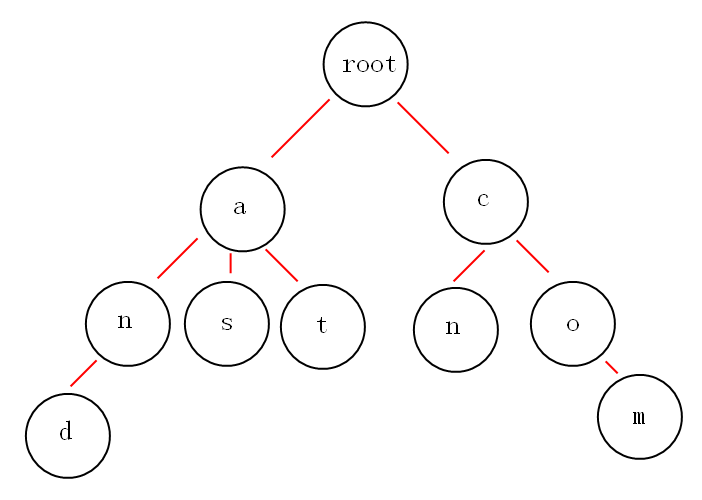
Trie



结构

50 /// Trie树节点

52 public class TrieNode

53 {

55 /// 26个字符，也就是26叉树

57 public TrieNode[] childNodes;

60 /// 词频统计

62 public int freq;

70 /// 途径该节点的单词的id组成的hashset

72 public HashSet<int> hashSet;

75 /// 初始化

77 public TrieNode()

78 {

79 childNodes = new TrieNode[26];

hashSet = new HashSet<int>();

80 freq = 0;

81 }

82 }

增加

95 /// 插入操作

101 public void AddTrieNode(ref TrieNode root, string word, int id)

102 {

103 if (word.Length == 0)

104 return;

106 //求字符地址，方便将该字符放入到26叉树中的哪一叉中

107 int k = word[0] - 'a';

109 //如果该叉树为空，则初始化

110 if (root.childNodes[k] == null)

111 {

112 root.childNodes[k] = new TrieNode();

114 //记录下字符

115 root.childNodes[k].nodeChar = word[0];

116 }

118 //该id途径的节点

119 root.childNodes[k].hashSet.Add(id);

120

121 var nextWord = word.Substring(1);

122

123 //说明是最后一个字符，统计该词出现的次数

124 if (nextWord.Length == 0)

125 root.childNodes[k].freq++;

126

127 AddTrieNode(ref root.childNodes[k], nextWord, id);

128 }

删除

258 /// 删除操作

264 public void DeleteTrieNode(ref TrieNode root, string word, int id)

265 {

266 if (word.Length == 0)

267 return;

268

269 //求字符地址，方便将该字符放入到26叉树种的哪一颗树中

270 int k = word[0] - 'a';

271

272 //如果该叉树为空,则说明没有找到要删除的点

273 if (root.childNodes[k] == null)

274 return;

275

276 var nextWord = word.Substring(1);

277

278 //如果该前缀单独占一个分支，则删除

279 if (root.childNodes[k].hashSet.size()==1

&&root.childNodes[k].hashSet.find(id))

280 {

Root.childNodes[k]=null;

return;

}

282 //删除途经节点id

283 root.childNodes[k].hashSet.Remove(id);

285 DeleteTrieNode(ref root.childNodes[k], nextWord, id);

286 }

改

234 public void UpdateTrieNode(ref TrieNode root

, string newWord, string oldWord, int id)

235 {

236 //先删除

237 DeleteTrieNode(oldWord, id);

238

239 //再添加

240 AddTrieNode(newWord, id);

241 }

查找

145 /// 检索单词的前缀,返回改前缀的Hash集合

150 public HashSet<int> SearchTrie(ref TrieNode root

, string word)

151 {

155 int k = word[0] - 'a';

157 var nextWord = word.Substring(1);

//如果前缀不匹配

if(root.childNodes[k]==null)

return null;

158 //前缀词查完了

159 if (nextWord.Length == 0)

160 {

162 return (root.hashSet);

163 }

//recursive calls

165 return SearchTrie(ref root.childNodes[k], nextWord);

168 }

词频

174 /// 统计指定单词出现的次数

179 public int WordCount(string word)

180 {

181 int count = 0;

182

183 WordCount(ref trieNode, word, ref count);

184

185 return count;

186 }

189 /// 统计指定单词出现的次数

195 public void WordCount(ref TrieNode root, string word, ref int count)

196 {

197 if (word.Length == 0)

198 return;

199

200 int k = word[0] - 'a';

201

202 var nextWord = word.Substring(1);

203

204 if (nextWord.Length == 0)

205 {

206 //采用动态规划的思想，word最后节点记录这途经的id

207 count = root.childNodes[k].freq;

208 }

209

210 WordCount(ref root.childNodes[k], nextWord

, ref count);

211 }