

# Topics: Closed Hashing

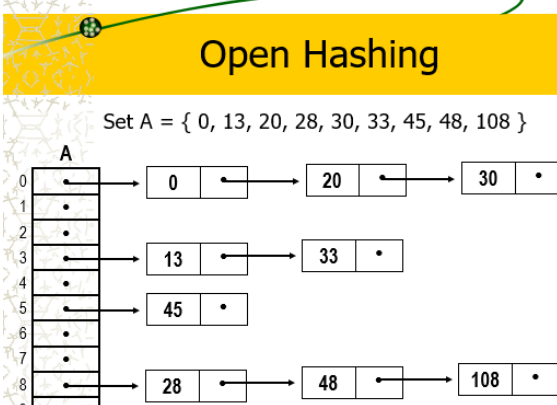
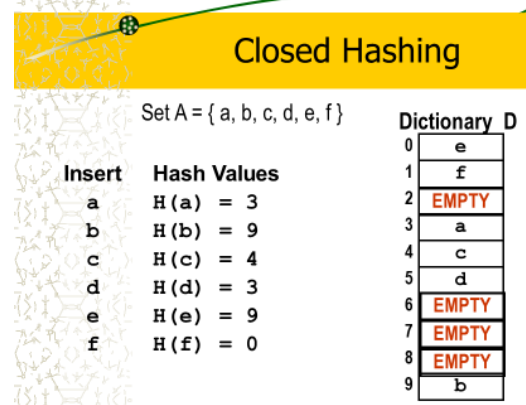
## Two types of Hashing

### A. Open Hashing (or external hashing)

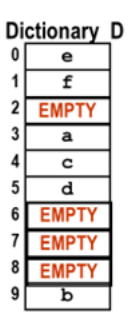
✚ This allows the set to be stored in potentially unlimited space.

### B. Closed Hashing (or internal hashing)

✚ This uses a fixed space for storage and thus limits the size of the sets

Open Hashing (or external hashing)	Closed Hashing (or internal hashing)														
 <p>Set A = { 0, 13, 20, 28, 30, 33, 45, 48, 108 }</p> <p><b>Hash function:</b> Groups the elements according to the last digit of the element.</p> <p><b>Array size:</b> No. of groups</p>	 <p>Set A = { a, b, c, d, e, f }</p> <p><b>Insert Hash Values</b></p> <table border="1"> <thead> <tr> <th>Insert</th> <th>Hash Values</th> </tr> </thead> <tbody> <tr> <td>a</td> <td>H(a) = 3</td> </tr> <tr> <td>b</td> <td>H(b) = 9</td> </tr> <tr> <td>c</td> <td>H(c) = 4</td> </tr> <tr> <td>d</td> <td>H(d) = 3</td> </tr> <tr> <td>e</td> <td>H(e) = 9</td> </tr> <tr> <td>f</td> <td>H(f) = 0</td> </tr> </tbody> </table> <p><b>Assumption:</b> There exists a hash function H() that returns a hash value for each of the elements a, b, c, d, e, and f.</p> <p><b>Array Size:</b> &gt;= No. of elements</p> <p><b>Rule of Thumb:</b> The packing density of the array, i.e. the ratio of the number of elements and the array size is 80%.</p>	Insert	Hash Values	a	H(a) = 3	b	H(b) = 9	c	H(c) = 4	d	H(d) = 3	e	H(e) = 9	f	H(f) = 0
Insert	Hash Values														
a	H(a) = 3														
b	H(b) = 9														
c	H(c) = 4														
d	H(d) = 3														
e	H(e) = 9														
f	H(f) = 0														
<ul style="list-style-type: none"> <li>This allows the set to be stored in potentially unlimited space</li> <li>Hash function determines the group that the element is part of</li> </ul>	<ul style="list-style-type: none"> <li>This uses a fixed space for storage and thus limits the size of the sets</li> <li>Hash function determines the exact location of the element or starting point in searching the element.</li> </ul>														

## Practice Exercise 1:

	<ol style="list-style-type: none"> <li>Write an appropriate definition of data type Dictionary, such that int the declaration Dictionary D; D is an array of size 10 of characters. In addition, define as macros the following: <ol style="list-style-type: none"> <li>SIZE - define as the array size</li> <li>EMPTY and DELETED – values will be of the same data type as the elements of the dictionary (or a field in the structured element) but are not part of the set of elements in the dictionary. Example are the non-alpha letters such as *, #, ?, etc.</li> </ol> </li> <li>Write the code of function initDictionary(). The function will initialize all array components to EMPTY.</li> <li>Assume that a hash function H() exists and it returns to the calling function the hash value of the given character element, write the code of the functions: <ol style="list-style-type: none"> <li>insert()</li> <li>delete()</li> <li>member()</li> </ol> </li> </ol>
---	---

## Closed Hashing pa more...

### Closed Hashing

Set A = { a, b, c, d, e, f }

Insert	Hash Values
a	H(a) = 3
b	H(b) = 9
c	H(c) = 4
d	H(d) = 3
e	H(e) = 9
f	H(f) = 0

### Terms associated with closed hashing:

**Synonyms:** two or more elements that have the same hash values.  
Example: a and d , b and e

**Displacement:** a condition wherein an element can not be stored in the position returned by the hash function because a non-synonym is stored in that position. Example: H(f) = 0, f can not be stored at index 0 because it is displaced by a non-synonym e, i.e. H(e)=9

**Collision:** an event which occurs when an element is attempted to be inserted at the position returned by the hash function, and the position is already occupied by another element. Example: H(d) = 3, d can not be stored at index 3 because it is occupied by element a.

**Linear Hashing:** a solution to collision wherein the element is placed in the next available position in the circular array.

Formula:  $H_i(x) = ( H(x) + i ) \% MAX$

Where MAX is the size of the preallocated space or size of the array.

**Load factor or packing density** – the ratio of the number of elements to be stored to the number of available spaces.

**Rule of thumb:** Packing density is 80%

0	e
1	f
2	EMPTY
3	a
4	c
5	d
6	EMPTY
7	EMPTY
8	EMPTY
9	b

## Research on the following:

- 2) Perfect Hash function
- 3) Average Search Length

## Exercise

(Average Search Length)

**Hash Values**

Hash(A) = 1  
Hash(B) = 4  
Hash(C) = 9  
Hash(D) = 9  
Hash(E) = 0  
Hash(F) = 3  
Hash(G) = 4  
Hash(H) = 3

**Dictionary D**

0	EMPTY
1	EMPTY
2	EMPTY
3	EMPTY
4	EMPTY
5	EMPTY
6	EMPTY
7	EMPTY
8	EMPTY
9	EMPTY

Do the following:

- Insert the elements A, B, C, D, E, F, G, and H in an initially empty dictionary with hash values 1, 4, 9, 9, 0, 3, 4, and 3 respectively. Note: Solution for collision is linear hashing, i.e. next available space in the dictionary which is treated as a circular array
- Determine the search length of each element. Search length (SL) of element x:  
 $SL = \text{Actual location of } x - \text{Hash}(x) + 1$
- Determine the Average Search Length of all 8 elements  
 $\text{Ave SL} = \text{sum of SL} / \text{no. of elements}$

Note: The formula for Search length is only applicable for elements whose actual location is greater than or equal to the value returned by the hash function. Find the formula for elements whose actual location is less than the value returned by the hash function. Challenge: Find the formula that will work for both scenarios.