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 课程实验报告

|  |
| --- |
| 实验目的：  设计N=26、建立在英语上的TrieN-ary Trie的ADT，并应用该结构实现文本文档的索引化和单词查询（单词整体查询和利用单词前缀查询）。 |
| 软件开发工具：  Visual Studio 2022 |
| 1. **实验内容**   **1.1问题描述**  哈夫曼算法输出的结果就是一个二元Trie，在二元Trie中，每个左子树分支用0表示，右子树分支用1表示。如下图就是就是一个二元Trie的示例图。    可以将二元Trie扩展到N-ary Trie。在N-ary Trie中，每个结点都有0~N之间任何个儿子结点，其中每个分支都用一个相应的符号表示（在N-ary Trie中有N个不同的符号）。例如一个电话本可用一个N-ary Trie表示，其中N=10，分别表示0~9的十个数字，具体情况如下图所示：    **1.2基本要求**  （1）设计并实现N-ary Trie的ADT（N=26，建立在英语上的Trie），该ADT包括Trie的组织存储以及其上的基本操作：包括初始化，查找，插入，删除等。  （2）应用Trie结构实现文本文档的索引化。首先扫描文本文档（存放在txt文件中），然后在此基础上用Trie登记单词出现的位置（行号），最后在Trie上实现查询。  （3）用户输入的查询可以是针对一个单词的，也可是针对一个字符序列的（以某几个字母开头的单词）。  **1.3输入说明**  利用VS2022结合easyx库实现了图形化界面，并将鼠标点击、键盘输入、文件读入三种方式相结合。  **1.3.1输入界面设计**   1. **主界面**：鼠标点击选择进入不同界面。      1. **单词整体查询界面**：鼠标点击选择输入框，再由键盘输入要查询的完整单词      1. **单词前缀查询界面**：鼠标点击选择输入框，再由键盘输入要查询的单词前缀      1. **返回**：鼠标点击各子界面的“返回”按键，可返回主界面（如上面两张图的右下角）   **1.3.2输入样例**  （见第三部分“测试结果”。）  **1.4输出说明**  在输入的图形化界面中输出结果，同时在程序运行的终端窗口也输出结果。  （具体界面设计参考1.3，测试见第三部分“测试结果”。）   1. **数据结构与算法描述（整体思路描述，所需要的数据结构与算法）**   **2.1问题分析**  根据题目给的已知条件和基本要求将问题分成以下几个子任务：  （1）设计并实现具有基础功能的N-ary Trie数据结构  （2）利用Trie结构实现外部文档的单词查询，输出行号  （3）实现单词整体查询以及前缀查询  **2.2主程序设计**  #include“Ntree.h”  #include“oper.h”  Int main(){  indexTextFile(...);//读取文本文档并进行索引化  Init();initMenu();//图形化（主）界面初始化  While(true){//不断读取鼠标消息  If(在主界面){  ...//根据点击范围跳转到对应界面  }else if(在两个查询界面){  ...//根据点击范围返回主界面或实现文本框输入  }  }  }  **2.3设计思路**  （1）思路：根据“设计并实现N-ary Trie数据结构”的要求定义相关类和节点结构体，类中声明几种ADT方法，结构体存放每个的子节点和行号；根据“实现两种方式查询、查看单词表、删除单词表中单词的功能”的要求，利用上述定义的N-ary Trie数据结构实现查询，并使用递归遍历的方法输出单词表和删除单词。  （2）程序结构设计：①Ntree.h:存放实现N-ary Trie的Trie类和TrieNode节点结构体，以及存放单词表的wordNode结构体;②oper.h:存放索引化文本文档、递归遍历输出查询结果及单词表等操作；③picture.cpp:实现图形化界面的主函数，调用以上头文件实现功能。    **2.4数据及数据类型定义**   1. **Trie结构体**   该结构体维护trie中的每一个节点，每个节点有一个children[N]存放26个子节点表示26个字母，一个int类型的向量数组存放以当前节点为结尾的单词的行号，一个布尔类型的变量用于判断当前节点是否为单词结尾，最后还有一个构造函数用于初始化。    （2）**TrieNode类**  该类用于维护26元单词查找树，包含一个TrieNode类型的root表示树的根节点、一个构造函数用于初始化以及一些ADT方法（将在2.5中详细展开分析）。    （3）**wordNode结构体**  该结构体用于维护最终的单词表，与TrieNode类似，但是只有一个string类型变量表示单词和一个int类型的向量存放当前单词的行号。    **2.5算法设计及分析**   1. **Trie的ADT方法**   **①insert函数：插入单词**  对于单词中的每个字符c，执行以下操作：计算c在字母表中的索引，检查curr子节点中是否存在与c对应的子节点。不存在则创建一个新节点，并将其赋值给curr->children[index]；将curr移动到当前字符对应的子节点，处理下一个字符。在最后一个字符所在的节点上做标志，表示从根节点到当前节点的路径构成了一个完整的单词；更新当前单词在文本中的位置position。    **②search1函数：确认整个单词是否存在**  从根节点开始遍历单词中的每个字符，查找对应的子节点，如果子节点不存在，则返回false，表示该单词不在Trie中。如果遍历完所有字符后，当前节点是一个单词的结束节点，则返回true，否则返回false。    **③search2函数：以prex为前缀的单词是否存在**  用一个for循环，对于每个单词使用compare函数比较单词表中单词的前缀和给定的前缀 p是否相等。如果相等，则返回 true，表示找到了以给定前缀开头的单词；如果循环结束后没有找到以给定前缀开头的单词，则返回 false，表示不存在以该前缀开头的单词。    **④searchWord1函数：查询整个单词返回行号**  遍历给定单词的每个字符，查找对应的子节点，直到找到最后一个字符所对应的节点，然后返回该节点的位置信息。    **⑤searchWord2函数：查询单词前缀返回行号**  创建一个wordNode类型的向量用于存储搜索到的单词信息。用一个for循环，遍历Trie 中存储的单词，使用compare函数比较单词表中的单词的前缀和给定的前缀是否相等。如果相等，则将该单词的信息（单词和位置）存储到向量p中，并更新计数器 n。根据实际搜索到的单词数量，调整向量 p 的大小为 n。最后返回存储以给定前缀开头的单词信息的向量 p。     1. **oper.h**   **①indexTextFile函数：读取文本文档并构建Trie树**  首先打开指定的文件，再逐行读取文件内容。在每一行中使用一个字符串变量word来收集单词，并将每个字符转换为小写字母以统一大小写。在遍历每个字符时，如果遇到的是字母字符，则将其添加到 word 中。如果遇到的是非字母字符，则将之前收集的单词插入到Trie中，并清空word。在每一行处理完毕后，如果word非空，则说明还有一个单词未处理完毕，将其插入到Trie中。最后关闭文件。    **②displayTrieWords函数：递归遍历树输出所有单词**  首先检查当前节点是否为空，如果为空则直接返回。然后，如果当前节点表示一个单词的结尾，则输出该单词以及其出现的行号。接着，函数递归遍历当前节点的所有子节点，对每个子节点执行相同的操作，同时更新前缀为当前前缀加上对应字符。     1. **picture.cpp（该部分涉及图形化界面代码，此处仅展示最重要的两个函数）**   **①drawDisplayWordList函数：绘制单词表**  首先初始化绘图环境。然后绘制表格，startX、startY定义表格左上角的起始位置，cellWidth、cellHeight定义表格中单元格的宽度和高度；使用两个for循环绘制表格的框架，包括横线和竖线。最后填充单词，使用循环遍历单词表v，并使用outtextxy()函数将每个单词绘制在表格中。当绘制完一列（25个单词）时，startX 增加 cellWidth，startY 重置为初始值。    **②handInput函数：响应鼠标点击事件**  在主界面（MAIN\_MENU）时，根据鼠标点击的位置，判断用户点击了哪个选项，并相应地更新状态和显示相应的界面。  在单词查询界面（WORD\_QUERY1 和 WORD\_QUERY2）时，同样根据鼠标点击的位置执行不同的操作。如果点击了返回主界面按钮，就清除当前绘图区域，重新初始化并显示主界面；如果点击了单词查询按钮，会读取用户输入的单词，并根据当前状态进行相应的查询操作。如果是整体查询（WORD\_QUERY1），则搜索并显示单词在文本中的位置；如果是前缀查询（WORD\_QUERY2），则搜索并显示具有相同前缀的单词及其位置。在查询过程中，结果会在图形界面上显示，并输出到终端控制台。       1. **测试结果** 2. **单词整体查询界面**：在方框中输出要查询的单词，在方框下显示结果；同时在终端输出中一并显示结果。      1. **单词前缀查询界面**：在方框中输入单词前缀，在下方输出所有可能涉及的单词以及单词所在的行号；同时在终端中一并显示结果。        1. **查看单词表界面**：程序运行后在“查看单词表”界面自动显示文本文档中出现过的所有单词，并在终端输出所有单词以及单词所在的行号（图形化界面放不下行号，就在终端输出了。）          1. **文本文档验证正确性**   下图是用作测试的文本文件的内容，其中每段英文为“一行”并且空行也算一行。此处标志出单词“is”的位置，对比（1）可知结果正确。     1. **分析与探讨**   （1）本题还有一些上述报告中没有展示功能，比如说当输入一个不存在的单词或者以输入为前缀的单词不存在时，都会输出错误提示，如下图：     1. 本题还有一个操作是把文本文档中所有的字母都转化成了小写字母，这样在搜索以及维护单词表的过程中都更加高效便利，而且不会出错。这里运用了函数tolower。      1. **时间复杂度**   **①insert函数**  插入单词时要遍历单词中的每一个字母，而每一个字母都要从子节点a开始遍历直到找到当前字母，因此时间复杂度为O(m)，其中m是单词中每个字母在26个字母中序号的累加值。  **②search1、searchWord1函数**  这两个函数基本结构相同，就是返回的内容不同，此处只分析search1。遍历单词中每个字母，判断当前字母在上一个字母的子节点中是否存在，因此时间复杂度为O(n)，其中n为单词长度。  **③search1、searchWord1函数**  这两个函数基本结构相同，就是返回的内容不同，此处只分析search2。这个搜索跟上面的搜索不同，这里直接在单词表中进行逐一查找，时间复杂度为O(n)，其中n为当前单词在单词表中的序列号（单词表按a到z排序，所以其实时间复杂度跟单词每个字母的序列号有关）。   1. **附录：实现源代码** 2. Ntree.h   #pragma once  #include<bits/stdc++.h>  using namespace std;  const int N = 26;  int n = 0;  //单词表  struct wordNode {  vector<int> positions;  string word;  };  vector<wordNode> v(1000000);  //Trie节点的定义  class TrieNode {  public:  TrieNode\* children[N];  vector<int> positions; //记录单词出现的行号  bool isEndOfWord;  TrieNode() {  isEndOfWord = false;  for (int i = 0; i < N; ++i)  children[i] = nullptr;  }  };  //Trie类的定义  class Trie {  public:  TrieNode\* root;  Trie() { root = new TrieNode(); }  void insert(const string& word, int position);//插入单词  bool search1(const string& word);//整个单词是否存在  bool search2(const string& prex);//以prex为前缀的单词是否存在  vector<int> searchWord1(const string& word);//查询整个单词返回行号  vector<wordNode> searchWord2(const string& prex);//查询单词前缀返回行号  };  //插入单词  void Trie::insert(const string& word, int position) {  TrieNode\* curr = root;  for (char c : word) {  int index = c - 'a';  if (!curr->children[index])  curr->children[index] = new TrieNode();  curr = curr->children[index];  }  curr->isEndOfWord = true;  curr->positions.push\_back(position);  }  //整个单词是否存在  bool Trie::search1(const string& word) {  TrieNode\* curr = root;  for (char c : word) {  int index = c - 'a';  if (!curr->children[index])  return false;  curr = curr->children[index];  }  return curr != nullptr && curr->isEndOfWord;  }  //以prex为前缀的单词是否存在  bool Trie::search2(const string& prex) {  for (int i = 0; i < v.size(); i++) {  if (v[i].word.compare(0, prex.length(), prex) == 0) {  return true;  }  }  return false;  }  //查询单词返回行号  vector<int> Trie::searchWord1(const string& word) {  TrieNode\* curr = root;  for (char c : word) {  int index = c - 'a';  if (!curr->children[index])  return {};  curr = curr->children[index];  }  return curr->positions;  }  //查询单词前缀返回行号  vector<wordNode> Trie::searchWord2(const string& prex) {  vector<wordNode> p(100000);  for (int i = 0; i < v.size(); i++) {  if (v[i].word.compare(0, prex.length(), prex) == 0) {  p[n].word = v[i].word;  p[n].positions = v[i].positions;  n++;  }  }  p.resize(n);  return p;  }   1. Oper.h   #pragma once  #pragma once  #include"Ntree.h"  Trie Ti;  int num = 0;//单词表中单词个数  //将文本文档索引化并构建Trie  void indexTextFile(const string& filename, Trie& trie) {  ifstream file(filename);  if (!file.is\_open()) {  cout << "文件无法打开！" << filename << endl;  return;  }  string line;  int lineNumber = 1;  while (getline(file, line)) {  string word;  for (char c : line) {  if (isalpha(c)) {  word += tolower(c); //转换为小写字母  }  else {//遇到非字母字符，将之前收集的单词插入到Trie中  if (!word.empty()) {  trie.insert(word, lineNumber);  word.clear();  }  }  }  if (!word.empty()) {  trie.insert(word, lineNumber);  }  lineNumber++;  }  file.close();  }  //递归遍历Trie并输出所有单词及其行数  void displayTrieWords(TrieNode\* node, const string& prefix = "") {  if (node == nullptr) return;  //如果当前节点是单词的结尾，输出该单词及其行数  if (node->isEndOfWord) {  v[num].word = prefix;  cout << prefix << " ( 所在行号:";  v[num].positions = node->positions;  for (auto j : node->positions) {  cout << j << " ";  }  cout << ")" << endl;  num++;  }  //递归遍历所有子节点  for (int i = 0; i < N; ++i) {  if (node->children[i] != nullptr) {  char c = 'a' + i;  string newPrefix = prefix + c;  displayTrieWords(node->children[i], newPrefix);  }  }  }  //显示Trie中的所有单词及其行数  void displayTrieWords(Trie& trie) {  cout << "单词表如下：" << endl;  displayTrieWords(trie.root);  }   1. Picture.cpp   #define \_CRT\_SECURE\_NO\_WARNINGS 1  #include<graphics.h>  #include<conio.h>  #include<stdio.h>  #include<bits/stdc++.h>  #include"oper.h"  #include"Ntree.h"  //界面菜单  enum MenuState {  MAIN\_MENU,//主界面  WORD\_QUERY1,//单词整体查询  WORD\_QUERY2,//单词前缀查询  DISPLAY\_WORD\_LIST//查看单词表  };  MenuState currentState = MAIN\_MENU;//当前界面  //窗口初始化  void init() {  initgraph(700, 700);  IMAGE mage;  loadimage(&mage, \_T("111.jpg"));  putimage(0, 0, &mage);  settextcolor(BLACK);  setbkmode(TRANSPARENT);  }  //主界面初始化  void initMenu() {  settextstyle(55, 0, \_T("楷体"));  fillrectangle(150, 100, 550, 200);  char arr[] = "单词整体查询";  int w = textwidth(arr);  int h = textheight(arr);  int x1 = 200 - w / 2;  int y1 = 50 - h / 2;  outtextxy(150 + x1, 100 + y1, \_T("单词整体查询"));  fillrectangle(150, 300, 550, 400);  outtextxy(150 + x1, 300 + y1, \_T("单词前缀查询"));  fillrectangle(150, 500, 550, 600);  char arr1[] = "查看单词表";  w = textwidth(arr1);  x1 = 200 - w / 2;  outtextxy(150 + x1, 500 + y1, \_T("查看单词表"));  }  //单词整体查询  void drawWordQuery1() {  init();  settextstyle(50, 0, \_T("楷体"));  outtextxy(60, 100, \_T("请输入要查询的单词："));  setlinecolor(BLACK);  rectangle(200, 200, 500, 280);  settextstyle(30, 0, \_T("楷体"));  outtextxy(610, 650, \_T("返回"));  setlinecolor(BLACK);  rectangle(600, 650, 680, 680);  }  //单词前缀查询  void drawWordQuery2() {  init();  settextstyle(50, 0, \_T("楷体"));  outtextxy(60, 100, \_T("请输入要查询的单词前缀："));  setlinecolor(BLACK);  rectangle(200,200,500,280);  settextstyle(30, 0, \_T("楷体"));  outtextxy(610, 650, \_T("返回"));  setlinecolor(BLACK);  rectangle(600, 650, 680, 680);  }  //查看单词表  void drawDisplayWordList() {  init();  settextstyle(30, 0, \_T("楷体"));  outtextxy(30, 20, \_T("单词表如下："));  setlinecolor(BLACK);  settextstyle(30, 0, \_T("楷体"));  outtextxy(610, 20, \_T("返回"));  setlinecolor(BLACK);  rectangle(600, 20, 680, 50);  //确定表格的位置和大小  settextstyle(20, 0, \_T("楷体"));  int startX = 40,startY = 60;  int cellWidth = 130,cellHeight = 25;  //绘制表格框架  for (int i = 0; i <= 25; ++i) {  line(startX, startY + i \* cellHeight, startX + 5 \* cellWidth, startY + i \* cellHeight); // 横线  }  for (int i = 0; i <= 5; ++i) {  line(startX + i \* cellWidth, startY, startX + i \* cellWidth, startY + 25 \* cellHeight); // 竖线  }  //填充  for (int i = 0; i < num; i++) {  outtextxy(startX + 10, startY + (i%25) \* cellHeight + 9, v[i].word.c\_str());  if ((i+1) % 25 == 0) {  startX += 130; startY = 60;  }  }  }  //鼠标点击  void handInput(int x, int y) {  if (currentState == MAIN\_MENU) {//主界面  if (x > 150 && x < 550 && y > 100 && y < 200) {  currentState = WORD\_QUERY1;  cleardevice();  drawWordQuery1();//单词整体查询界面  }  else if (x > 150 && x < 550 && y > 300 && y < 400) {  currentState = WORD\_QUERY2;  cleardevice();  drawWordQuery2();//单词前缀查询界面  }  else if (x > 150 && x < 550 && y > 500 && y < 600) {  currentState = DISPLAY\_WORD\_LIST;  cleardevice();  drawDisplayWordList();//查看单词表  }  }  else if (currentState == WORD\_QUERY1 || currentState == WORD\_QUERY2) {//两个查询界面  if (x > 600 && x < 680 && y>650 && y < 680) {//返回主界面  cleardevice();  init();  initMenu();  currentState = MAIN\_MENU;  }  else if (x > 200 && x < 500 && y>200 && y < 280) {//单词查询  line(205, 205, 205, 275);  string s;  cin >> s;  settextstyle(50, 0, \_T("楷体"));  outtextxy(200, 200, s.c\_str());  //整体查询  if (currentState == WORD\_QUERY1) {  if (Ti.search1(s)) {//单词存在  vector<int> p = Ti.searchWord1(s);  settextstyle(30, 0, \_T("楷体"));  outtextxy(60, 400, \_T("该单词所在行号为："));  cout << "该单词所在行号为： ";  int w = textwidth("0 ");  int x = 350;  for (int pos : p) {  string ss = to\_string(pos);  outtextxy(x + w, 400, ss.c\_str());  cout << pos << " ";  x = x + w;  }  cout << endl;  }  else {//单词不存在  outtextxy(60, 400, \_T("该单词不存在！"));  cout<<"该单词不存在！" << endl;  }  }  //前缀查询  else if (currentState == WORD\_QUERY2) {  if (Ti.search2(s)){//单词存在  vector<wordNode> p = Ti.searchWord2(s);  settextstyle(25, 0, \_T("楷体"));  outtextxy(60, 300, \_T("可能的完整单词有："));  cout << "可能的完整单词有： "<<endl;  int k = 1, w=textheight("0");  int y = 340;  settextstyle(20, 0, \_T("楷体"));  for ( auto i:p) {  int x = 320;  string ss = '(' + to\_string(k) + ')';  outtextxy(60, y, ss.c\_str());  outtextxy(120, y, i.word.c\_str());  cout << "(" << k << ") " << i.word << " 所在行号为： ";  outtextxy(220, y, \_T("所在行号为："));  for (auto j : i.positions) {  string sss = to\_string(j);  outtextxy(x + w, y, sss.c\_str());  cout << j << " ";  x = x + w;  }  cout << endl;  k++; y += w;  }  }  else {//单词不存在  outtextxy(60, 400, \_T("该单词不存在！"));  cout << "该单词不存在！" << endl;  }  }  }  }  }  int main() {  string filename = "text.txt";  indexTextFile(filename, Ti);//读取文本文档并索引化    init();  initMenu();  displayTrieWords(Ti);//构造单词表  while (true) {//鼠标点击  MOUSEMSG m;  m = GetMouseMsg();  if (currentState==MAIN\_MENU && m.uMsg == WM\_LBUTTONDOWN) {//主界面  handInput(m.x, m.y);  }  else if ((currentState == WORD\_QUERY1 || currentState == WORD\_QUERY2) && m.uMsg == WM\_LBUTTONDOWN) {//两个查询界面  handInput(m.x, m.y);  }  else if (currentState == DISPLAY\_WORD\_LIST && m.uMsg == WM\_LBUTTONDOWN) {//单词表查找界面返回主界面  if (m.x > 600 && m.x < 680 && m.y>20 && m.y < 50) {  cleardevice();  init();  initMenu();  currentState = MAIN\_MENU;  }  }  }  } |