计算机应用编程实验

Hashtable字符串检索

熊永平@计算机学院

ypxiong@bupt.edu.cn

教三楼132

2020.9

实验背景

- 一些实际问题
  - 如何实现在搜索引擎输入框自动提示？
  - 一个查询串的重复度越高，说明查询它的用户越多，也就越热门。大型搜索引擎每小时有几十亿个查询请求，如何统计搜索引擎最热门前100个查询？
  - 在实现一个编辑器时，如何对输入的单词进行拼写检查？
  - GFW每次google中输入**词后，如何在**名单中查找并reset？
  - 搜索引擎的网络爬虫，每天要爬取几十亿网页，哪些URL是爬过的？
  - 收到一封邮件后，能否快速在几亿个垃圾邮件黑名单地址里快速判断发件人是否在黑名单里面？
  - 检测引擎中包含几千万条特征字符串规则，如何在10G网络流量环境下检测网络流中的恶意软件特征？

海量字符串检索

- **问题**
  - 在给定的海量个数的字符串中查找特定的字符串
- **挑战**
  - 实际需求
    - 几亿规模
  - 数据量大
    - 200,000,000量级
  - 外存便宜
    - 存储成本低
  - 内存不够大？
    - 200,000,000*40bytes=8000,000,000bytes=8G

哈希表

- **思想**
  - 根据设定的哈希函数 H(key) 和所选中的处理冲突的方法，将一组关键字映象到一个有限的、地址连续的地址集 (区间) 上，并以关键字在地址集中的“象”作为相应记录在表中的存储位置，如此构造所得的查找表称之为“哈希表”。
  - 哈希表所得的存储位置成为哈希地址或者散列地址
- **冲突处理方法：拉链法**
  - 将具有相同哈希地址的记录链成一个单链表，m个哈希地址就设m个单链表，然后用一个数组将m个单链表的表头指针存储起来，形成一个动态的结构。
  - 优点：插入、删除方便
  - 缺点：占用存储空间多

哈希表举例

例：已知一组关键字 (19, 14, 23, 1, 68, 20, 84, 27, 55, 11, 10, 79)

哈希函数为：H(key)=key MOD 13，用链地址法处理冲突。

哈希表分析

- **查找过程**
  - 利用线性表存储集合元素
  - 利用Hash函数计算元素对应的地址entry，并在对应的区域存储该元素
  - 实际查找通过先hash计算entry，再遍历链表匹配元素是否相同（字符串匹配，必须逐个字符比较）
- **问题**
  - 解决冲突：选择合适的hash函数
  - 合适的哈希表大小

经典的Hash算法

- 常用的**HASH**算法
  - unsigned int RSHash (char* str, unsigned int len);
  - unsigned int JSHash (char* str, unsigned int len);
  - unsigned int PJWHash (char* str, unsigned int len);
  - unsigned int ELFHash (char* str, unsigned int len);
  - unsigned int BKDRHash(char* str, unsigned int len);
  - unsigned int SDBMHash(char* str, unsigned int len);
  - unsigned int DJBHash (char* str, unsigned int len);
  - unsigned int DEKHash (char* str, unsigned int len);
  - unsigned int BPHash (char* str, unsigned int len);
  - unsigned int FNVHash (char* str, unsigned int len);
  - unsigned int APHash (char* str, unsigned int len);

ELFHASH举例

- **HASH函数**

```
unsigned int ELFHash(char *str)
{
   unsigned int hash = 0;
   unsigned int x = 0;
   while (*str)
   {
      hash = (hash << 4) + (*str++);//hash左移4位，把当前字符ASCII存入hash低四位。
      if ((x = hash & 0xF0000000L) != 0)
      {
         //如果最高的四位不为0，则说明字符多余7个，现在正在存第7个字符，如果不处理，再加下一个字符时，第一个字符会被移出，因此要有如下处理。
         //该处理，如果最高位为0，就会仅仅影响5-8位，否则会影响5-31位，因为C语言使用的算数移位
         //因为1-4位刚刚存储了新加入到字符，所以不能>>28
         hash ^= (x >> 24);
         //上面这行代码并不会对X有影响，本身X和hash的高4位相同，下面这行代码&~即对28-31(高4位)位清零。
         hash &= ~x;
      }
   }
   //返回一个符号位为0的数，即丢弃最高位，以免函数外产生影响。(我们可以考虑，如果只有字符，符号位不可能为负)
   return (hash & 0x7FFFFFFF);
}
```

程序要求

- 程序命名为
  - hashtable_search
- 输入数据
  - 词典串dict.txt：127万个
  - 待匹配的1.7万个字符串：string.txt
- 实验结果**result.txt**
  - 所有查找到的串，一行一个
    - Keyword1
    - Keyword2

报告要求

- **实验报告**
  - 主要数据结构和流程
  - 实验过程
  - 遇到的问题
  - 结果指标：cpu 内存 准确率等
  - 结论和总结

THE END