國立臺南大學資訊工程學系

資工三「演算法」課程

第三次作業

**題目: Dictionary using Hashing**

|  |  |  |
| --- | --- | --- |
| 班級 | ： | 資工三 |
| 姓名 | ： | 黃詩豪 |
| 學號 | ： | S10259002 |

老師：陳宗禧

中華民國 104年12月13日

# 目錄

1. **簡介及問題描述…………………………………………………………………1**
   1. **簡介………………………………………………………………………………………2**
   2. **問題……………………………………………………………………………………4**
2. **理論分析……………………………………………………………………………8**
3. **演算法則…………………………………………………………………………10**
4. **程式設計環境架構.………………………………………………………………12**
5. **程式.………………………………………………………………………………14**
6. **執行結果、討論與心得…………………………………………………………18**

參考文獻………………………………………………………………………………22

**(一) 簡介及問題描述**

1. 簡介

利用 Hash 建構一 Dictionary，並且利用該 Dictionary 做 Searching 以及其時間分析

2. 問題

利用Hash 建構一Dictionary，並且利用該Dictionary做Searching以及其時間分析。請設計與實作下列功能：

1. Build Hashing with a Text File Input test.txt
2. 定義Token & Words in Dictionary
3. Hashing (*K*=*s0s1s2…sn*)
   1. Hash function, ord(c) means ASCII code of character *c* and *q* is prime number.
   2. Hash function, where , *p* and *q* (hash table size) are prime numbers, i.e., .
4. 實作 b 兩個 Hashing，討論不同的 *p* (=11, 13, 97, 101, 149, 151, 199, 211, ...) and *q* (=211, 223, 227, 229, 233, ...)。當發生Collision時的處理方式(利用linked list)。此外，要統計Collision次數、平均Load Factor、平均Searching and Insertion 的次數。
5. Cuckoo for Hashing: Cuckoo hashing is a form of hashing that employs two hash tables *T1* and *T2*. (*T1* and *T2* table sizes 可不一樣)
   1. 先執行*h1(x)*: 若 *T1* 的位置沒有任何資料，則存*x*；若已經有存資料*y*，則將*x* 儲存在該位置，執行步驟(2)=>*h2(y)*。
   2. 再執行*h2(y)*: 若 *T2* 的位置沒有任何資料，則存*y*；若已經有存資料*z*，則將*y* 儲存在該位置，執行步驟(1)=>*h1(z)*。

統計Collision次數、平均Searching and Insertion的次數、以及無法儲存的資料。

1. Add a word (input from file or keyboard)
2. Delete a word (input from file or keyboard)
3. Search a word (input from file or keyboard)
4. Searching with an input test file, part of the Text File test.txt

**(二) 理論分析**

1. 定義Token & Words in Dictionary

在本實驗中，一個Word的定義是去除以下符號「"，[，]，(，)，-，'」的單字，但是如果有縮寫，例如：I'm，此類的則不去除符號，或是連字詞，例如：real-word，此類的也不去除符號。另外，數字也算是一個word，且以小寫表示。

2. Hash 1和Hash 2分析

Hash 1和Hash 2都是使用Open Hashing，且對於Collision發生的處方式是使用linked list掛在後面，對於Insert來說需要花O(1)的時間，Search與Delete會隨著Collision發生的次數越多且linked list越長，花費的時間會越久，所以為O(n)，n為linked list的長度。

3. Cuckoo Hash分析

Cuckoo Hash在此實驗是利用Hash 1與Hash 2做hashing，且有兩個Table儲存word，由於Cuckoo Hash是先執行*h1(x)*: 若*T1* 的位置沒有任何資料，則存*x*；若已經有存資料*y*，則將*x* 儲存在該位置，執行步驟(2)=>*h2(y)*。再執行*h2(y)*: 若*T2* 的位置沒有任何資料，則存*y*；若已經有存資料*z*，則將*y* 儲存在該位置，執行步驟(1)=>*h1(z)*。因此需要設定Cuckoo Hash的次數*k(1, 2, 3, ...)*以限制重複hash的次數，所以Insert的時間複雜度為O（k）。Search與Delete最多只需要各作一次Hash 1與Hash 2就可以找到該word，因此時間複雜度為O（1）。

**(三) 演算法則**

1. 第一個演算法(Algorithm)

**ALGORITHM** *hash1*(word[s0s1s2...sn], q)

//**Input:** word is a string, q is a prime unsigned integer

//**Output:** key value of the word is a unsigned integer

*sum* ← 0

**for** i ← 0 **to** *n* **do**

*sum* ← *sum* + *ord(si)*

**return** *(sum* **mod** *q)*

1. 演算法時間複雜度(time complexity)

累加每個字元ASCII code的十進位數值，最後再跟q取餘數，就是算出來的key值，故時間複雜度為O(n)，n為字串的字元數量

1. 演算法空間複雜度(space complexity)

程式執行過程未有動態配置的記憶體，所以空間複雜度為O(n)，其中n為字串的字元數量

2. 第二個演算法(Algorithm)

**ALGORITHM** *hash2*(word[s0s1s2...sn], q, p)

//**Input:** word is a string, q and p are the prime unsigned integer

//**Output:** key value of the word is a unsigned integer

*h* ← 0

**for** i ← n **to** *0* **do**

*h* ← *(h \* p* + *ord(si))* **mod** *q*

**return** *h*

1. 演算法時間複雜度(time complexity)

利用Hornor's rule計算，減少加法與乘法操作的次數為n次與n次，雖然時間複雜度為O(n)，但是比利用原本公式計算的方式少了很多次的加法與乘法操作次數，n為字串的字元數量

1. 演算法空間複雜度(space complexity)

程式執行過程未有動態配置的記憶體，所以空間複雜度為O(n)，其中n為字串的字元數量

3. 第三個演算法(Algorithm)

**ALGORITHM** *cuckooHash*(word[s0s1s2...sn], q, p)

//**Input:** word is a string, q and p are the prime unsigned integer

//**Output:** none

*key* ← *0*, *x ← word, y*

**for** i ← *0* **to** *5* **do**

*key* ← **hash1**(*x, q*)

**if** *Table1*[*key*] **is null**

*Table1*[*key*] ← *x*

**else**

*y* ← *Table1*[*key*]

*Table1*[*key*] ← *x*

*key* ← **hash2**(*y, q, p*)

**if** *Table2*[*key*] **is null**

*Table2*[*key*] ← *y*

**else**

*x* ← *Table2*[*key*]

*Table2*[*key*] ← *y*

**end for**

1. 演算法時間複雜度(time complexity)

利用Hash 1與Hash 2來做hashing，但限定只做5次就停止，如果都沒空的位置，則該word就放不進去，故時間複雜度為O(1)

1. 演算法空間複雜度(space complexity)

程式執行過程未有動態配置的記憶體，所以空間複雜度為O(n)，其中n為字串的字元數量

**(四) 程式設計環境架構**

程式設計語言、工具、環境與電腦硬體等規格說明…

1. 程式語言

C in Linux GCC

OS: Linux Mint 17.2 "Rafaela" – Cinnamon (64-bit)

2. 程式開發工具

Eclipse，Version: Mars Release (4.5.0)

Eclipse C/C++ Plugin: CDT 8.8.0 for Eclipse Mars

Compiler: Linux GCC

3. 電腦硬體

CPU: Intel Xeon E3-1231 v3，Main Memory: 32GB

**(五) 程式 (含source code, input code, and output code)**

程式含source code, input code, and output code等…

1. 主程式

見附件Source\_code.zip

2. Input Code Format

使用老師給的測試文章與網路上小說的文字檔

(1) TestFile1.txt，TestFile11.txt，TestFile12.txt

(2) TestFile2.txt，TestFile21.txt，TestFile22.txt

3. Output Code Format

分別輸出使用三種hash產生的Collision，Load Factor，平均Searching與Insertion次數，

還有成功率與失敗率，且q = 257，p = 11

(1)

|  |  |  |  |
| --- | --- | --- | --- |
|  | Hash 1 | Hash 2 | Cuckoo Hash |
| TestFile1.txt | Collision : 66  Load Factor : 0.54  Avg Insertion : 1.4 | Collision : 65  Load Factor : 0.54  Avg Insertion : 1.4 | Collision : 78  Avg Insertion : 1.4  Not Stored : 0 |
| TestFile11.txt | Average Searching : 1.2  Success : 1  Failure : 0  Taken Time : 0 | Average Searching : 1.3  Success : 1  Failure : 0  Taken Time : 0 | Avg Searching : 1  Success : 1  Failure : 0  Taken Time : 0 |
| TestFile12.txt | Average Searching : 0.92  Success : 0.46  Failure : 0.54  Taken Time : 0 | Average Searching : 0.9  Success : 0.46  Failure : 0.54  Taken Time : 0 | Avg Searching : 0.46  Success : 0.46  Failure : 0.54  Taken Time : 0 |

(2)

|  |  |  |  |
| --- | --- | --- | --- |
|  | Hash 1 | Hash 2 | Cuckoo Hash |
| TestFile2.txt | Collision : 357  Load Factor : 0.91  Avg Insertion : 2.1 | Collision : 358  Load Factor : 0.91  Avg Insertion : 2.1 | Collision : 4927  Avg Insertion : 9.1  Not Stored : 134 |
| TestFile21.txt | Avg Searching : 2  Success : 0.51  Failure : 0.49  Taken Time : 2.5e-05 | Avg Searching : 1.8  Success : 0.51  Failure : 0.49  Taken Time : 0 | Avg Searching : 0.47  Success : 0.47  Failure : 0.53  Taken Time : 0 |
| TestFile22.txt | Avg Searching : 1.7  Success : 0.49  Failure : 0.51  Taken Time : 1.2e-05 | Avg Searching : 1.9  Success : 0.49  Failure : 0.51  Taken Time : 0 | Avg Searching : 0.42  Success : 0.42  Failure : 0.58  Taken Time : 0 |

**(五) 執行結果、討論與心得**

執行結果與討論 (執行時間、problem *n*的大小等問題討論)等…

1. 執行結果

Output of program:

分別輸出使用三種hash產生的Collision，Load Factor，平均Searching與Insertion次數，

還有成功率與失敗率

2. 討論

執行時間、問題大小等問題討論

1. Running Time

使用TestFile2.txt與TestFile22.txt來分別討論三個hash的結果，因為測試資料量小，所以查詢時間過快，因此時間近乎0。於Hash 1與Hash 2當中，平均查詢次數皆在2.5次上下，所以查詢速度並不慢，但又比Cuckoo Hash高2次左右，因為前者處理Collision是使用linked list，所以如果Collision都發生在同一地方，會造成Hash 1與Hash 2查詢的次數增加。但是Cuckoo Hash處理Collision的方式就是不儲存該值，如果無法存的話，所以可以見到查詢的失敗率均在7成左右，因為有些字在建立字典時無法被加入字典。

Hash 1

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| q | p | collision | load factor | avg insertion | avg searching | success | failure | time (ns) |
| 211 | 11 | 432 | 0.75 | 3.4 | 3.1 | 0.49 | 0.51 | 6 |
| 223 | 13 | 432 | 0.71 | 3.2 | 3.1 | 0.49 | 0.51 | 20 |
| 227 | 97 | 416 | 0.77 | 3 | 2.7 | 0.49 | 0.51 | 0 |
| 229 | 101 | 415 | 0.77 | 2.8 | 2.8 | 0.49 | 0.51 | 0 |
| 233 | 149 | 394 | 0.85 | 2.5 | 2.4 | 0.49 | 0.51 | 0 |

Hash 2

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| q | p | collision | load factor | avg insertion | avg searching | success | failure | time (ns) |
| 211 | 11 | 391 | 0.95 | 2.3 | 2.3 | 0.49 | 0.51 | 9 |
| 223 | 13 | 383 | 0.93 | 2.4 | 2.3 | 0.49 | 0.51 | 17 |
| 227 | 97 | 384 | 0.91 | 2.3 | 1.9 | 0.49 | 0.51 | 0 |
| 229 | 101 | 374 | 0.95 | 2.3 | 2.2 | 0.49 | 0.51 | 0 |
| 233 | 149 | 380 | 0.91 | 2.3 | 2.1 | 0.49 | 0.51 | 0 |

Cuckoo Hash

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| q | p | collision | avg insertion | not insertion | avg searching | success | failure | time (ns) |
| 211 | 11 | 7771 | 14 | 233 | 0.25 | 0.25 | 0.75 | 6 |
| 223 | 13 | 7839 | 14 | 229 | 0.22 | 0.22 | 0.75 | 0 |
| 227 | 97 | 7242 | 13 | 212 | 0.26 | 0.26 | 0.74 | 0 |
| 229 | 101 | 6841 | 12 | 204 | 0.3 | 0.3 | 0.7 | 0 |
| 233 | 149 | 6526 | 12 | 186 | 0.3 | 0.3 | 0.7 | 0 |

(2) Problem size *n*

由(1)可見，當Table Size (q) 越大的時候平均搜尋次數會減少，加快查找的時間，但是Table 1的大小也就變大。但是Cuckoo Hash卻並不是這樣，因為Cuckoo Hash需要找兩個Table，因此每次查詢最多需要兩次，如果欲查找的字都在Table 2的話，那平均查詢次數就會增加。

3. 心得

這次的作業當中，發現Hash Function在實作上非常的輕鬆，而且查詢的速度也非常快，能在大量資料中，快速地找到資料，但是所付出的代價就是大量的空間，可是在現在的環境中，記憶體的價格越趨便宜且容量越大，所以在追求效率的情況下，使用Hash或許是一個不錯的方法。

**參考文獻**

1. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein, "Introduction to Algorithms," Third Edition, The MIT Press, 2009.
2. R.C.T. Lee, S.S. Tseng, R.C. Chang, and Y.T.Tsai, "Introduction to the Design and Analysis of Algorithms," McGraw-Hill, 2005.
3. Anany V. Levitin, "Introduction to the Design and Analysis of Algorithms," 2nd Edition, Addison Wesley, 2007.
4. Richard Neapolitan and Kumarss Naimipour, "Foundations of Algorithms," Fourth Edition, Jones and Bartlett Publishers, 2010.