数据结构实验报告

**【算法说明】**

**哈希算法：**

1.使用散列表/哈希表（数组）存储数据。

2.每一个数据都可以计算出一个哈希值，这个哈希值决定了这个数据存储的位置，哈希值也就是哈希表数组相应的下标，通过这样按值寻址的方式，搜索数据的时间复杂度为常数，极大提高了搜索的效率。

3.当哈希值相同也就是遇到冲突的时候，通过将数值存储在其他位置来解决冲突（可将产生冲突的值存到该数组的其他位置，新建一个数组储存或将链表链接在产生冲突的地方储存发生冲突的数据）

**从文件中获取字符**

**遍历，去掉其它符号用两个字符的值来判断是否为单引号**

**对字符进行处理**

**得到单词**

**哈希值为：第一位字符\*1000+第四位\*单词长度\*10+单词长度**

**计算单词的哈希值并将 其存储到相应位置**

**该位置无数据**

**该位置已有数据**

**比较数据是否相同**

**将数据存入该位置**

**向后遍历数组**

**该位置对应的单词计数加一**

**到达数组尾部**

**使哈希值等于0**

**【测试结果】**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **测 试 用 例 表** | | | | | |
|  | 单词总数 | 单词和数量 | 堆积比例 | 散列表实现运行时间 | 数组实现运行时间 |
| 测试输入 | Input.txt | | | | |
| 测试目的 | 统计不同单词数 | 统计词频 | 有多少哈希值重复 | 比较运行时间 | 比较运行时间 |
| 实际输出 | 19330 | 略 | 6 | 29179ms | >5mins |
| 错误原因 | 无 | 无 | 无 | 无 | 无 |
| 当前状态 | 良好 | 良好 | 良好 | 良好 | 良好 |

**【分析与报告】**

1. 测试结果分析
2. 通过散列表可以极大的提升程序的时间效率
3. 哈希值的选取不同会有不同的时间效率
4. 堆积比例低的运行时间不一定短（在更改哈希值时将哈希值改成第一位字符\*1000+第四位\*单词长度\*10+第一位字符%单词长度+单词长度，堆积比例为7，但是时间变为271011ms）
5. 探讨更多问题解决的途径
6. 可以通过改变哈希值来减小堆积比例
7. 在解决冲突时，可以通过链表来储存冲突的数据，在读取链表中的数据时需要通过for循环来使指针逐位向后移动（冲突的数据不能太多，否则会使时间复杂度过大）
8. 在解决冲突时，可以通过再建一个数组s来储存冲突的数据，如果哈希值相同两个数据不同，就在s中查找或者储存。

**【附录】**

**散列表代码：**

#include<iostream>

#include<fstream>

#include<string>

#include<time.h>

using namespace std;

//建立哈希表

//在main函数外建立哈希表使s和number被自动初始化

struct words

{

string s;

int number;

};

words hashtable[30000];

int main()

{

//打开文件

ifstream input;

ofstream output;

input.open("input.txt");

output.open("output.txt", ofstream::out);

if (!input)

cerr << "unable to open";

if (!output)

cerr << "unable to write";

string str;//用于处理字符

int hashnumber;//哈希值，数组下标

int first;//第一个字母0-26

int fourth;//第四个字母0-26

int wordlen;//单词长度

int temp=0;//检测是否已经将数据放入哈希表

int sum = 0;//总单词数

char chr;//储存字符

int conflict=0;//累计冲突次数

int lastchr;//用于检测全角的单引号

clock\_t start\_time = clock();

{

while (!input.eof())

{

str = "";//清空str

while (!input.eof())//对输入的字符进行处理变成单词

{

input.get(chr);

if (chr < 65)//65以下的都不是字母

{

if (str != "" && chr == '-')//保留“-”

str = str + chr;

else if (str != "" && (chr == -95 || chr == -81))//全角单引号用两个chr值来检测

{

if (chr == -95)

lastchr = 1;

else if (chr == -81 && lastchr == 1)

str = str + "'";//将全角单引号转换成半角

else

break;//如果只有一半则不是单引号

}

else

break;//其他符号表示这个单词已经结束，跳出循环

}

else

str = str + chr;

}

if (str != "")//避免开头就是符号或者多个符号相连

{

temp = 0;//temp=0数据未放入哈希表

//字符的ascll码减去相应数值

//大写字母在0-25之间，小写字母在1-26间

//按照A对应0，a对应1，以此类推

//第一位字符

if (str[0] < 96)//大写字母

first = (int)str[0] - 65;

else

first = (int)str[0] - 96;

//第四位字符

wordlen = str.length();

if (wordlen > 3 && str[3] > 64)//判断第四位字符是否为字母

{

if (str[3] > 96)//小写字母

fourth = (int)str[3] - 96;

else

fourth = (int)str[3] - 65;

}

else//当第四位字母不存在或者为连字符或单引号是，将第四位的值看作是0

fourth = 0;

//计算哈希值

hashnumber = first \* 1000 + fourth \*wordlen\* 10 + wordlen;

//哈希值为第一位字符\*1000+第四位\*单词长度\*10+单词长度

//将str放入哈希表的相应位置，如果该位置有数据

if (hashtable[hashnumber].number != 0)

{

if (hashtable[hashnumber].s == str)//数据相同

hashtable[hashnumber].number++;

else

{

conflict++;

while (hashtable[hashnumber].number != 0)//遍历后面直到找到相同数据或空位

{

hashnumber++;

if (hashtable[hashnumber].s == str)//找到了相同的数据

{

hashtable[hashnumber].number++;

temp = 1;

break;

}

if (hashnumber == 29999)//如果末尾挤满了

hashnumber = 0;//在前面找位置

}

if (temp == 0)//遍历时未发现相同数据

{

hashtable[hashnumber].s = str;

hashtable[hashnumber].number = 1;

}

}

}

else//哈希值对应位置上无数据

{

hashtable[hashnumber].s = str;

hashtable[hashnumber].number++;

}

}

}

for (int i = 0; i < 30000; i++)//将哈希表上有数据的地方输出

{

if (hashtable[i].number != 0)

{

output << hashtable[i].s << " " << hashtable[i].number << endl;

sum++;//计算总单词数

}

}

}

clock\_t end\_time = clock();

output << "单词数" << sum << " " << "冲突" << conflict << " " << "堆积比例" << conflict / 30000 << endl;

output << "运行时间" << static\_cast<double>(end\_time - start\_time) / CLOCKS\_PER\_SEC \* 1000 << "ms";

}

**数组实现代码：**

#include<iostream>

#include<fstream>

#include<string>

using namespace std;

int main()

{

ifstream input;

ofstream output;

input.open("input.txt",ios::in);

output.open("output.txt", ofstream::out);

if (!input)

{

cerr << "unable to open";

}

if (!output)

{

cerr << "unable to write";

}

struct word

{

string s;

int number;

};

//word store[3000];

int length = 30000;

word\*store = new word[length];

string str;

int i = 0;

int temp = 0;

while (!input.eof())

{

getline(input, str,' ');

temp = 0;

if (i == 0)

{

store[0].s=str;

i++;

store[0].number = 1;

}

else {

for (int j = i; j > 0; j--)

{

if (str == store[j - 1].s)

{

store[j - 1].number++;

temp = 1;

break;

}

}

if (temp == 0)

{

store[i].s = str;

store[i].number = 1;

i++;

}

}

if (i == length-1)

{

int needed = length\*1.5 - length;

length = length\*1.5;

word\*temp = new word[length];

copy(store, store+needed, temp);

delete[]store;

store = temp;

}

}

for (int j = 0; j < i; j++)

output << store[j].s << store[j].number<<endl;

system("pause");

}