高级计算机编程实战

Hashtable字符串检索

熊永平@计算机学院

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

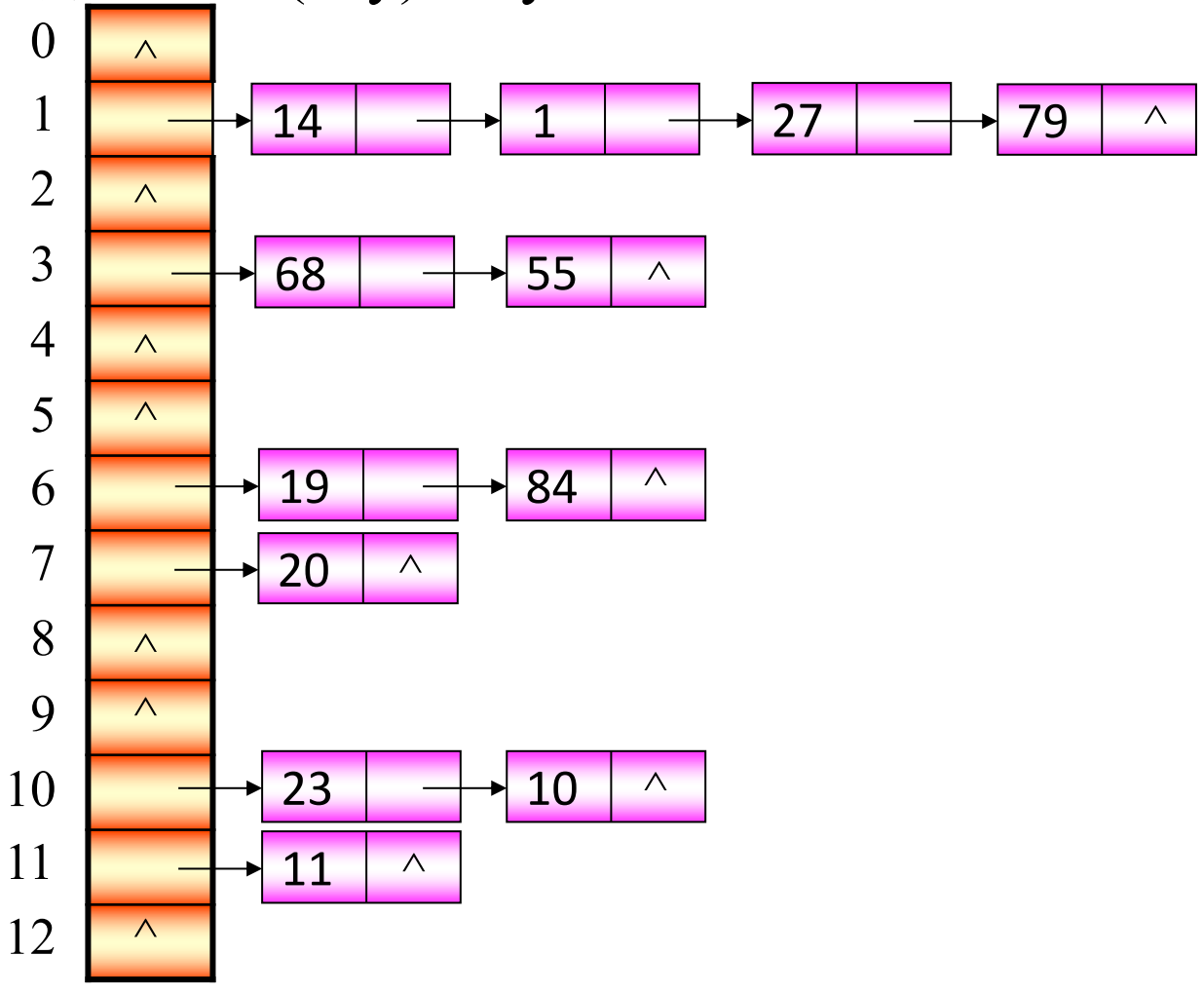
哈希表

- **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哈希地址
- **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)

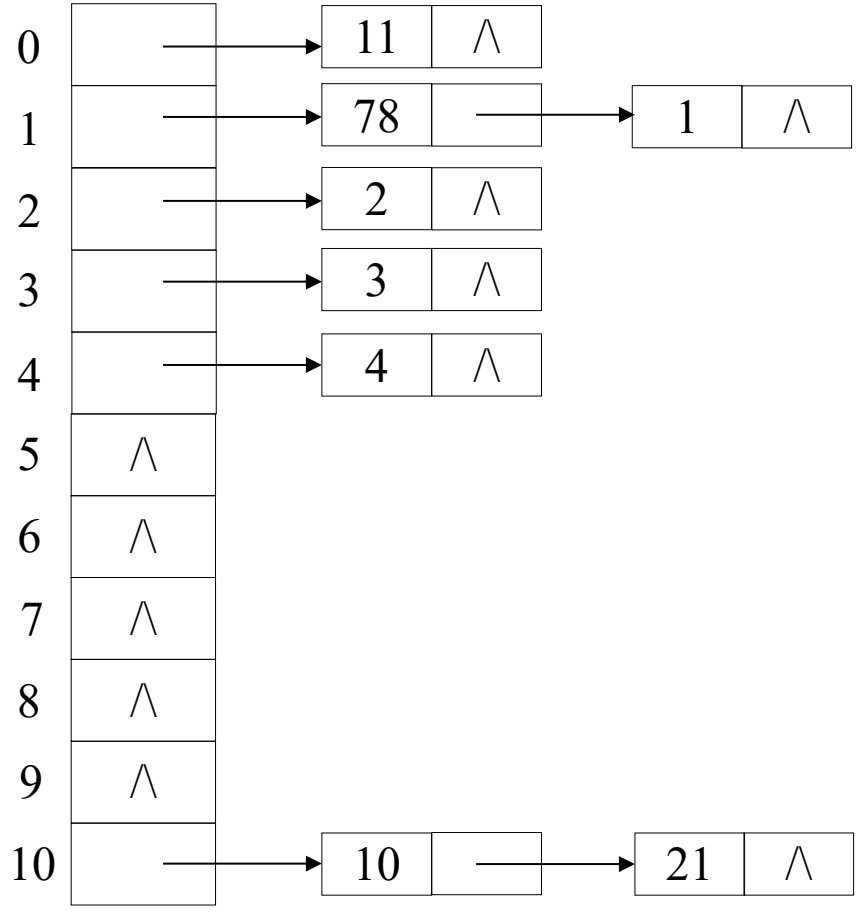
哈希函数为：H(key)=key MOD 13，用链地址法处理冲突。



哈希表

❖哈希查找的性能分析

{11,78,10,1,3,2,4,21}



链地址法成功平均查找长度为

ASL=(1*6+2*2)/8=1.25

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

哈希表的查找分析

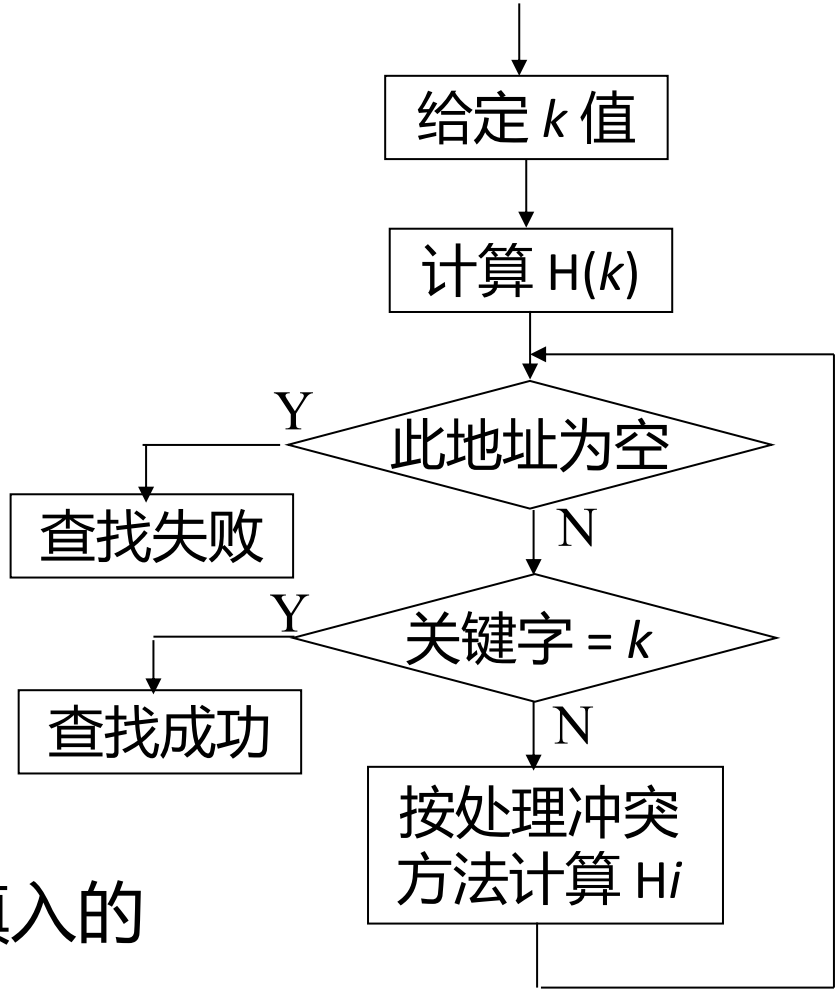
查找过程和造表过程一致。假设采用开放定址处理冲突，则查找过程为：

哈希查找分析：

- 哈希查找过程仍是一个给定值与关键字进行比较的过程；
- 评价哈希查找效率仍要用ASL；
- 决定哈希表查找的ASL的因素：

1) 选用的哈希函数；
2) 选用的处理冲突的方法；
3) 哈希表饱和的程度，装载因子

$\alpha = n/m$ 值的大小（n —— 表中填入的记录数，m —— 哈希表的长度）



哈希表

❖哈希函数的构造方法

实际造表时，采用何种构造哈希函数的方法取决于建表的关键字集合的情况（包括关键字的范围和形态），总的原则是使产生冲突的可能性降到尽可能地小。

选取哈希函数，考虑以下因素：

1）计算哈希函数所需时间

2）关键字长度

3）哈希表长度（哈希地址范围）

4）关键字分布情况

5）记录的查找频率

哈希表

❖哈希函数的构造方法

字符串当成数字，关键字可有下列构造方法：

1. 直接定址法　2. 数字分析法　3. 平方取中法
4. 折叠法　5. 除留余数法　6. 随机数法

字符串哈希函数有

BKDRHash，APHash，DJBHash，JSHash，
RSHash，SDBMHash，PJWHash，ELFHash

1、直接定址法

此类函数取关键码的某个线性函数值作为哈希地址：

Hash(key) = a*key+b　{a, b为常数}

这类哈希函数是一对一的映射，一般不会产生冲突。但它要求哈希地址空间的大小与关键码集合的大小相同。

- 示例：有一组关键码如下：{ 942148, 941269, 940527, 941630, 941805, 941558, 942047, 940001 }。哈希函数为

  Hash(key) = key-940000

***Hash* (942148) = 2148** ***Hash* (941269) = 1269**

***Hash* (940527) = 527** ***Hash* (941630) = 1630**

***Hash* (941805) = 1805** ***Hash* (941558) = 1558**

***Hash* (942047) = 2047** ***Hash* (940001) = 1**

- 可以按计算出的地址存放记录。

数字分析法

- 设有 n 个 d 位数，每一位可能有 r 种不同的符号。这 r 种不同符号在各位上出现的频率不一定相同。根据哈希表的大小，选取其中各种符号分布均匀的若干位作为哈希地址。
- 计算各位数字中符号分布均匀度λk的公式：

$$\lambda_k = \sum_{i=1}^{r} \left(\circ_i^k - n/r \right)^2$$

- 其中，$\circ_i^k$ 表示第 i 个符号在第 k 位上出现的次数，n/r 表示各种符号在 n 个数中均匀出现的期望值。

- 计算出的 λ_k 值越小，表明在该位 (第 k 位) 各种符号分布得越均匀。

9	**4**	**2**	**1**	**4**	**8**	**①位，$\lambda_1 = 57.60$**
9	**4**	**1**	**2**	**6**	**9**	**②位，$\lambda_2 = 57.60$**
9	**4**	**0**	**5**	**2**	**7**	**③位，$\lambda_3 = 17.60$**
9	**4**	**1**	**6**	**3**	**0**	**④位，$\lambda_4 = 5.60$**
9	**4**	**1**	**8**	**0**	**5**	**⑤位，$\lambda_5 = 5.60$**
9	**4**	**1**	**5**	**5**	**8**	**⑥位，$\lambda_6 = 5.60$**
9	**4**	**2**	**0**	**4**	**7**	
9	**4**	**0**	**0**	**0**	**1**	
①	**②**	**③**	**④**	**⑤**	**⑥**	

- 若哈希表地址范围有 3 位数字, 取各关键码的 ④⑤⑥ 位做为记录的哈希地址。

- 数字分析法仅适用于事先明确知道表中所有关键码每一位数值的分布情况，它完全依赖于关键码集合。如果换一个关键码集合，选择哪几位要重新决定。

除留余数法

设哈希表中允许地址数为m，取一个不大于 m，但最接近于或等于 m 的质数 p 作为除数，用以下函数把关键码转换成哈希地址：

$$hash(key) = key \% p \qquad p \le m$$

其中，“%”是整数除法取余的运算，要求这时的质数 p 不是接近 2 的幂。

- 示例: 有一个关键码 key = 962148，哈希表大小 m = 25，即 HT[25]。取质数 p = 23。哈希函数 hash(key) = key % p。则哈希地址为

  hash(962148) = 962148 % 23 = 12。

- 可按计算出的地址存放记录。注意，使用哈希函数计算出的地址范围是 0 到 22，而 23、24 这几个地址实际上不能用哈希函数计算出来，只能在处理冲突时达到这些地址。

平方取中法

- 它首先计算构成关键码的标识符的内码的平方，然后按照哈希表的大小取中间的若干位作为哈希地址。
- 设标识符可以用一个计算机字长的内码表示。因为内码平方数的中间几位一般是由标识符所有字符决定，所以对不同的标识符计算出的哈希地址大多不相同。
- 在平方取中法中，一般取哈希地址为8的某次幂。例如，若哈希地址总数取为 $m = 8^r$，则对内码的平方数取中间的 r 位。

标识符	内码	内码平方	哈希地址
A	01	01	001
*A*1	0134	20420	042
*A*9	0144	23420	342
B	02	04	004
DMAX	04150130	21526443617100	443
*DMAX*1	0415013034	526447352151420	352
AMAX	01150130	1354236617100	236
*AMAX*1	0115013034	345424652151420	652

标识符的八进制内码表示及其平方值和哈希地址

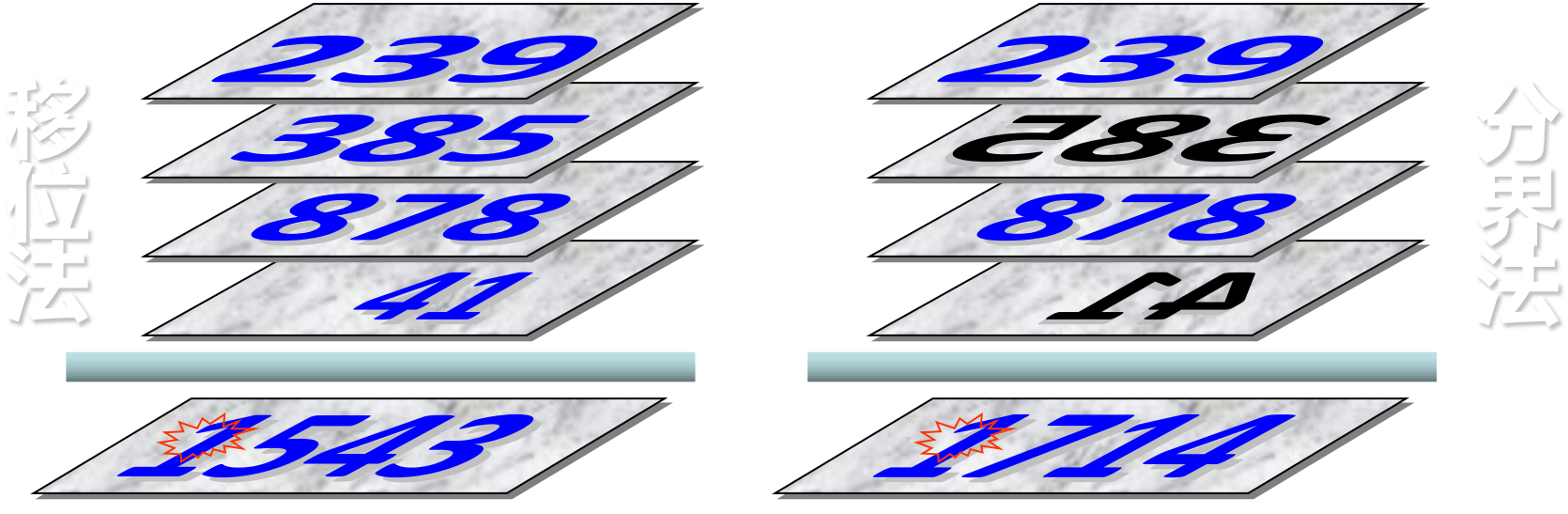
折叠法

- 此方法把关键码自左到右分成位数相等的几部分，每一部分的位数应与哈希表地址位数相同，只有最后一部分的位数可以短一些。
- 把这些部分的数据叠加起来，就可以得到具有该关键码的记录的哈希地址。
- 有两种叠加方法：

  移位法：把各部分最后一位对齐相加；

  分界法：各部分不折断，沿各部分的分界来回折叠，然后对齐相加。

- 示例: 设给定的关键码为 key = 23938587841, 若存储空间限定 3 位, 则划分结果为每段 3 位。 上述关键码可划分为 4 段：
- 239 385 878 41
- 叠加，然后把超出地址位数的最高位删去, 仅保留最低的3 位，做为可用的哈希地址。



- 一般当关键码的位数很多，而且关键码每一位上数字的分布大致比较均匀时，可用这种方法得到哈希地址。

- 假设地址空间为HT[400]，利用以上函数计算，取其中3位，取值范围在0～999，可能超出地址空间范围，为此必须将0～999压缩到0～399。可将计算出的地址乘以一个压缩因子0.4，把计算出的地址压缩到允许范围。

字符串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：约120万
  - 待匹配字符串：words.txt
- 实验结果result.txt
  - 所有查找到的串，一行一个
    - Keyword1
    - Keyword2
- 计分指标
  - 给出4个值
  - 装载因子a分别为：1、2、10、50时的所有words中的字符串平均每个字符串所比较的字符串个数

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 平均比较次数等
  - 结论和总结


THE END