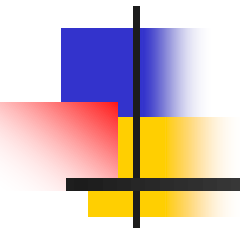


Chapter 08

Hashing

Hashing

- สร้างตารางสำหรับเก็บข้อมูลโดยกำหนดคีย์ที่ต้องการค้นเป็น **index** แทนตำแหน่งที่อยู่ของข้อมูล(ในอาร์เรย์)

index	key	Value/Information
0		
1		
2		
....		

- ข้อมูลจะกระจายอยู่ในตารางที่เตรียมไว้โดยไม่มีการเรียงลำดับ ตำแหน่งที่เก็บข้อมูลเกิดจากการนำค่าคีย์มาเข้าสูตรคำนวณที่เตรียมไว้ล่วงหน้า เพื่อแปลงค่าคีย์ไปเป็นตัวเลข(index) ของอาร์เรย์ ถ้าตำแหน่งนั้นในอาร์เรย์ว่างอยู่ก็นำข้อมูลไปเก็บได้เลย แต่ถ้าตำแหน่งที่คำนวณได้มีข้อมูลตัวอื่นเก็บอยู่ก่อนแล้ว(เรียกว่าเกิดการชน Collision) ให้แก้ปัญหาโดยนำไปใส่ในตำแหน่งอื่นที่ว่างอยู่แทน
- การค้นหาข้อมูลที่ต้องการ ให้ทำกลับกัน โดยการนำค่าคีย์ไปเข้าสูตรเพื่อหาตำแหน่ง ถ้าตำแหน่งนั้นมีข้อมูลที่ตรงกับค่าคีย์ที่เก็บอยู่ก็นำข้อมูลไปใช้ แต่ถ้าไม่ใช่ก็ให้ใช้วิธีแก้ปัญหาเช่นเดียวกับตอนเกิด Collision

Hashing Functions

- ต้องสร้างสูตรที่ใช้แปลงคีย์ไปเป็นตำแหน่งที่เก็บข้อมูลในตาราง (Hashing Function)
- Hashing Functions

$$0 \leq h(K) < M \; ; \; M = \text{ขนาดของตารางที่เตรียมไว้}$$

- สูตรที่ดี เมื่อแปลงค่าคีย์แต่ละตัวไปเป็นตำแหน่งแล้วต้องไม่ซ้ำกัน(Collision) หรือซ้ำกันน้อยที่สุด ข้อมูลควรจะกระจายอยู่ทั่วทั้งตาราง ไม่กระจุกรวม(cluster)อยู่ในส่วนใดส่วนหนึ่งของตาราง
- Load Factor (LF)
  - อัตราส่วนระหว่าง ขนาดของข้อมูล(N) กับขนาดของตาราง(M)
  - ถ้าค่า LF ต่ำ โอกาสที่จะเจอตำแหน่งที่ว่างจะมีมาก

    LF = λ = N / M

    N คือขนาดของข้อมูลจริง , M คือขนาดของตารางที่เตรียมไว้
  - โดยทั่วไปมักจะกำหนดให้ LF ไม่เกิน 70% เพื่อลดโอกาสการเกิด Collision

Hashing Methods

- ตัวอย่างการสร้างสูตร Hashing function h(k) เพื่อแปลงค่าคีย์เป็น index ของอาร์เรย์ (สมมุติว่าค่าคีย์เป็นตัวเลข)
  - นำค่าคีย์(ตัวเลข) มาหารด้วยค่าคงที่(ขนาดของตารางนิยมกำหนดให้เป็นตัวเลข prime number) แล้วนำไปใช้เป็นตำแหน่งสำหรับเก็บข้อมูล

```
h(k) = k mod (Tsize);  // Tsize คือขนาดของตาราง
public static int  hash(long key, int Tsize) {
        return (int) key%Tsize;
}
```

- ตัวอย่างการสร้างสูตรเพื่อคำนวณค่าคีย์(สมมุติค่าคีย์เป็นตัวอักษร)

```
public static int  Stringhash(String key, int Tsize) {
    long  sum = 0;
        for (int i=0; i<key.length(); i++)
            sum = (long) (sum + key.charAt(i)*Math.pow(10,i));
        return (int) sum%Tsize;
}
```

Division Method

Example (20,19,34,10,31,3,42,14)

ข้อมูล 8 ตัว ใช้ตาราง 11 ช่อง LF = 8/11 = 73%

ถ้าใช้สูตร $h(K) = K \bmod 11 = K \% 11$

h(20)	=	20 mod 11	=	9	
h(19)	=	19 mod 11	=	8	
h(34)	=	34 mod 11	=	1	
h(10)	=	10 mod 11	=	10	
h(31)	=	31 mod 11	=	9	Collision h(20)
h(3)	=	3 mod 11	=	3	
h(42)	=	42 mod 11	=	9	Collision h(20)
h(14)	=	14 mod 11	=	3	Collision h(3)

0	1	2	3	4	5	6	7	8	9	10
	34		3 11					19	20 31 42	10

ต้องย้าย 11, 31, 42 ไปเก็บไว้ในตำแหน่งใหม่

Division Method

Example (ant,rat,cat,dog,pig,crab,fish,bird)

ข้อมูล 8 ตัว ใช้ตาราง 11 ช่อง LF = 8/11 = 73%

ถ้าใช้สูตร *h(K) = (sum of ASCII) mod 11*

ant	=	(97+110+116)%11	=	323%11	= 4
rat	=	(114+97+116)%11	=	327%11	= 8
cat	=	(99+97+116)%11	=	312%11	= 4
dog	=	(100+111+103)%11	=	314%11	= 6
pig	=	(112+115+103)%11	=	320%11	= 1
crab	=	(99+114+97+98)%11	=	408%11	= 1
fish	=	(102+105+115+104)%11	=	426%11	= 8
bird	=	(98+105+114+100)%11	=	417%11	= 10

0	1	2	3	4	5	6	7	8	9	10
	pig crab			ant cat		dog		rat fish		bird

ต้องย้าย crab, cat, fish ไปเก็บไว้ในตำแหน่งใหม่

Midsquare Method

Example (131,242,151,114,312,217,175)

ข้อมูล 7 ตัว ใช้ตาราง 10 ช่อง LF = 7/10 = 70%

ถ้ายกกำลังสองแล้วใช้ตัวเลขหลักร้อยมาเป็นตำแหน่ง (Table size 0-9)

131^2 =	17161	h(K) = 1
242^2 =	58564	h(K) = 5
151^2 =	22801	h(K) = 8
114^2 =	12996	h(K) = 9
312^2 =	97344	h(K) = 3
217^2 =	47089	h(K) = 0
175^2 =	30625	h(K) = 6

$$h(k) = (k*k / 100) \% 10$$

Select binary bits

Example (31,42,51,14,12,7,17)

ข้อมูล 7 ตัว ใช้ตาราง 16 ช่อง LF = 7/16 = 44%

ถ้ายกกำลังสองแล้วเลือกบิทที่ 5,4,3,2 (Table size = 2^4 = 16)

31^2	=	961	=	0000001111000001	=	0000	=	0
42^2	=	1764	=	0000011011100100	=	1001	=	9
51^2	=	2601	=	0000101000101001	=	1010	=	10
14^2	=	196	=	0000000011000100	=	0001	=	1
12^2	=	144	=	0000000010010000	=	0100	=	4
7^2	=	49	=	0000000000110001	=	1100	=	12
17^2	=	289	=	0000000100100001	=	1000	=	8

*h(k) = ((k*k) >> 2) & 15*

Bitwise Operator

Rightshift Operator [var]>>[num] , Leftshift Operator [var]<<[num]

Bitwise AND [var]&[var] , Bitwise OR [var]|[var] , Bitwise XOR [var]^[var]

Bitwise Complement ~[var]

Folding Method

Example (257145368,25842354,12487654,248645452,15874348,23107830)

ตัดคีย์ออกเป็น 3 ส่วนแล้วพับ (Tsize = 1000)

key	shift folding	boundary folding
257145368	257+145+368 = 770	752+145+863 = 760
25842354	025+842+354 = 221	520+842+453 = 815
12487654	012+487+654 = 153	210+487+456 = 153
248645452	248+645+452 = 345	842+645+254 = 741
15874348	015+874+348 = 237	510+874+843 = 227
23107830	023+107+830 = 960	320+107+038 = 465

Extraction Method

Example 80 Students have id = 50211501 .. 50211580

- **เลือกหลักสิบและหลักหน่วยมาใช้**

$$h(k) = k\%100$$

- **Example 80 Students have id = 50211501 .. 50211580 and 80 Students have id = 50270601 .. 50270680**
- **เลือกหลักที่ 3,1,0**

K = Student I.D.	h(K) = digit 3,1,0
50211501	101
50211502	102
50211503	103
...	...
50270601	001
50270602	002

$$h(k) = k\%10000/1000*100 + k\%100$$

%10000 = 4 หลักสุดท้าย

/1000 = ตัด 3 หลักสุดท้ายออก

%100 =2 หลักสุดท้าย

Collision Resolution

- บางครั้งค่าคีย์ที่ต่างกันอาจคำนวณออกมาเป็นตำแหน่งเดียวกันได้ เช่น

  10%9 , 19%9 , 28%9 คำนวนแล้วได้ตำแหน่งที่ 1 เหมือนกัน เรียกว่าเกิดการชน(Collision) ต้องหาวิธีจัดการกับข้อมูลเหล่านี้

- การแก้ปัญหาเมื่อเกิดการชนกันของตำแหน่งข้อมูล
  - แก้ด้วยวิธีหาตำแหน่งใหม่ (Open Addressing)
    - Linear Probing เลื่อนหาตำแหน่งข้างเคียงที่ว่างอยู่
    - Quadratic Probing เลื่อนหาตำแหน่งที่ว่างอยู่แบบกำลังสอง
    - Double Hashing สร้างสูตรใหม่เพิ่มเติม

```
i = 0 ;  success = 0;
do {  h = ( hash_function  + collision_resol ) % table_size ;
      if  ( table[h] == null )
         { table[h] = data ;
           success = 1; }
      else
           i++ ;
   } while (success == 0) ;
```

  - แก้ด้วยวิธีใช้ลิงค์ลิสต์ (Separate Chaining)

Linear Probing

$$h(k)i = (h(k) + i \times C) \bmod m$$

Example (415, 604, 871, 921, 163, 121, 895)

Table address is 0 - 10 h(k) = k mod 11 C = 1

415 mod 11 = 8

604 mod 11 = 10

871 mod 11 = 2

921 mod 11 = 8 => 8 + 1 = 9

163 mod 11 = 9 => 9 + 1 = 10 => 10 + 1 = 11

=> 11 mod 11 = 0

121 mod 11 = 0 => 0 + 1 = 1

895 mod 11 = 4

การเลื่อนข้อมูลไปยังตำแหน่งข้างเคียงทำให้เกิดปัญหาข้อมูลอยู่รวมกันเป็นกลุ่ม เรียกว่า Clustering

0	1	2	3	4	5	6	7	8	9	10
163^1	121^1	871		895				415	921^1	604

การแก้ปัญหาการชนด้วยการเลื่อนข้อมูลไปด้านข้างทีละตัว มักจะทำให้เกิดการกระจุกตัวของข้อมูล (clustering)

Linear Probing

Example (415, 604, 871, 163, 121, 921, 895)

Table address is 0 - 10 **h(k) = k mod 11 C = 1**

415 mod 11 = 8

604 mod 11 = 10

871 mod 11 = 2

163 mod 11 = 9

121 mod 11 = 0

921 mod 11 = 8 => 8 + 1 = 9 => 9 + 1 = 10

=> 10 + 1 = 11 => 11 mod 11 = 0 => 0 + 1 = 1

895 mod 11 = 4

0	1	2	3	4	5	6	7	8	9	10
121	921^{4}	871		895				415	163	604

Quadratic Probing

$$h(k)i = (h(k) + i^2) \bmod m$$

แก้ปัญหาข้อมูลอยู่รวมกันเป็นกลุ่ม ด้วยการกระจายข้อมูลให้ห่างออกไป(กำลังสอง)

The sequence of probes = 0, 1, 4, 9, 16, 25, …

Example (415, 604, 871, 921, 163, 121, 895)

Table size is 0 – 10 **Hashing function is k mod 11**

415 mod 11 = 8

604 mod 11 = 10

871 mod 11 = 2

921 mod 11 = 8 => 8 + 1 = 9

163 mod 11 = 9 => 9 + 1 = 10 => 9 + 4 = 13 => 13 mod 11 = 2

=> 9 + 9 = 18 => 18 mod 11 = 7

121 mod 11 = 0

895 mod 11 = 4

0	1	2	3	4	5	6	7	8	9	10
121		871		895			163^3	415	921^1	604

Double Hashing

$$h(k)i = (h(k) + i \times h'(k)) \bmod m$$

แก้ปัญหาการชนกัน ด้วยการบวกฟังก์ชันใหม่

Example (415, 604, 871, 921, 163, 121, 895)

Table size is 0-10 ; h(k) = k mod 11; h'(k) = k mod 9

415 mod 11 = 8

604 mod 11 = 10

871 mod 11 = 2

921 mod 11 = 8 => 8 + 1*(921 mod 9) = 8+3=11 => 0

163 mod 11 = 9

221 mod 11 = 1

895 mod 11 = 4

0	1	2	3	4	5	6	7	8	9	10
921[1]	121	871		895				415	163	604

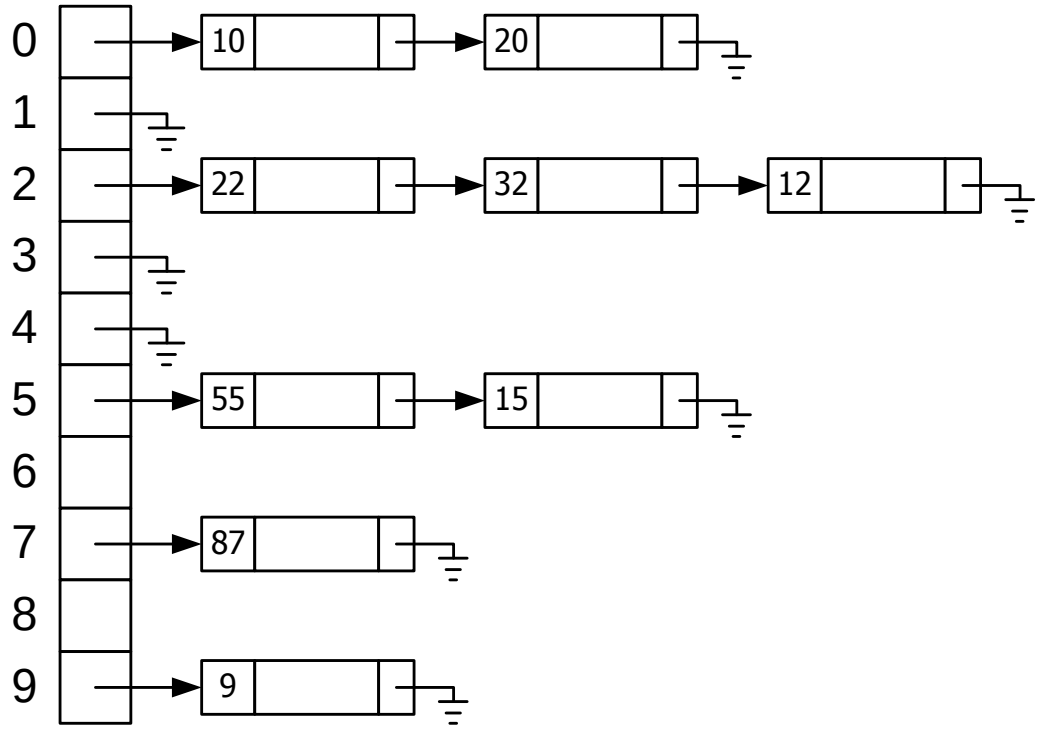
Coding Example

```
static int  Save_Hashing ( int  [] table , int data)
{  int  h, i = 0 ;
       do  {
              h =  (hash_function (data) + collision_resolve(i) )%Tsize;
              i++ ;
       }while  (table[h] != 0)  ;
       table[h] = data ;
       return  h; // return ตำแหน่งที่เก็บ
}
static  int  Search_Hashing ( int  [] table , int data)
{  int  h, i = 0 ;
       do  {
              h =  (hash_function (data) + collision_resolve(i))%Tsize ;
              i++ ;
           } while  (table[h] != key) && table[h] != 0)  ;
       if  (table[h] != 0)
          return  h; // return ตำแหน่งที่ค้นเจอ
       else
          return -1;  // กำหนดให้ -1 คือค้นไม่เจอ
}
```

Separate Chaining

- สร้างตารางของลิงค์ลิสต์ ชี้ไปยังตำแหน่งที่เก็บข้อมูลที่เกิดการชน

Example ถ้าตัวที่เกิดการชนคือ (10, 22, 55, 87, 9, 20, 32, 12, 15)
สร้างลิงค์ลิสต์ไว้ 10 ตาราง h(k) = k mod 10




```
ArrayList <Node> [] collisionTable = (ArrayList <Node> []) new ArrayList[10];
for (int i = 0; i<10; i++)
    collisionTable[i] = new ArrayList<Node>();
```

Re-Hashing

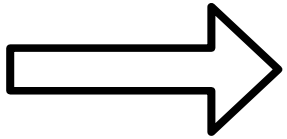
- ถ้ามีข้อมูลเพิ่มขึ้น ทำให้ขนาดตารางเดิม เล็กเกินไป

- เปลี่ยนสูตร เปลี่ยนตาราง (Re-Hashing)

Example (20,19,34,10,31,3,42)

(k mod 9) -> (k mod 15)

k%9	
0	
1	19
2	20
3	10
4	31
5	3
6	42
7	34
8	



k%15	
0	
1	31
2	
3	3
4	19
5	20
6	34
7	
8	
9	
10	10
11	
12	42
13	
14	

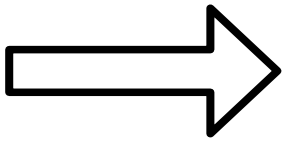
Extendible Hashing

- ใช้สูตรเดิมแต่เพิ่มจำนวนตาราง โดยกำหนดเงื่อนไขในการเลือกตารางจากค่าคีย์

Example (20,19, 34, 10, 31, 3, 42)

h(k) = k mod 9

T	
0	
1	19
2	20
3	10
4	31
5	3
6	42
7	34
8	



T[1]	
k<=20	
0	
1	19
2	20
3	10
4	3
5	
6	
7	
8	

T[2]	
k>20	
0	
1	
2	
3	
4	31
5	
6	42
7	34
8	