string> dictionary;  set<string> matchword;  };  #endif // WIDGET\_H |   5.5 widget.cpp   |  | | --- | | #include "widget.h"  #include "ui\_widget.h"  Widget::**Widget**(QWidget \*parent)  : QWidget(*parent*)  , ui(new Ui::Widget)  {  ui->setupUi(this);  connect(ui->pushButton\_ChooseDicFile,&QPushButton::clicked,  this,&::Widget::pushButon\_ChooseDicFile\_clicked);  connect(ui->pushButton\_AddDic,&QPushButton::clicked,  this,&::Widget::on\_pushButton\_AddDic\_clicked);  connect(ui->pushButton\_ClearDic,&QPushButton::clicked,  this,&::Widget::on\_pushButton\_ClearDic\_clicked);  connect(ui->pushButton\_EraseDic,&QPushButton::clicked,  this,&::Widget::on\_pushButton\_EraseDic\_clicked);  connect(ui->lineEdit\_QueryWord,&QLineEdit::textChanged,  this,&::Widget::lineEdit\_QueryWord\_changed);  }  Widget::~***Widget***()  {  delete ui;  }  void Widget::**lineEdit\_QueryWord\_changed**(){  QString word=ui->lineEdit\_QueryWord->text();  if(word.size()){  vector<string> vec=tree.query(word.toStdString(),1);  vector<string> eql=tree.query(word.toStdString(),0);  QStringList strlist;  QStringListModel \*model = new QStringListModel(this);  matchword.clear();  for(auto i:vec) matchword.insert(i);  for(auto i:matchword) strlist<<QString::fromStdString(i);  model->setStringList(strlist);  ui->listView->*setModel*(*model*);  ui->lcdNumber->display(QString::number(eql.size(),10));  ui->lcdNumber\_2->display(QString::number(vec.size()-eql.size(),10));  ui->lcdNumber\_3->display(QString::number(vec.size(),10));  }else{  QStringListModel \*model = new QStringListModel(this);  ui->listView->*setModel*(*model*);  ui->lcdNumber->display(0);  ui->lcdNumber\_2->display(0);  ui->lcdNumber\_3->display(0);  }  }  void Widget::**pushButon\_ChooseDicFile\_clicked**()  {  QString curPath=QDir::currentPath()+"//text";  QString dlgTitle="选择一个字典文件";  QString filter="文本文件(\*.txt);;所有文件(\*.\*)";  QString aFileName=QFileDialog::getOpenFileName(this,dlgTitle,curPath,filter);  if (!aFileName.isEmpty()){  ifstream in(aFileName.toStdString());  string dic;  while(in>>dic){  tree.insert(dic);  if(!dictionary.count(dic)) dictionary.insert(dic);  }  in.close();  QStringList strlist;  QStringListModel \*model = new QStringListModel(this);  for(auto i:dictionary) strlist<<QString::fromStdString(i);  model->setStringList(strlist);  ui->listView\_2->*setModel*(*model*);  ui->lcdNumber\_4->display(QString::number(dictionary.size(),10));  Widget::lineEdit\_QueryWord\_changed();  QMessageBox::information(this,"从文件向词典中添加单词","单词添加完毕！",QMessageBox::Ok,QMessageBox::NoButton);  }  }  void Widget::**on\_pushButton\_AddDic\_clicked**()  {  QString dlgTitle="向词典中添加单词";  QString txtLabel="请输入单词";  QString defaultInput="";  QLineEdit::EchoMode echoMode=QLineEdit::Normal;  bool ok=false;  QInputDialog Dialog;  QString text = Dialog.getText(this, dlgTitle,txtLabel, echoMode,defaultInput,*&ok*);  if (ok && !text.isEmpty()){  string dic=text.toStdString();  tree.insert(dic);  if(!dictionary.count(dic)) dictionary.insert(dic);  QStringList strlist;  QStringListModel \*model = new QStringListModel(this);  for(auto i:dictionary) strlist<<QString::fromStdString(i);  model->setStringList(strlist);  ui->listView\_2->*setModel*(*model*);  ui->lcdNumber\_4->display(QString::number(dictionary.size(),10));  Widget::lineEdit\_QueryWord\_changed();  QMessageBox::information(this,"向词典中添加单词","单词添加完毕！",QMessageBox::Ok,QMessageBox::NoButton);  }  }  void Widget::**on\_pushButton\_ClearDic\_clicked**()  {  tree.clear();  dictionary.clear();  QStringListModel \*model = new QStringListModel(this);  ui->listView\_2->*setModel*(*model*);  ui->lcdNumber\_4->display(QString::number(dictionary.size(),10));  Widget::lineEdit\_QueryWord\_changed();  QMessageBox::information(this,"清空词典","词典已经清空！",QMessageBox::Ok,QMessageBox::NoButton);  }  void Widget::**on\_pushButton\_EraseDic\_clicked**()  {  QString dlgTitle="从词典中删除单词";  QString txtLabel="请输入单词";  QString defaultInput="";  QLineEdit::EchoMode echoMode=QLineEdit::Normal;  bool ok=false;  QInputDialog Dialog;  QString text = Dialog.getText(this, dlgTitle,txtLabel, echoMode,defaultInput,*&ok*);  if (ok && !text.isEmpty()){  string dic=text.toStdString();  tree.erase(dic);  if(dictionary.count(dic)) dictionary.erase(dic);  QStringList strlist;  QStringListModel \*model = new QStringListModel(this);  for(auto i:dictionary) strlist<<QString::fromStdString(i);  model->setStringList(strlist);  ui->listView\_2->*setModel*(*model*);  ui->lcdNumber\_4->display(QString::number(dictionary.size(),10));  Widget::lineEdit\_QueryWord\_changed();  QMessageBox::information(this,"从词典中删除单词","单词删除完毕！",QMessageBox::Ok,QMessageBox::NoButton);  }  }  void Widget::**on\_pushButton\_DicOut\_clicked**()  {  QString curPath=QDir::currentPath()+"//text";  QString dlgTitle="导出词典";  QString filter="文本文件(\*.txt);;所有文件(\*.\*)";  QString aFileName=QFileDialog::getSaveFileName(this,dlgTitle,curPath,filter);  if (!aFileName.isEmpty()){  ofstream out(aFileName.toStdString());  string dic;  for(auto i:dictionary){  out<<i<<'\n';  }  out.close();  QMessageBox::information(this,"导出词典","词典导出完毕！",QMessageBox::Ok,QMessageBox::NoButton);  }  }  void Widget::**on\_pushButton\_WordOut\_clicked**()  {  QString curPath=QDir::currentPath()+"//text";  QString dlgTitle="导出匹配词";  QString filter="文本文件(\*.txt);;所有文件(\*.\*)";  QString aFileName=QFileDialog::getSaveFileName(this,dlgTitle,curPath,filter);  if (!aFileName.isEmpty()){  ofstream out(aFileName.toStdString());  string dic;  for(auto i:matchword){  out<<i<<'\n';  }  out.close();  QMessageBox::information(this,"导出匹配词","匹配词导出完毕！",QMessageBox::Ok,QMessageBox::NoButton);  }  }  void Widget::**on\_pushButton\_BKTreeOut\_clicked**()  {  QString curPath=QDir::currentPath()+"//text";  QString dlgTitle="导出词典树结构";  QString filter="png图片(\*.png);;jpg图片(\*.jpg);;pdf文件(\*.pdf);;所有文件(\*.\*)";  QString aFileName=QFileDialog::getSaveFileName(this,dlgTitle,curPath,filter);  if (!aFileName.isEmpty()){  string picpath=aFileName.toStdString();  string dotpath=picpath.substr(0,picpath.size()-4)+".dot";  tree.bKTreeOut(dotpath);  string cmd="dot -T"+picpath.substr(picpath.size()-3,3)+" "+dotpath+" -o "+picpath;  system(cmd.c\_str());  QMessageBox::information(this,"导出词典树结构","词典树导出完毕！",QMessageBox::Ok,QMessageBox::NoButton);  }  } | | | | |