Bloom filter 偽陽性實驗報告

1.前言：

Bloom filter 由 m 個位元的內存和 h 個具均勻分布性且獨立的 hash function 所組成，每一個 hash function 可將一個 key 分佈到 [1,m] 這個範圍內的一個整數 。起初這 m 個位置的內存都會被設為零，透過將 key 放入 h 個 hash function 中會將內存當中 h 個位置被設為 1，使 bloom filter 中標記有這個元素的存在，透過這個方法能讓我們用**很小的空間存大量的資料**。

當我們要搜尋一個元素是否存在時就只要將這個元素透過 h 個 hash function 的轉換，去檢查這些位置的內存是否「都有」被標為 1，我們就能以**時間複雜度 O(1)**的速度在大量的資料中查找。

在 Bloom filter 的查找中是會有**偽陽性(False positive)**存在的，當我們將要查找的資料放入 Bloom filter 的時候會遇到兩個狀況：

- 資料經過 h 個 hash function 後去跟 Bloom filter **比較只要有一個位元是零，代表資料一定不在 Bloom filter 當中**。
- 資料經過 h 個 hash function 後去跟 Bloom filter **比較在相應的位元中都是 1 的話，代表資料有可能在 Bloom filter 當中，並不是一定有，如果真的不存在就是偽陽。**

所以在使用 Bloom filter 時我們必須**控制偽陽性的機率不能過高**，否則這個資料結構會變的相當無用。

2.實驗目的及原理：

目的：透過多次查找 Bloom filter 當中的元素的計算在不同 hash function 個數下的偽陽性的機率並證明出在不同的元素和位元比例下會有不同的最佳 hash function 個數使得偽陽性機率最低。

原理：

一個 Bloom filter 具有以下參數：

- n 個元素
- k 個 hash function
- m 個位元

透過 Bloom filter 的偽陽性公式可推導

$$f = (1 - \left[1 - \frac{1}{m}\right]^{kn})^k \approx (1 - e^{\frac{-kn}{m}})^k \text{-----(1)}$$

將$e^{\frac{-kn}{m}}$以 p 做取代

$$f = (1 - p)^k = e^g \text{ -----(2)}$$

$$g = kln(1 - p)\text{-----(3)}$$

我們要計算出偽陽性的最小值可以透過將公式(3)這個凹函數作微分找出等於零的點即為最小值。

3.實驗設計：

實驗主要就透過實際操作 Bloom filter 對其 insert 多個元素過後，再進行查詢的動作，每當查詢時 Bloom filter 告述我們此元素存在於其中時就去跟原始的 insert 資料去做比對如果跟原始資料比對有出入代表就是偽陽的情況產生，將上述動作做多次後取平均來計算偽陽性。

程式碼範例：

以下程式碼為將測試的資料透過 hash function 放入 Bloom filter 當中

```
unsigned  hashfunction_1(unsigned data)
{
    data ^= (data << 8);
    data ^= (data >> 7);
    data ^= (data << 5);
    unsigned index = data % SIZE;
    bitvector[index] = true;
    return index;
}
```

再來的部分為**要查找的資料**也透過 hash function 得到相應在 Bloom filter 當中的位置

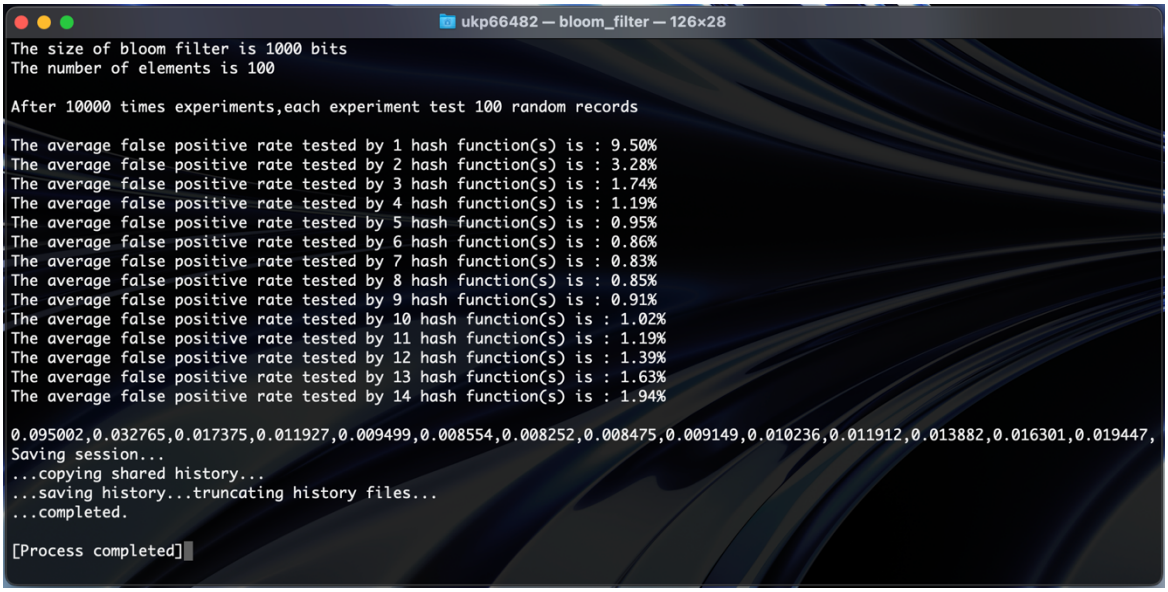
```
unsigned hashfunction_1_transform(unsigned data)
{
    data ^= (data << 8);
    data ^= (data >> 7);
    data ^= (data << 5);
    unsigned index = data % SIZE;
    return index;
}
```

最後就是透過檢查要查找的資料在 Bloom filter 當中有沒有都被標為 1，如果有就要去查找原始資料當中有沒有我們現在查找的這筆資料，如果沒有這筆資料就代表偽陽的發生。

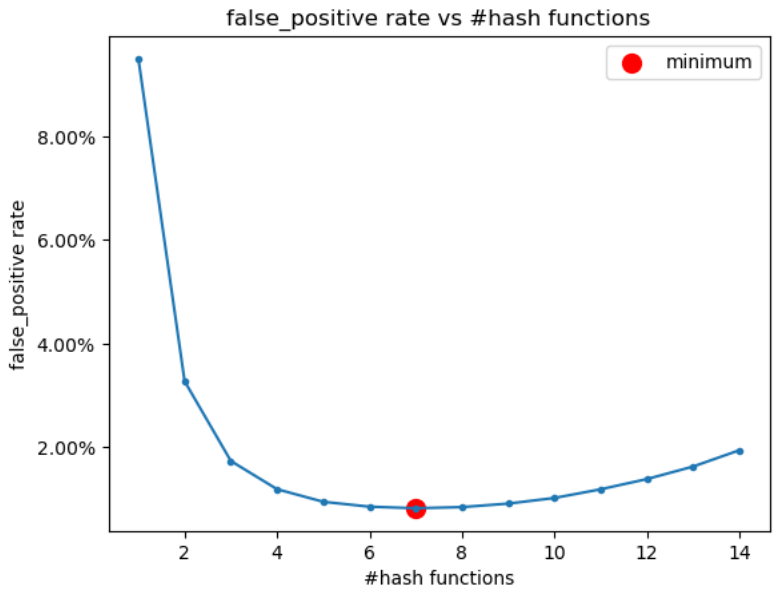
```
1.  void  find_element_1(unsigned index,unsigned data)
2.  {
3.      bool find = false;
4.      if(bitvector[index] == true) {
5.          for (int i = 0; i < ELEMENT; i++) {
6.              if (element[i] == data) {
7.                  find = true;
8.                  break;
9.              }
10.         }
11.         if(find == false) {
12.             false_positive[1]++;
13.         }
14.     }
15. }
```

4.實驗數據及結果：

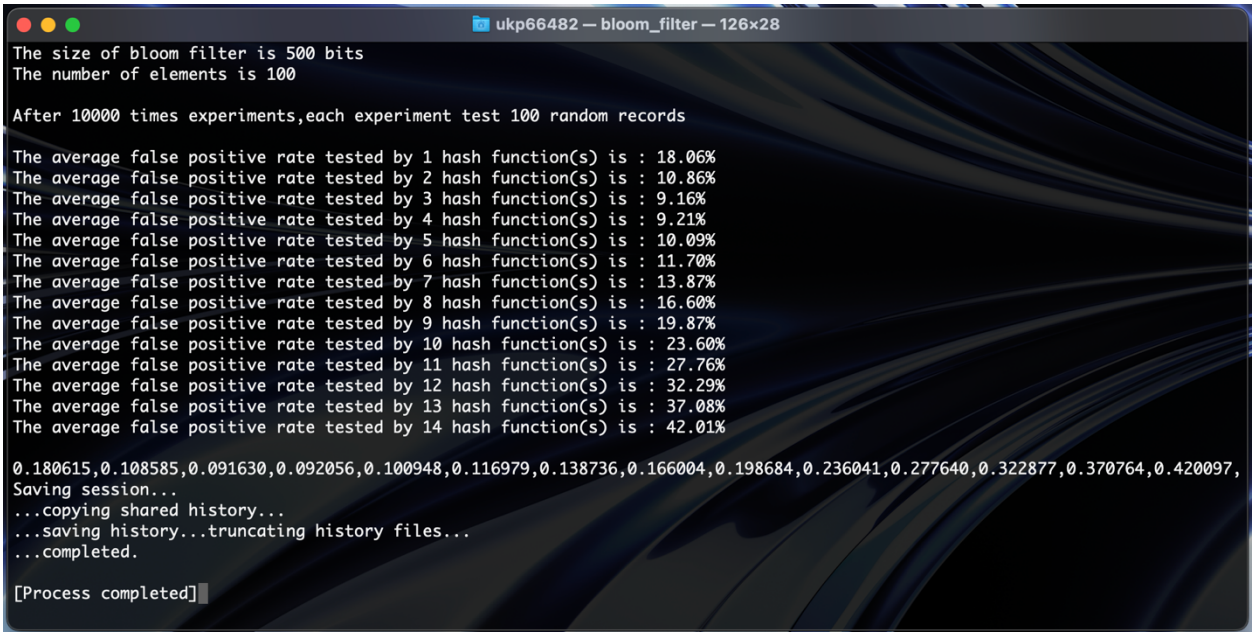
當我們在 Bloom filter 中配置 **1000 個位元 100 個元素**在其中，並透過 100 筆隨機的資料做查詢，此動作我們連續做 10000 次減少誤差，**偽陽性的機率最小值發生在 hash function 個數為 7**。



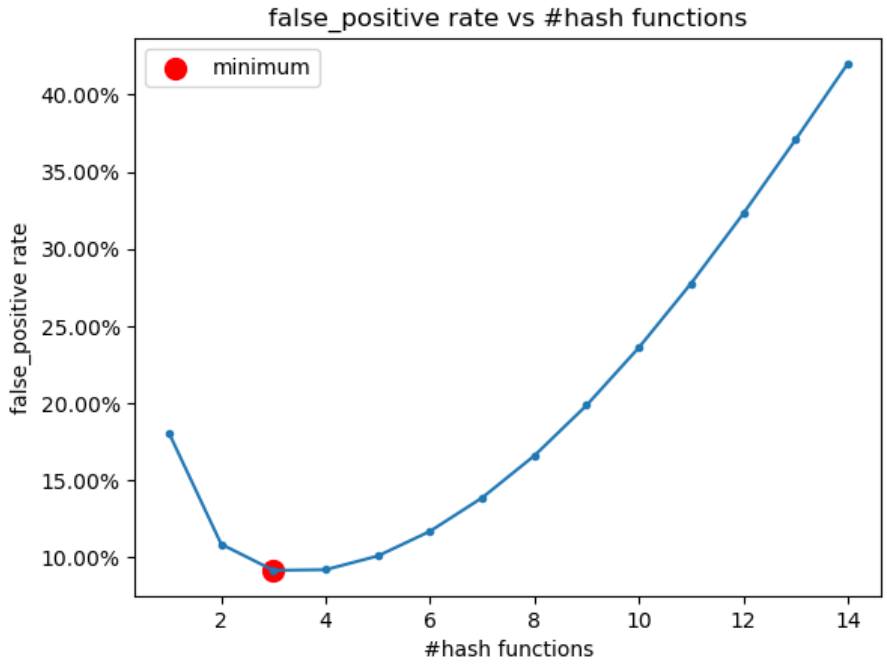
將所有 hash function 個數的偽陽性機率透過圖表表示，可發現圖形如公式(3)的描述為凹向上的圖形在 **hash function 個數為 7 時有偽陽性的最小值**。



當我們在 Bloom filter 中配置 **500 個位元 100 個元素**在其中，並透過 100 筆隨機的資料做查詢，此動作我們連續做 10000 次減少誤差，**偽陽性的機率最小值發生在 hash function 個數為 3**。

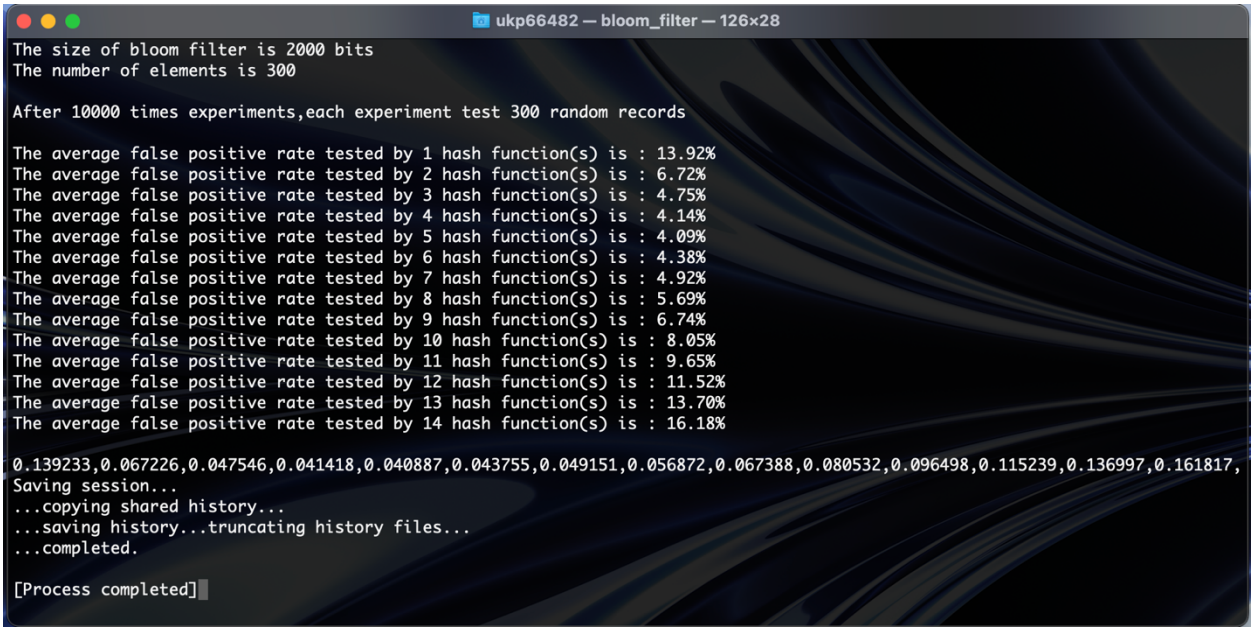


將所有 hash function 個數的偽陽性機率透過圖表表示，可發現圖形如公式(3)的描述為凹向上的圖形在 **hash function 個數為 3 時有偽陽性的最小值**。



當我們在 Bloom filter 中配置 **2000 個位元 300 個元素**在其中，並透過 300 筆隨機的資料做查詢，

此動作我們連續做 10000 次減少誤差，**偽陽性的機率最小值發生在 hash function 個數為 5**。



將所有 hash function 個數的偽陽性機率透過圖表表示，可發現圖形如公式(3)的描述為凹向上

的圖形在 **hash function 個數為 5 時有偽陽性的最小值**。