Chapter 9 Searching

Content

9.1 Basic concepts

Search(查找)

Given: Distinct(无重复的) keys $k_1, k_2, \ldots, k_n$ and collection *Table* of n records of the form

$(k_1, R_1), (k_2, R_2), \ldots, (k_n, R_n)$

here R_j is the record information associated with key k_j for $1 <= j <= n$.

Search Problem: For key value K, locate the record (k_j, R_j) in *Table* such that $k_j = K$.

Search的评价指标：平均查找长度ASL (Average Searching Length).
查找若干个记录需要的平均关键字比较次数

Successful vs. Unsuccessful

A successful search is one in which a record with key $k_j = K$ is found.

An unsuccessful search is one in which no record with $k_j = K$ is found (and presumably no such record exists).

more expensive

Extract-match query vs. range query
精确查找vs范围查找

An **extract-match** query is a search for the record whose key value matches a **specified key** value

A **range query** is a search for **all** records whose key value falls within **a specified range of key** values

Approaches to Search

1. Sequential search (顺序查找)

给定值依次和集合中各个记录的关键字进行比较，
ASL取决于集合中记录的组织方式/顺序

- Records Ordered by insert order
- Records Ordered by search Frequency

How to decide?

2. Binary search (二分/折半查找)

- Records Ordered by key values

3. Direct access by key value (hashing 直接查找)

本章重点

4. Tree indexing methods(索引查找).

将在chapter 10 介绍

9.2 Searching in the sorted Records

9.2.1 Records Ordered by key values

9.2.2 Records Ordered by search Frequency

9.2.3 Self-Organizing Lists

9.2.1 Records Ordered by key values
---Using binary search(折半查找)

45

Position	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Key	11	13	21	26	29	36	40	41	45	51	54	56	65	72	77	83

4 3 1 2

O(log(n))

ASL =0.5 log(n)

思考：折半查找适合对存放在LList的有序集合进行查找吗？

9.2.1 Records Ordered by key values
---Using binary search(折半查找)

```
int binary(int array[], int n, int K) {
   int l = -1;
   int r = n;     // l, r are beyond array bounds
   while ( (l+1) != r)  // Stop when l, r meet
   {
       int i = (l+r)/2;     // Check middle
       if (K < array[i])    r = i;   // Left half
       if (K == array[i]) return i; // Found it
       if (K > array[i])    l = i;   // Right half
   }
   return n; // Search value not in array
}
```

思考：折半查找适合对存放在LList的有序集合进行查找吗？

9

9.2.2 Records Ordered by search Frequency

Order record by (expected) frequency of search occurrence.

- Perform sequential search

1. 事先通过实验统计得到
2. 实时动态更新

Expected search cost（ASL）：

$$\text{ALS} = 1p_1 + 2p_2 + \ldots + np_n.$$

$p_1, p_2, \ldots, p_n$ 为各记录被查找的归一化频度，
n 为记录集中记录条数

10

Frequency Distributions example

(1) All records have equal frequency. $\text{ASL} = \sum_{i=1}^{n} i / n = (n+1)/2$

(2) Exponential Frequency

$$p_i = \begin{cases} 1/2^i & 1 \le i \le n-1 \\ 1/2^{n-1} & i = n \end{cases} \qquad \text{ASL} \approx \sum_{i=1}^{n} (i / 2^i) \approx 2.$$

(3) 80/20 rule（实际最常见）：

- 80% of search are to 20% of the records.
- For distributions following 80/20 rule,

实时动态更新

$$\text{ASL} \approx 0.1n.$$

11

9.2.3 Self-Organizing Lists

- Self-organizing(自组织) lists modify the order of records within the list based on the actual pattern of record search.
- Self-organizing lists use a heuristic(启发) for deciding how to reorder the records.
  - Count: Order by actual historical frequency of search.
  - Move-to-Front: When a record is found, move it to the front of the list.
  - Transpose: When a record is found, swap it with the record ahead of it

original sequence: 0,1,2,3,4,5,6,7,8,9
Searching sequence 9, 9, 3, 0, 5, 5, 9

12

Direct access by key value

不论是顺序查找还是折半查找，不论记录是按关键字还是按查找频度排序，平均查找长度ASL (Average Searching Length) 肯定大于1。

- 对于需频繁查找的记录集合(查找表)，希望ASL = 0
- 只有一个办法：预先知道所查关键字在表中的位置。即要求：在关键字与记录在表中的存储位置之间建立一个确定的关系---hashing

13

9.3 Hashing

14

14

9.3.1 What is hashing(散列)

Hashing: The process of mapping(映射) a key value to a position in a table.

① A hash function maps key values to positions. It is denoted by *h*.

② A hash table(哈希表/散列表) is an array that holds the records. It is denoted by HT.

- ✓ HT has *M* slots/positions, indexed form 0 to *M*-1.

15

9.3.2 Hash Functions

Hash 函数构造原则

① MUST return a value within the hash table index range (0 ~ M-1).

② SHOULD evenly distribute(均匀分布) the records stored among the hash table slots.

16

Hash Functions Examples (1)

1) int h(int k) {
   return(k % M);
   }

除留余数法, H(k) = k%M
适合关键字的取值范围远大于哈希表长度 *M*.

2) Mid-square method: Square the key value, take the middle *r* bits from the result for a hash table of 2^r slots.

```
int h(int k) {
    return(k*k & 0x000FF000L) >>12);
}
```

平方取中法，适合于关键字为数字, r的大小取决于关键字的取值范围

17

Hash Functions Examples (2)

3) For strings: Sum the ASCII values of the letters and take results modulo *M*.

折叠法，适合于关键字为字符串或数字位数多, sum 远大于*M*

```
int h(char* k) {
   int i, sum;
   for (sum=0, i=0; k[i] != '\0'; i++)
      sum += (int) k[i];
   return(sum % M);
}
```

18

Hash Functions Examples (3)

4) ELF Hash: From Executable and Linking Format (ELF), UNIX System V Release 4.

```
int ELFhash(char* key) {
   unsigned long h = 0;
   while(*key) {
      h = (h << 4) + *key++;
      unsigned long  g = h & 0xF0000000L;
      if (g)  h ^= g >> 24;
      h &= ~g;  }
   return h % M;
}
```

19

构造哈希函数的其他方法

1) 直接定址法： h(key) = (a × key + b)%M

2) 随机数法： h(key) = Random(key)

3) 数字分析法：提取分布均匀的若干位或它们的组合作为地址

实际应用时采用何种构造哈希函数的方法取决于关键字集合的情况(包括关键字的取值范围和分布)，总的原则是使<u>*产生冲突的可能性降到尽可能地小。*</u>

20

Collisions（冲突）

- 哈希函数是一个压缩映射，在一般情况下，很容易产生“冲突 (Collisions) ” 现象，
  - 即：key1≠ key2， 而 h(key1) = h(key2)
- Collisions are inevitable in most applications.
  - h(k) = k %M，具有相同余数的关键字在HT中有冲突

很难找到一个不产生冲突的哈希函数。一般情况下，只能构建/选择恰当的哈希函数，使冲突尽可能少地产生；之外，还需要找到一种“处理冲突” 的方法。

21

9.3.3 collision resolution/处理冲突的方法

Each record i has a home position $h(k_i)$. If another record occupies i's home position

collisions occur

What to do? **How to do?**

“处理冲突” 的实际含义是

为产生地址冲突的记录寻找下一个哈希地址

1. Closed Hashing 闭域法/开放定址法
2. Open Hashing 开域法/链地址法

22

9.3.3.1 Closed Hashing(闭域法/开放定址法)

- **stores all records directly in the hash table.**

为每个记录求得一个地址序列(Probe sequence)

$H_0, H_1, H_2, \ldots, H_s$ $1 \le s \le m-1$, m：哈希表的长度

其中：H_0 = h(key) home position

$H_i = (H_0 + d_i) \% m$, $i=1, 2, \ldots, s$

Probe Function
探测函数 $P(i)=d_i$

- 增量序列d_i的几种取法

1) 线性探测：$d_i = i$

2) 平方探测：$d_i = 1^2, -1^2, 2^2, -2^2, \ldots$,

3) 随机探测：d_i 是一组伪随机数列

23

Pseudo-Random Probing

- Select a (random) permutation(序列) of the numbers from 1 to m-1 作为增量序列：

$$r_1, r_2, \ldots, r_{m-1}$$

- All insertions and searches use the same permutation.

Example: Hash table size of m = 101
- r_1=2, r_2=5, r_3=32.
- $h(k_1)$=30, $h(k_2)$=28.
- Probe sequence for k_1: 30, 32, 35, 62.
- Probe sequence for k_2: 28, 30, 33, 60.

24

Close Hashing example

{ 19, 01, 23, 14, 55, 68, 11, 82, 36 }　　h(key) = key % 11

若采用线性探测再散列处理冲突 $d_i = i$

	0	1	2	3	4	5	6	7	8	9	10
	55	01	23	14	68	11	82	36	19		
i	0	0	1	0	2	5	1	4	0		

冲突次数：13

若采用二次探测再散列处理冲突　$d_i = 1^2, -1^2, 2^2, -2^2, \ldots,$

	0	1	2	3	4	5	6	7	8	9	10
	55	01	23	14	36	82	68		19		11
i	0	0	1	0	1	0	3				2

冲突次数：7

25

Close Hashing example

若要再插入一个数，将其放置于剩余的2个空slot的概率分别为多大？

h(key) = key % 11

若采用线性探测再散列处理冲突 $d_i = i$

0	1	2	3	4	5	6	7	8	9	10
55	01	23	14	68	11	82	36	19		
								P：	10/11	1/11

若采用二次探测再散列处理冲突　$d_i = 1^2, -1^2, 2^2, -2^2, \ldots,$

0	1	2	3	4	5	6	7	8	9	10
55	01	23	14	36	82	68		19		11
						P：	5/11		6/11	

26

Secondary Clustering

Old Probe Function 探测函数：P(K, i) = d_i

If two keys hash to the same slot, they follow the same probe sequence. This is called secondary clustering（二次聚集）.

For example
h(key) = key % 11　采用线性探测再散列处理冲突　$d_i = i$

Key1 =12,　H_i：1, 2, 3, 4, 5, 6,…..
key2=23,　H_i：1, 2, 3, 4, 5, 6,…..
key3= 34,　H_i：1, 2, 3, 4, 5, 6,…..

二次聚集容易导致较多的冲突次数

27

Double hashing(双哈希)

Old Probe Function 探测函数：p(K, i) = d_i

To avoid secondary clustering, modify P(i) to be a function of the original key value K and count i:

New Probe Function P(K, i) = $d_i * h_2(K)$
哈希函数：h(K)　√
增量序列：d_i　√
二次哈希函数：$h_2(K)$

$H_i = (H_0 + P(K,i))\%\ m$
$= (h(k) + d_i * h_2(K))\%m, \quad i=1, 2, \ldots, s$
1　　2

28

Double Hashing（双哈希）

example1 : Hash table of size M=11，$h(k)$= k % M,

$p(K, i) = i * h_2(K)$ ，$h_2(K)$ = k % 5 +1,

k_1 = 12, k_2=23, k_3= 34,

- $h(k_1)$=1, $h(k_2)$=1, $h(k_3)$=1
- $h_2(k_1)$=3, $h_2(k_2)$=4, $h_2(k_3)$=5
- Probe sequence for k_1 is: 1, 4, 7, 10
- Probe sequence for k_2 is: 1, 5, 9, 13
- Probe sequence for k_3 is: 1, 6, 11, 16

OLD
1, 2, 3, 4
1, 2, 3, 4
1, 2, 3, 4

29

double hashing example

{ 19, 01, 23, 12, 55, 68, 24, 86, 35 }　h(key) = key % 11

若采用双哈希 $p(K, i) = i * h_2(K)$ 处理冲突，h2(K) = k % 5 +1

	0	1	2	3	4	5	6	7	8	9	10
	55	01	68	35	12	23		24	19	86	
i	0	0	0	1	1	1		1	0	0	

冲突总次数：4

30

Insertion a record into hash table (close hashing)

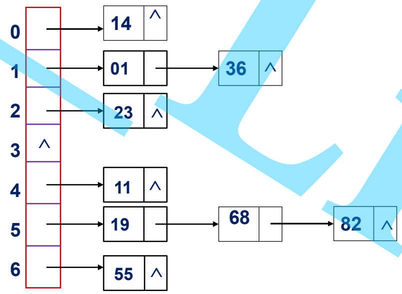
```
// Insert e into HT
template <class Key, class Elem,  class KEComp,  classEEComp>
hashInsert(const Elem& e) {
  int home;       // Home position for e
  int pos = home = h(e.key); // Init
  for (int i=1;   !(EEComp::eq(EMPTY, HT[pos])); i++) {
    pos = (home + p(e.key, i)) % M;
    if (EEComp::eq(e, HT[pos]))     return false;  // Duplicate
  }
  HT[pos] = e;      // Insert e
  return true;
}
```

31

9.3.3.2 Open Hashing(开域法)/链地址法

将所有哈希地址相同的记录都链接在同一链表中

{ 19, 01, 23, 14, 55, 68, 11, 82, 36 }　H(key) = key % 7



32

9.3.4 Searching in HT

Search for the record with key *K* in HT :

1. Compute the home table location *h*(*K*).
2. Starting with slot *h*(*K*), locate the record containing key *K* using (if necessary) a collision resolution policy.

33

Searching in HT using Closed hashing

1. 对于给定值 K，根据哈希函数及探测函数计算哈希地址
   $H_0=h(K), H_i=(H_0+P(K, i)) \% m, \ i=1,2,\ldots m-1;$
2. 从i=0开始，比较K ≟ HT[H_i]， 若等于，查找成功，停止
3. 否则，查找不成功

Example : m=11 h(key) = key % 11

$H_i = (H_0 + d_i) \% 11, \quad i=1, 2, \ldots, 10, \quad d_i = i$

0	1	2	3	4	5	6	7	8	9	10
55	01	23	14	68	11	82	36	19		

K=68   *i=2* ↑ 查找成功 SL=3

K=26   ↑ *i=5* SL=6 查找不成功

34

34

哈希表查找(Closed hashing)的代价分析

Example : 依次查找 { 19, 01, 23, 14, 55, 68, 11, 82, 36 }

m=11, H(key) = key % m

$H_i = (H_0 + d_i) \% m, \quad i=1, 2, \ldots, m-1, \quad d_i = i$

	0	1	2	3	4	5	6	7	8	9	10
	55	01	23	14	68	11	82	36	19		
SL	1	1	2	1	3	6	2	5	1		

平均查找长度ASL：(4*1+2*2+3+5+6)/9=2.44

通常ASL ≠ 0，这是因为有冲突存在

35

Search in HT using closed hashing

```
// Search for the record with Key K
template <class Key, class Elem, class KEComp, class EEComp>
hashSearch(const Key &K, Elem &e) const {
  int home;          // Home position for K
  int  pos = home = H(K); // Initial posit
  for (int i = 1; !KEComp::eq(K, HT[pos]) &&
         !EEComp::eq(EMPTY, HT[pos]); i++)
       pos = (home + p(K, i)) % m; // Next
  if (KEComp::eq(K, HT[pos])) {    // Found it
        e = HT[pos];
        return true;
  }
  else return false;  // K not in hash table
}
```

36

Searching in HT using Open hashing

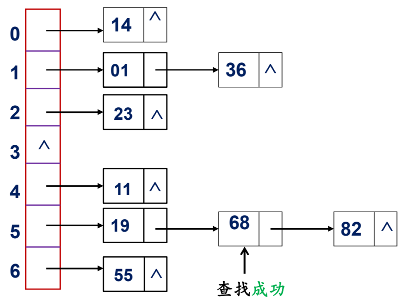
1. 对于给定值K，根据哈希函数计算哈希地址 H_0=H(K)
2. 从H_0对应链表的第一个结点开始，比较结点记录关键字是否与K相等

   若相等，查找成功，返回
3. 返回 查找不成功

37

Searching in HT using Open hashing

Example : H(key) = key % 7



K=68 SL=2 查找成功

K=26 SL=3 查找不成功

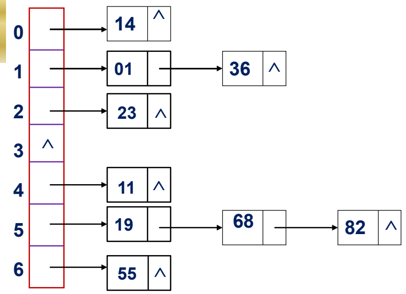
38

哈希表查找（Open hashing)的代价分析

Example ：依次查找 { 19, 01, 23, 14, 55, 68, 11, 82, 36 }

H(key) = key % 7

平均查找长度ASL:
(6*1+2*2+1*3)/9 =1.4



从查找过程得知，哈希表查找的ASL实际上并不等于零。Why?

因为有冲突存在

39

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（load factor) α=n/m 的大小（n—记录数， m—表的长度）

给定处理冲突方法，ASL是装载因子的函数。

线性探测 $ASL \approx \frac{1}{2}(1+\frac{1}{1-\alpha})$

随机探测 $ASL \approx -\frac{1}{\alpha}\ln(1-\alpha)$

链地址法 $ASL \approx 1+\frac{\alpha}{2}$

构造哈希表时，可选择一个适当的装载因子α，使得ASL限定在某个范围内。

40

9.3.5 Deletion record from HT

- Deleting a record must not hinder(影响) later searches.
- We do not want to make positions in the hash table unusable because of deletion.

- these problems can be resolved by placing a special mark in place of the deleted record, called a tombstone.
- A tombstone will not stop a search, but that slot can be used for future insertions.

tombstones will add the ASL, try the following step to shorten the ASL

1. Local reorganizations.
2. Periodically rehash the table

41

本章作业

- 9.6
- 9.14

42

Chapter9 end

43

43

课后练习

Assume that you have a 10 slots closed hash. Show the final hash table if you used the hash function h(k) = k mod 10 and pseudo-random probing on number sequence: 3, 12, 9, 2, 79, 44.

here the pseudo-random probing d_i will be: 5, 9, 2, 1, 4, 8, 6, 3, 7.

44