# Ch.8 Hashing 習題

# Q1

- Consider a hash table with 26 buckets and 3 slots. Assume that there are 12 distinct keys and each key begins with a capital letter. If we have the hash function h(k) map each key into a bucket using its leading letter and correspond the letters A to Z to the numbers 0 to 25. Please answer the followings questions :

  a. Compute the loading factor of this table and explain what is loading factor.

  b. Given the keys D, B, GH, GG, B2, C3, C2, C1, A2, B7, C6, GA, please convert them into home buckets and complete the hash table. In addition, you should indicate which keys are synonyms, and which keys cause overflow.

# Q1_Ans (a)

a.參考講義 8.2

- We defined loading factor α:
  - $\alpha = \frac{n}{s*b} = \frac{The\ numbers\ of\ dictionary\ pairs\ in\ a\ hash\ \ table}{The\ total\ numbers\ of\ possible\ pairs\ in\ a\ hash\ table}$
  - n = 12 (distinct keys or dictionary pairs)
  - b = 26 (buckets)
  - s = 3 (slots, size per bucket)
- Ans: $\alpha = \frac{12}{3*26} = \frac{2}{13} \fallingdotseq 0.154$

# Q1_Ans (b)

b.

| Bucket, h(k) | Slot1 | Slot2 | Slot3 | Overflow |
|---|---|---|---|---|
| 0 | A2 | | | |
| 1 | B | B2 | B7 | |
| 2 | C3 | C2 | C1 | C6 |
| 3 | D | | | |
| 4 | | | | |
| 5 | | | | |
| 6 | G | GG | GA | |
| 7 | | | | |
| ... | | | | |
| 26 | | | | |

Ans:

Synonyms: $h(k_1) = h(k_2)$

- B, B2, B7
- C3, C2, C1, C6
- G, GG, GA

Overflow:

(Bucket3 have no empty slots)

- C6

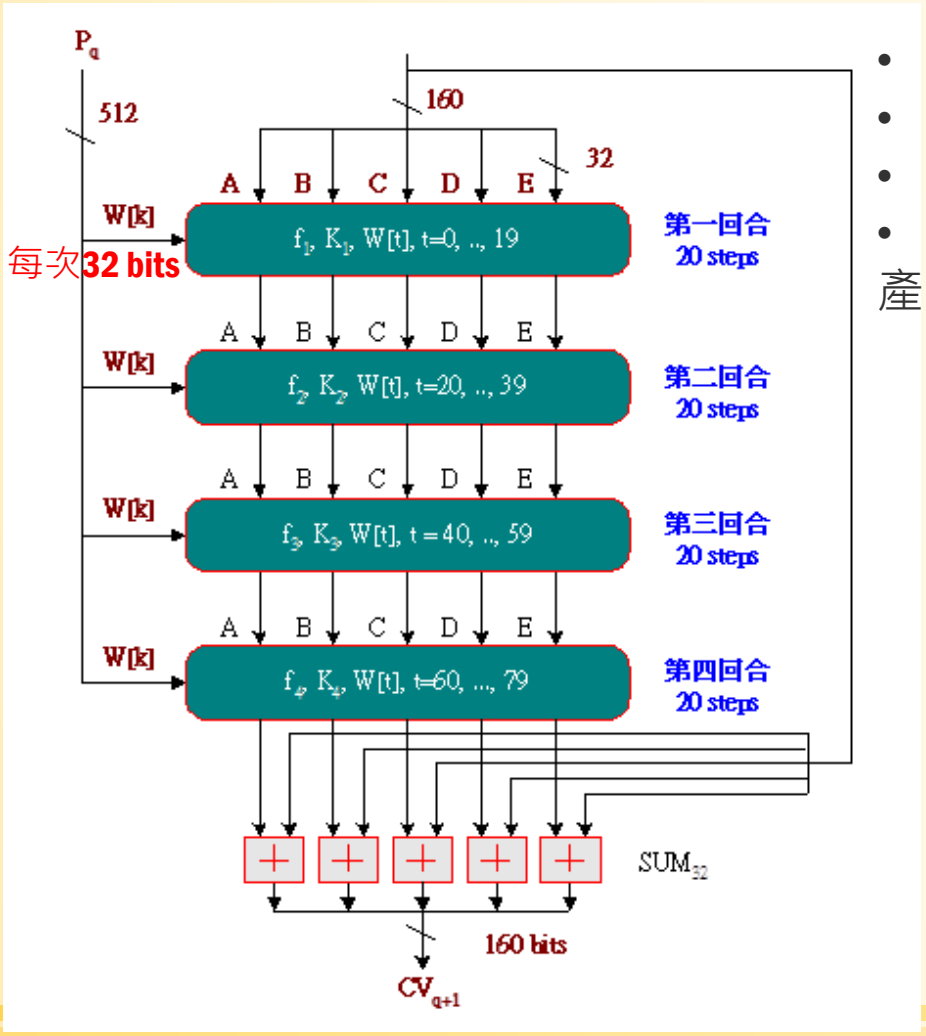
# Q2

- Please propose a hash application. TAs provide the hash application in cryptography as below. Your answer should have the same integrity as TAs' example.

# Q2_Ans (1)

- **Secure Hash Algorithm (SHA)**，這邊舉例**SHA-1**：
  - 『安全雜湊演算法』（**Secure Hash Algorithm, SHA**）是美國 **NIST** 所制定的標準，於 **1993** 年與 **1995** 年分別發表 **FIPS PUB 180** 與 **FIPS PUB 180-1**，目前都通稱後面的版本為 **SHA-1**。
  - 網路安全相關課程，會介紹**SHA**系列跟**MD**系列相關的知識及應用。
  - **SHA** 是以 **MD4** 為基礎發展出來的，其設計方式與 **MD5** 非常相似。首先我們列出 **SHA-1** 的特性，再來推演它的演算法，特性如下：
    - 可以輸入不定長度的訊息，但不可以超過 $2^{64}$ 個位元。
    - 附加位元後的訊息，以 **512** 位元為單位。
    - 每個步驟計算與最後運算的結果，皆得到 **160** 個位元的雜湊值。

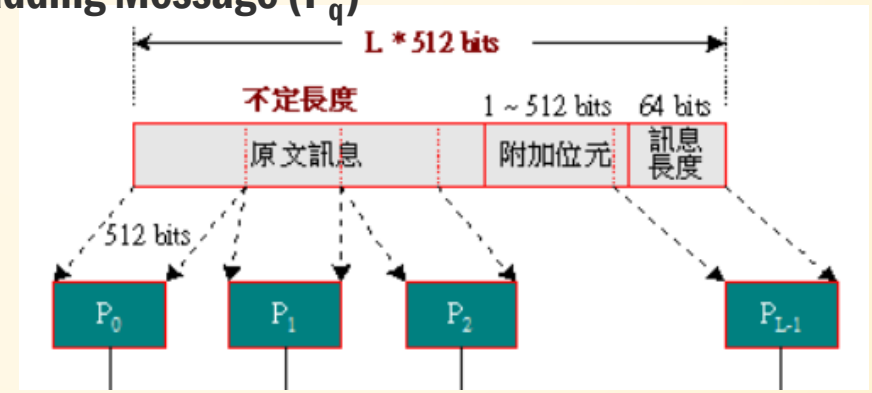
# Q2_Ans (2)




- $P_q$ 傳遞訊息(將**m bits**的**message**透過**Padding**，並使得每一輸出區塊為**512bits**)
- **h(k)** 為**160 bits**，會切分成**5**個**32bits register**做運算
- 運算分成 **4 rounds**，每一**round**進行**20**次加法、**XOR**、**shift**運算
- 總共重複做**80**次，最後再打亂位置排序利用**logical 32bits SUM**

產生**160 bits hash value**。

**Padding Message ($P_q$)**



# Q2_Ans (3)

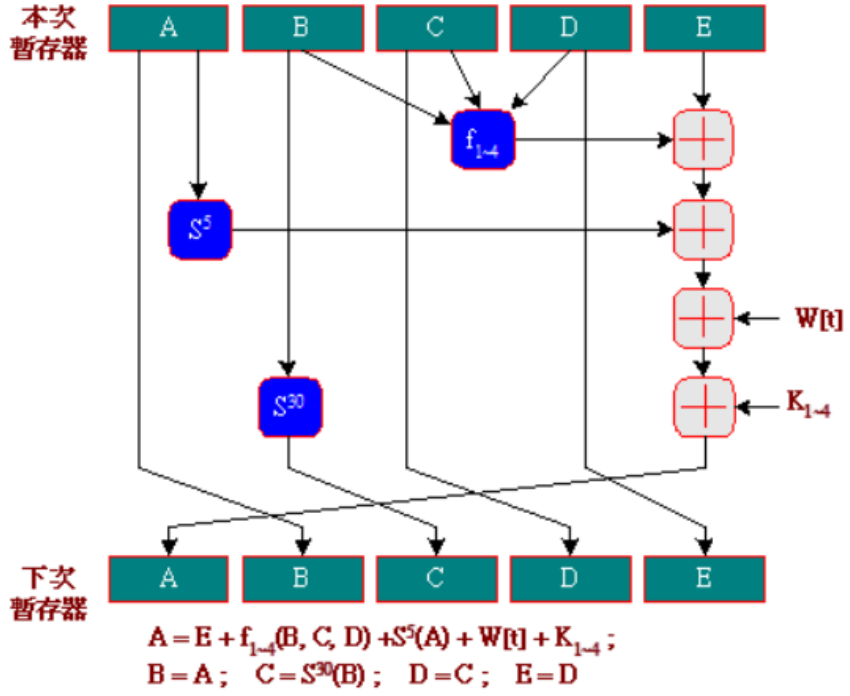
$A = E + f_{1\text{-}4}(B, C, D) + S^5(A) + W[t] + K_{1\text{-}4}$

$B = A$

$C = S^{30}(B)$

$D = C$

$E = D$




- 這個步驟我們稱作**Compression**。
- **$f_{1\text{~}4}$**為**steps**執行到一定程度時，會換不同的常數代入運算，及不同的邏輯運算。**(Steps**總共**80**個，每**20**個**steps**換一個輸入常數**)**。
- **$S^n$** 代表對該**32bits register**做向左**n**個**shift**，且**register**為一個**circular**，所以被擠出去的**n**個**bits**，會由**register**右側放入。
- 最後運算的結果，會打亂進行擺放，給下一次運算時，輸入的**register**進行使用。

總結：通過大量常數加法、**XOR**、**shift**以及位置擺放打亂的操作以後，我們可以獲得一個足夠隨機的**hash value**。

**$f_{1\text{~}4}$** 邏輯運算

| 回合 | 步驟編號 | 基本邏輯函數 | 簡式 |
|---|---|---|---|
| 1 | $0 \leqq t \leqq 19$ | $f_1 = (B \wedge C) \vee (\overline{B} \wedge D)$ | BC+$\overline{B}$ D |
| 2 | $20 \leqq t \leqq 39$ | $f_2 = B \oplus C \oplus D$ | B⊕C⊕D |
| 3 | $40 \leqq t \leqq 59$ | $f_3 = (B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$ | BC+BD+CD |
| 4 | $60 \leqq t \leqq 79$ | $f_4 = B \oplus C \oplus D$ | B⊕C⊕D |

# Q2_Ans (Appendix_1)

## 4-4-1 暫存器起始值

共計有 5 個 32 位元的暫存器，還未輸入計算之前，給予的起始值如下：（以 16 進位表示）

A：67 45 23 01
B：EF CD AB 89
C：98 BA DC FE
D：10 32 54 76
E ：C3 D2 E1 F0

前面 4 個暫存器與 MD5 相同，但 SHA-1 是採用『較高位元結尾』（Big-endian）的格式儲存（不同於 MD5）。

# Q2_Ans (Appendix_2)

## 4-4-2 輸入常數

每一運算回合都會分別給一個固定常數（**類似『鹽』的功能**），四個回合共有四個常數（$K_{1\sim4}$）；如以步驟來計算，共有 80 個步驟（t），每個步驟的常數如下：（以 16 進位表示）

| 回合 | 步驟編號 | 輸入常數 | 取值方式 (整數) |
|---|---|---|---|
| 第一回合 | $0 \leqq t \leqq 19$ | $K_1$ = 5A82799 | $[2^{30} \times 2]$ |
| 第二回合 | $20 \leqq t \leqq 39$ | $K_2$ = 6ED9EBA1 | $[2^{30} \times 3]$ |
| 第三回合 | $40 \leqq t \leqq 59$ | $K_3$ = 8F1BBCDC | $[2^{30} \times 5]$ |
| 第四回合 | $60 \leqq t \leqq 79$ | $K_4$ = CA62C1D6 | $[2^{30} \times 10]$ |

## 4-4-3 訊息擴充

為了打亂訊息內資料的關聯性、或訊息的格式，SHA-1 採用了訊息擴充的方法，將訊息（512 個位元）擴充成 80 個 32 位元的小區塊（W[k], k=0, 1, …, 79）。其方法是，首先將訊息以每 32 位元為單位，分割為 16 個小區塊，分別存入 W[0]、W[1]、…、W[15] 之中，而第 16 個以後的小區塊，分別以下面公式計算填入：

$$W[t] = S^1(W[t-16] \oplus W[t-14] \oplus W[t-8] \oplus W[t-3]),\ t = 16, 17, \ldots, 79$$

譬如：

$$W[16] = S^1(W[0] \oplus W[2] \oplus W[8] \oplus W[13])$$
$$W[17] = S^1(W[1] \oplus W[3] \oplus W[9] \oplus W[14])$$
$$\ldots\ldots,$$
$$W[79] = S^1(W[63] \oplus W[65] \oplus W[71] \oplus W[76])$$

# Q3 (a)

- When implementing a hash table, one of the commonly used collision resolution methods is open addressing. Assume that k is the key, m is the size of the hash table, and each table entry can hold one key.

  a. The simplest open addressing method is called linear probing, where if a slot is already occupied, the following slots are "probed" sequentially. Assume an auxiliary hash function $h'(k)=k \bmod m$. Please give the hash function $h(k,i)$ of linear probing with the given auxiliary function, where $i$ is the number of times the hash table has been probed with this key (and thus $i$ always starts from 0).

# Q3 (b)

b. Linear probing is easy to implement but suffers from the problem of primary clustering. A more complex open addressing method which can mitigate this problem is double hashing, using a hash function in the form of $h(k,i)=(h_1(k)+ih_2(k))$ mod $m$, with two auxiliary hash functions $h_1(k)$ and $h_2(k)$. The number $i$ is the number of times the hash table has been probed with the key $k$. Assume two auxiliary hash functions $h_1(k)=k$ mod 16, $h_2(k)=1+(k \bmod 15)$. Show the final hash table content after inserting all of the following keys to an initially empty hash table in the given order: {16,3,35,67,51,1,15,31,19,17}.

# Q3 (c)

c. Prof. Alpha proposes a different double hashing method, using two auxiliary hash functions $h1(k)=k \bmod 16$ and $h2(k)=2(k \bmod 8)$. Assume the table size m is 16. Please give an example to illustrate why Prof. Alpha's method is problematic.

# Q3 (d)

**In addition to open addressing, there is another kind called chaining, please explain chaining and implement Hash Table with Chaining List.**

# Q3_Ans (a)

a. h(k,i)=(h’(k)+i) mod m

- h’(k)=k mod m 代入，得h(k) = ((k mod m)+i) mod m
- If i is zero, then it maps to h(k) = k mod m, and this number must in the range of 0 to m-1.
- However, if h(k, 0) maps to a nonempty entry, we calculate h(k, 1) = ((k mod m)+1) mod m as an alternative, this hash function maps to next entry of the previous entry.
- If there is still nonempty in this entry, then we find h(k, 2) = h((k mod m)+2). By doing “previous i + 1”, we can iteratively find an empty entry, if there is at least one empty entry in the hash table.

# Q3_Ans (b_1)

b. h(k, i) = ((k mod 16)+i*(1+(k mod 15))) mod m

- Initial key set: {16,3,35,67,51,1,15,31,19,17}
- h(16, 0) = (16%16) % 16 = 0
- h(3, 0) = (3%16)%16 = 3
- h(35, 0) = (35%16)%16 = 3 (collision) => h(35,1) = (35%16+(1+(35%15)))%16= 9
- h(67, 0) = (67%16)%16 = 3 (collision) => h(67,1) = (67%16+(1+(67%15)))%16=11
- h(51, 0) = (51%16)%16 = 3 (collision) => h(51, 1) = (51%16+(1+(51%15)))%16=10
- h(1, 0) = (1%16)%16 = 1
- h(15, 0) = (15%16) %16 = 15

# Q3_Ans (b_2)

b. (continue)

- h(31, 0) = (31%16) %16 =15 (collision)

  =>h(31, 1) = (31%16+1*(1+(31%15)))%16=1 (collision)

  => h(31,2)=(31%16+2*(1+(31%15)))%16=3 (collision)

  => h(31, 3) = (31%16+3*(1+(31%15)))%16=5
- h(19, 0) = 19%16 = 3 (collision) => h(19, 1) = (19%16+1*(1+(19%15)))%16 = 8
- h(17, 0) = 17%16 = 1 (collision) => h(17, 1) = (17%16+1*(1+(17%15)))%16= 4

# Q3_Ans (b_3)

- b. (continue)

Hash Table

| | |
|---|---|
| 0 | 16 |
| 1 | 1 |
| 2 | |
| 3 | 3 |
| 4 | 17 |
| 5 | 31 |
| 6 | |
| 7 | |
| 8 | 19 |
| 9 | 35 |
| 10 | 51 |
| 11 | 67 |
| 12 | |
| 13 | |
| 14 | |
| 15 | 15 |

# Q3_Ans (c)

c. h(k, i) = (k mod 16 + i * 2*(k mod 8)) mod m, set m=16

- If the key A has the multiple of 16, we set k = 16A, where A ∈ N
- h(16A, 0) = (16%16+0*2*(16A%8))%16=0 ， no matter how we change the value of i.
- So, the collision always occurs and we cannot fix it by using this hash function.

# Q3_Ans (d)

**Please refer to:** https://www.geeksforgeeks.org/c-program-hashing-chaining/

# Q4

- Consider a hash function h(k)=k%D, where D is to be defined. We wish to find a proper D value using minimal number of attempts, while each attempt requires supplying the function with k and observing the result of h(k). Indicate how this may be achieved in the following two cases.
  a. D is known to be a prime number in the range [10, 20].
  b. D is of the form $2^k$, where k is an integer in [1, 5].

# Q4_Ans

**a. The prime between [10, 20] => D may be 11, 13, 17 and 19.**

- 我們假設**k=20**，並觀察**h(k)=k % D**可以發現，分別會得到**9, 7, 3, 1**，不同的**D**會有不同的**h(k)**結果，所以可以利用 **h(20)** 一次推導出**D**為哪一個質數。

**b. D is of the form $2^k$, when k is an integer in [1, 5] => D may be 2, 4, 8, 16, 32.**

- 我們假設**k=31**，並觀察**h(k)=k%D**，會發現可能的結果是**1, 3, 7, 15, 31**，每一個出來的結果都不一樣，因此我們可以藉由 **h(31)** 推導 **D** 是哪一個**2**的**k**次方項。

# Q5

- Suppose that you are to design a Bloom filter with minimum P(u) with n=100000, m=5000, and u=1000.

  a. Use any of the results obtained in the textbook, and determine the number of hash functions required. Show your computations.

  b. What is the probability, P(u), of a filter error with the required number of hash functions computed above?

# Q5_Ans

**a.** 跟 **b.** 一起解釋

- 已知 **n=100000, m=5000, and u=1000.**
- 由講義**8.4.2** 對**Bloom filter design**的介紹，為了最小化**Bloom filter**失誤機率，我們對**P(u)**進行微分，並設定其結果為**0 (**找極值**)**可以得到：
  - **$h = (\log_e 2)*(m/u) \sim 0.693*(m/u) = 3.465$**
  - 取其**upper bound**及**lower bound** 獲得**h = 3 or 4**
- 我們再將 **h** 代入 **P(u)** 的公式中，算出不同**hash function**下的失誤機率，可得：
  - **$h=3, P(u) \sim e^{-u/n}*(1-e^{-uh/m})^h \sim 0.0909$**
  - **$h=4, P(u) \sim e^{-u/n}*(1-e^{-uh/m})^h \sim 0.091$**
  - 計算後發現，最小**P(u)**發生於**h=3**，並且失誤機率為**0.0909 (9.09%)**

# Q6

- A hash table of length 10 uses open addressing with the hash function h(k)=k mod 10, and linear probing. After inserting 6 values into an empty hash table, the table is as shown below.
- Which one of the following choices gives a possible order in which the key values could have been inserted in the table?
  (A) 46, 42, 34, 52, 23, 33
  (B) 34, 42, 23, 52, 33, 46
  (C) 46, 34, 42, 23, 52, 33
  (D) 42, 46, 33, 23, 34, 52

| | |
|---|---|
| 0 | |
| 1 | |
| 2 | 42 |
| 3 | 23 |
| 4 | 34 |
| 5 | 52 |
| 6 | 46 |
| 7 | 33 |
| 8 | |
| 9 | |

# Q6_Ans

(A)

| | |
|---|---|
| 0 | |
| 1 | |
| 2 | 42 |
| 3 | 52 |
| 4 | 34 |
| 5 | 23 |
| 6 | 46 |
| 7 | 33 |
| 8 | |
| 9 | |

It doesn't create the hash table as the element 52 appears before 23

(B)

| | |
|---|---|
| 0 | |
| 1 | |
| 2 | 42 |
| 3 | 23 |
| 4 | 34 |
| 5 | 52 |
| 6 | 33 |
| 7 | 46 |
| 8 | |
| 9 | |

It doesn't create the hash table as the element 33 appears before 46

# Q6_Ans

(C)

| | |
|---|---|
| 0 | |
| 1 | |
| 2 | 42 |
| 3 | 23 |
| 4 | 34 |
| 5 | 52 |
| 6 | 46 |
| 7 | 33 |
| 8 | |
| 9 | |

Ans: (C)

(D)

| | |
|---|---|
| 0 | |
| 1 | |
| 2 | 42 |
| 3 | 33 |
| 4 | 23 |
| 5 | 34 |
| 6 | 46 |
| 7 | 52 |
| 8 | |
| 9 | |

It doesn't create the hash table as the element 23 appears before 34

# Q7

- Which one of the following hash functions on integers will distribute keys most uniformly over 10 buckets numbered 0 to 9 for i ranging from 0 to 2020?

  (A) h(i) =i^2 mod 10
  (B) h(i) =i^3 mod 10
  (C) h(i) = (11 * i^2) mod 10
  (D) h(i) = (12 * i) mod 10

https://www.geeksforgeeks.org/gate-gate-cs-2015-set-2-question-43/

# Q7_Ans (A)

- (A) Since mod 10 is used, the last digit matters. If you do square all numbers from 0 to 9, you get following:

| Number | Square | Last Digit |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 1 | 1 |
| 2 | 4 | 4 |
| 3 | 9 | 9 |
| 4 | 16 | 6 |
| 5 | 25 | 5 |
| 6 | 36 | 6 |
| 7 | 49 | 9 |
| 8 | 64 | 4 |
| 9 | 81 | 1 |

# Q7_Ans (B)

- (B) Since mod 10 is used, the last digit matters. If you do cube all numbers from 0 to 9, you get following:

| Number | Cube | Last Digit |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 1 | 1 |
| 2 | 8 | 8 |
| 3 | 27 | 7 |
| 4 | 64 | 4 |
| 5 | 125 | 5 |
| 6 | 216 | 6 |
| 7 | 343 | 3 |
| 8 | 512 | 2 |
| 9 | 729 | 9 |

# Q7_Ans (C)

- (C) Since mod 10 is used, the last digit matters. If you do 11*square all numbers from 0 to 9, you get following:

| Number | Square | Last Digit |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 11 | 1 |
| 2 | 44 | 4 |
| 3 | 99 | 9 |
| 4 | 176 | 6 |
| 5 | 275 | 5 |
| 6 | 396 | 6 |
| 7 | 539 | 9 |
| 8 | 704 | 4 |
| 9 | 891 | 1 |

# Q7_Ans (D)

- (D) Since mod 10 is used, the last digit matters. If you do 12*I all numbers from 0 to 9, you get following:

| Number | Square | Last Digit |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 12 | 2 |
| 2 | 24 | 4 |
| 3 | 36 | 6 |
| 4 | 48 | 8 |
| 5 | 60 | 0 |
| 6 | 72 | 2 |
| 7 | 84 | 4 |
| 8 | 96 | 6 |
| 9 | 108 | 8 |

# Q7_Ans Summary

- **(A), (B), (C) and (D)** 從 **i= 0~9**，我們就可以觀察到 **(A), (C) and (D)**都有很嚴重的**collision**狀況，而**(B)**則相對比較均衡每一個值都在一個**bucket**裡面。
  - **Ans: (B)**
- 如果要更嚴謹的證明方法，就要寫程式跑 **i=0~2020 mod 10**之後，並算使用**bucket**的分布，並算出標準差，而擁有標準差最小的代表越均勻，也就是我們的**(B)**。

# Q8

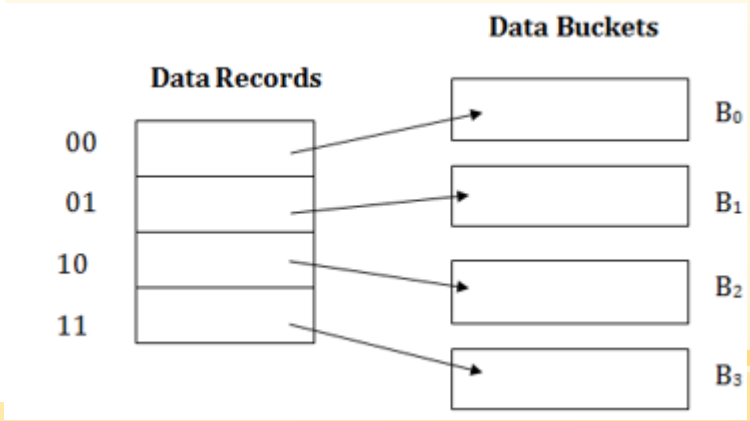
- The dynamic hashing is used to overcome the problem of static hashing like bucket overflow without resulting in poor performance. Please answer the questions below.

  (a) Please discuss the strengths and weaknesses of dynamic hashing.

  (b) Consider the following grouping of keys mapped into buckets, depending on the prefix of their hash address. Please fill up the data buckets.

  (C) If we need to insert key 9 with hash address 10001 into the above structure, what is the result of the insertion. (Please draw the similar structure as the second figure)

| Key | Hash address |
|---|---|
| 1 | 11010 |
| 2 | 00000 |
| 3 | 11110 |
| 4 | 00000 |
| 5 | 01001 |
| 6 | 10101 |
| 7 | 10111 |



# Q8_Ans (a)

(a)

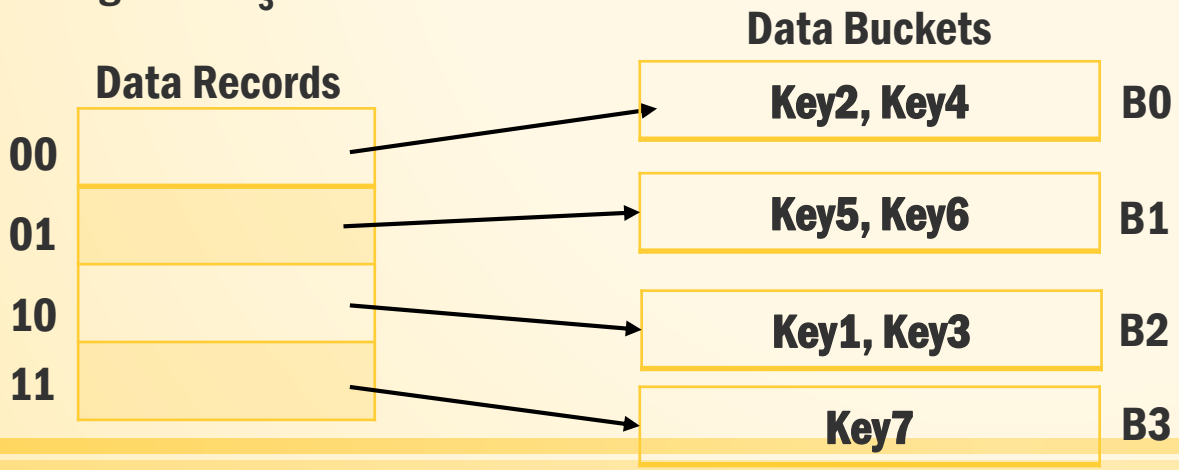
- Advantages of dynamic hashing:
  - In this method, the performance does not decrease as the data grows in the system. It simply increases the size of memory to accommodate the data.
  - In this method, memory is well utilized as it grows and shrinks with the data. There will not be any unused memory lying.
  - This method is good for the dynamic database where data grows and shrinks frequently.
- Disadvantages of dynamic hashing:
  - In this method, if the data size increases then the bucket size is also increased. These addresses of data will be maintained in the bucket address table. This is because the data address will keep changing as buckets grow and shrink. If there is a huge increase in data, maintaining the bucket address table becomes tedious.
  - In this case, the bucket overflow situation will also occur. But it might take little time to reach this situation than static hashing.

# Q8_Ans (b)

- (b)

| Key | Hash address |
|---|---|
| 1 | 11010 |
| 2 | 00000 |
| 3 | 11110 |
| 4 | 00000 |
| 5 | 01001 |
| 6 | 10101 |
| 7 | 10111 |

The last two bits of 2 and 4 are 00. So it will go into bucket $B_0$. The last two bits of 5 and 6 are 01, so it will go into bucket $B_1$. The last two bits of 1 and 3 are 10, so it will go into bucket $B_2$. The last two bits of 7 are 11, so it will go into $B_3$.

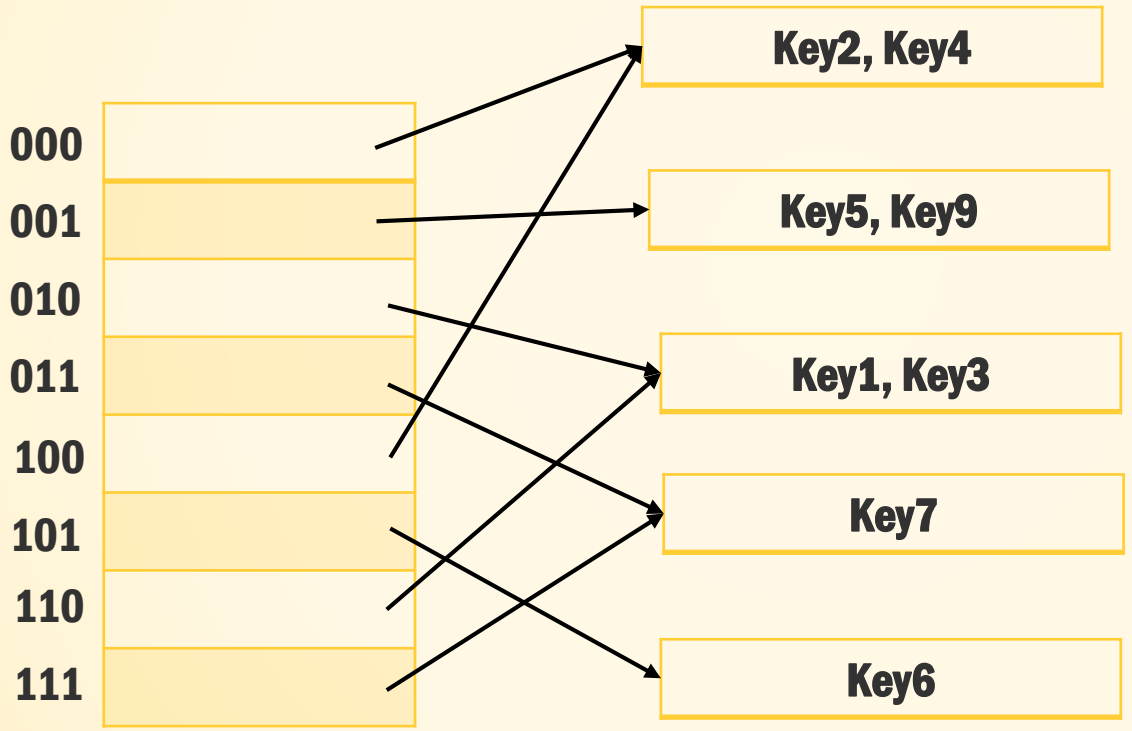


# Q8_Ans (c_1)

- (c) Insert key 9 with hash address 10001 into the above structure
  - Since key 9 has hash address 10001, it must go into the first bucket. But bucket B1 is full, so it will get split.
  - The splitting will separate 5, 9 from 6 since last three bits of 5, 9 are 001, so it will go into bucket B1, and the last three bits of 6 are 101, so it will go into bucket B5.
  - Keys 2 and 4 are still in B0. The record in B0 pointed by the 000 and 100 entry because last two bits of both the entry are 00.
  - Keys 1 and 3 are still in B2. The record in B2 pointed by the 010 and 110 entry because last two bits of both the entry are 10.
  - Key 7 are still in B3. The record in B3 pointed by the 111 and 011 entry because last two bits of both the entry are 11.

# Q8_Ans (c_2)

- Ans:


000
001
010
011
100
101
110
111
Key2, Key4
Key5, Key9
Key1, Key3
Key7
Key6