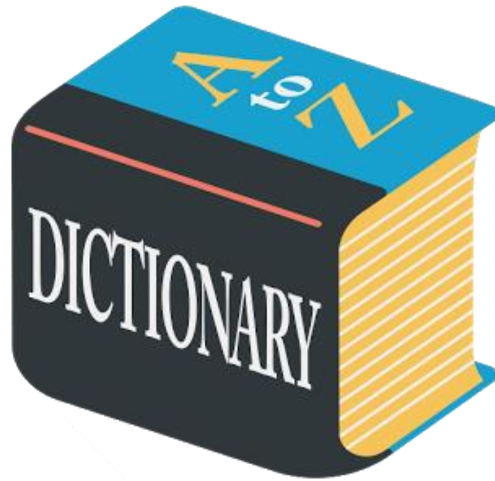
Dictionaries
&
Hash Tables



Unordered Dictionary ADT

- A dictionary stores key-element pairs (k,e), which is called items, where k is the key and e is the element.
- There are two types of dictionaries:
  - Unordered dictionary
  - Ordered dictionary
- Main operations of dictionary D: find, insert, remove
  - findElement(*k*), insertItem(*k*, *o*), removeElement(*k*)
  - size(), isEmpty()
  - keys(), elements()

- Applications:
  - address book
  - word-definition pairs
  - mapping host names to internet addresses (e.g., www.cs16.net to 128.148.34.101)

Log File

- A log file is a file that records either events that occur in an operating system or other software runs, or messages between different users of a communication software. [Wikipedia]
- A log file is a dictionary implemented by means of storing items in an unsorted sequence
  - insertItem takes ***O***(1) time since we can insert the new item at the beginning or at the end of the sequence
  - findElement and removeElement take ***O***(***n***) time since in the worst case (the item is not found) we traverse the entire sequence to look for an item with the given key
- Effective only for dictionaries of
  - small size or
  - when insertions are the most common operations, while searches and removals are rarely performed (e.g., historical record of logins to a workstation)

Dictionary

	insertItem	removeElement	Findelement
List/Vector	O(1)	O(n)	O(n)

Goal:

All operations run in O(1) time on Average.

Hash Table - based Dictionaries

Hash Functions and Hash Tables

- A hash table for a given key type consists of
  - Large Array (called table) of size N
  - Hash function h

- A hash function is any function that can be used to map a data set of an arbitrary size to a data set of a fixed size, which falls into the hash table.
- The goal of the hash function is to "disperse" the keys in an apparently random way.
- A hash function h maps keys of a given type to integers in a fixed interval $[0, N - 1]$
  - Ex: $h(x) = x \bmod N$ is a hash function for integer keys
  - The integer $h(x)$ is called the hash value of key x

- When implementing a dictionary with a hash table, the goal is to store item (k, o) at index $i = h(k)$

Example

Hash Function $\boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{x} \bmod \boldsymbol{N}$

Hash Table of size N=8

Add these keys to the hash table:

18 21 43 15 36

A[0]	
A[1]	
A[2]	
A[3]	
A[4]	
A[5]	
A[6]	
A[7]	

Example

Hash Function $h(x) = x \bmod N$

Add these keys to the hash table:

18 21 43 15 36

h(18) = 18 mode 8 = 2
h(21) = 21 mode 8 = 5
h(43) = 43 mode 8 = 3
h(15) = 15 mode 8 = 7
h(36) = 36 mode 8 = 2

Hash Table of size N=8

Index	Value
A[0]	
A[1]	
A[2]	18
A[3]	43
A[4]	36
A[5]	21
A[6]	
A[7]	15

Example

Hash Function $h(x) = x \bmod N$

Add these keys to the hash table:

18 21 43 15 36

h(18) = 18 mode 8 = 2
h(21) = 21 mode 8 = 5
h(43) = 43 mode 8 = 3
h(15) = 15 mode 8 = 7
h(36) = 36 mode 8 = 4

Now let's add 10.

h(10) = 10 mode 8 = 2

This is called **Collision.**

Hash Table of size N=8

A[0]	
A[1]	
A[2]	18
A[3]	43
A[4]	36
A[5]	21
A[6]	
A[7]	15

collision

- Two keys may hash to the same slot, which is called a ***collision***.
- Fortunately, we have effective techniques for resolving the conflict created by collisions.

- Questions

1. How to design h such that number of collisions is low?
2. How do we handle collisions?

Hash function

- To achieve a good hashing mechanism, It is important to have a good hash function with the following basic requirements:
  - **Easy to compute**: It should be easy to compute and must not become an algorithm in itself.
  - **Uniform distribution**: It should provide a uniform distribution across the hash table and should not result in clustering.
  - **Less collisions**: Collisions occur when pairs of elements are mapped to the same hash value. These should be avoided.

Hash Functions

- A hash function h(k), consist of two actions
  - mapping the key k to an integer, called the hash code,
  - and mapping the hash code to an integer within the range of indices of a array, called the compression map.

Hash code map

$\boldsymbol{h}_1$: keys $\rightarrow$ integers

Compression map

$\boldsymbol{h}_2$: integers $\rightarrow [0, N - 1]$

The hash code map is applied first, and the compression map is applied next on the result

$$\boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{h}_2(\boldsymbol{h}_1(\boldsymbol{x}))$$

Compression Maps: integers $\rightarrow [0, N-1]$

- A good hash function guarantees the probability that two different keys have the same hash is 1/N.
- The size N of the hash table is usually chosen to be a prime.
  - The reason involves number theory and is beyond the scope of this course

Division

- $h_2(k) = k \bmod N$
- disadvantage: repeated keys of the form $iN + j$ cause collisions

Multiply, Add and Divide (MAD)

- $h_2(k) = (ak + b) \bmod N$
- Where a and b are nonnegative integers

Collision Handling

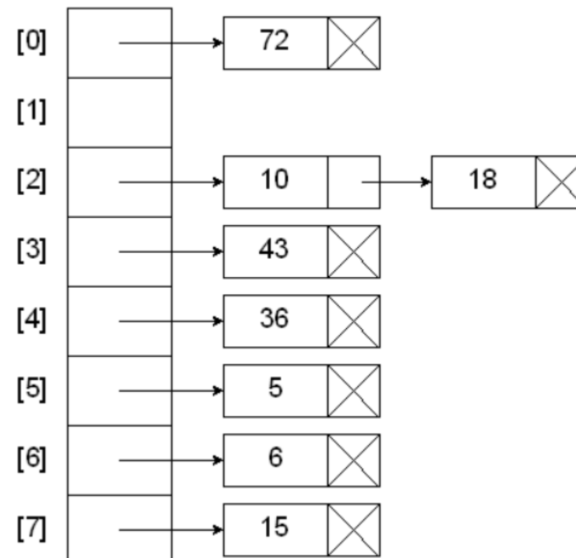
Collisions occur when different elements are mapped to the same cell

Chaining

- each cell in the table points to a linked list of elements that map there
- simple, but requires additional memory outside the table.

Hash key = key % table size

4 = 36 % 8
2 = 18 % 8
0 = 72 % 8
3 = 43 % 8
6 = 6 % 8
2 = 10 % 8
5 = 5 % 8
7 = 15 % 8



Chaining

- The separate chaining rule has one slight disadvantage:
  - it requires the use of an auxiliary data structure-a list, vector, or sequence-to hold items with colliding keys

Collision Handling

Open Addressing

- the colliding item is placed in a different cell of the table
- no additional memory, but complicates searching/removing
- common types:

  1) linear probing,

  2) quadratic probing

  3) double hashing

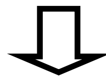
Open Addressing: Linear Probing

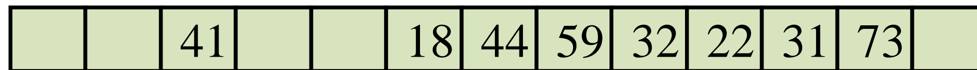
- Placing the colliding item in the next (circularly) available table cell

$$A[(\boldsymbol{h}(\boldsymbol{k}) + \boldsymbol{i}) \bmod N] \quad \text{for } \boldsymbol{i} = 0,1,2,\ldots$$

- Disadvantage:
  - Colliding items cluster together, causing future collisions to cause a longer sequence of probes (searches for next available cell)
- Example:
  - $\boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{x} \bmod 13$
  - Insert keys 18, 41, 22, 44, 59, 32, 31, 73, in this order

0 1 2 3 4 5 6 7 8 9 10 11 12





0 1 2 3 4 5 6 7 8 9 10 11 12

$h(18)$ = 18 mod 13 = 5
41 mod 13 = 2
22 mod 13 = 9
44 mod 13 = 5
59 mod 13 = 7
32 mod 13 = 6
31 mod 13 = 5
73 mod 13 = 8

Search with Linear Probing

Consider a hash table *A* that uses linear probing

findElement(***k***)

- Start at cell ***h***(***k***)
- Check consecutive locations until one of the following occurs
  - An item with key ***k*** is found, or
  - An empty cell is found, or
  - *N* cells have been unsuccessfully probed

Algorithm *findElement*(*k*)
    $i \leftarrow h(k)$
    $p \leftarrow 0$
    repeat
        $c \leftarrow A[i]$
        if $c = \varnothing$
            return *NO_SUCH_KEY*
        else if *c.key* () $= k$
            return *c.element*()
        else
            $i \leftarrow (i + 1) \bmod N$
            $p \leftarrow p + 1$
    until $p = N$
    return *NO_SUCH_KEY*

Example – Delete Problem

		18	43	36	21		15
0	1	2	3	4	5	6	7

1) Insert 26

		18	43	36	21	26	15
0	1	2	3	4	5	6	7

2) Delete 18
3) Find 26

Updates with Linear Probing

A special object, called ***AVAILABLE***, replaces deleted elements

- removeElement(***k***)
  - Search for an item with key ***k***
  - If it is found, replace it with item ***AVAILABLE*** and return element
  - Else, return ***NO_SUCH_KEY***

- insertItem(***k, o***)
  - Throw an exception if the table is full
  - Start at cell ***h***(***k***)
  - Search consecutive cells until a cell ***i*** is found that is either empty or stores ***AVAILABLE***
  - Store item (***k, o***) in cell ***i***

Open Addressing: Quadratic Probing

- Placing the colliding item in

$A[(\boldsymbol{h}(\boldsymbol{k}) + f(\boldsymbol{i})) \bmod \boldsymbol{N}]$ for $\boldsymbol{i} = 0,1,2,\ldots$

Where $f(i) = i^2$

- As with linear probing, the quadratic probing strategy complicates the removal operation,
- but it does avoid the kinds of clustering patterns that occur with linear probing.

Open Addressing: Double Hashing

- Use a secondary hash function $d(k)$ to place items in first available cell

  try $A[(h(k) + i*d(k)) \bmod N]$ for $i = 0,1,2,\ldots$
- $d(k)$ cannot have zero values

- The table size N must be a prime to allow probing of all the cells

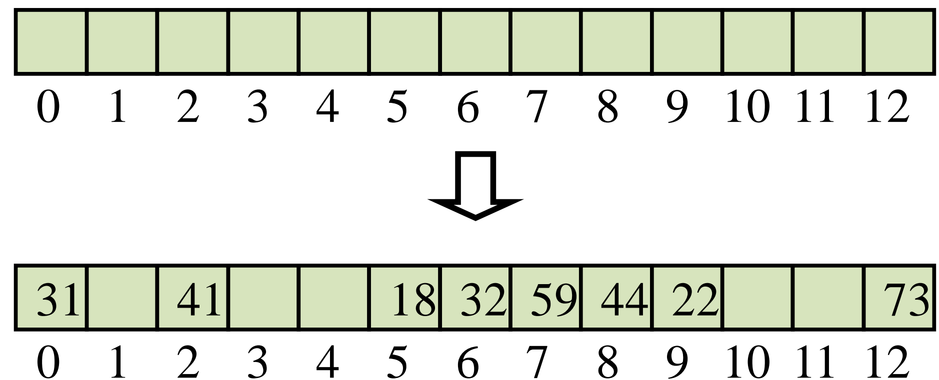
Example of Double Hashing

Consider a hash table storing integer keys that handles collision with double hashing

- $N = 13$
- $h(k) = k \bmod 13$
- $d(k) = 1 + (k \bmod 7)$

Insert keys 18, 41, 22, 44, 59, 32, 31, 73, in this order

k	$h(k)$	$d(k)$	Probes		
18	5	5	5		
41	2	7	2		
22	9	2	9		
44	5	3	5	8	
59	7	4	7		
32	6	5	6		
31	5	4	5	9	0
73	8	4	8	12	



Performance of Hashing

- In the worst case, searches, insertions and removals on a hash table take $O(n)$ time
  - occurs when all inserted keys collide
- However, the expected running time of all the dictionary ADT operations in a hash table is $O(1)$
- The **load factor** is the number of keys stored in the hash table divided by the capacity.
- In other words, The load factor is a measure of how full the hash table is allowed to get before its capacity is automatically increased.
- So, given a hash table T with **m** slots that stores **n** elements, we define the

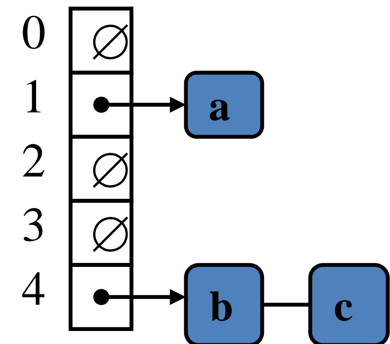
  load factor $\alpha = n/m$

- With the chaining n could be greater than m and so α might be ≥ 1.
- So, with open addressing, at most one element occupies each slot, and thus $n \leq m$, which implies $\alpha \leq 1$.

Chaining vs. Open Addressing

Chaining

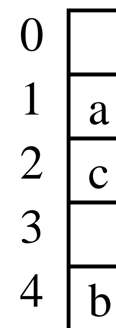
- Less sensitive to hash functions and load factor
- Supports $\alpha > 100\%$



h(a) = 1 h(b) = 4 h(c) = 4

Open Addressing

- Requires careful selection of hash function to avoid clustering
- Degrades past $\alpha > 70\%$
- Can't support $\alpha > 100\%$
- Better memory usage



Exercises

- You are given an array A of integers. Determine the integer that occurs most frequently in A.
- Demonstrate what happens when we insert the keys 5, 28, 19, 15, 20, 33, 12, 17,10 into a hash table with collisions resolved by chaining. Let the table have 9 slots, and let the hash function be $h(k) = k \ mod \ 9$.

Exercises

- Professor Marley hypothesizes that he can obtain substantial performance gains by modifying the chaining scheme to keep each list in sorted order. How does the professor's modification affect the running time for successful searches, unsuccessful searches, insertions, and deletions?
- Consider inserting the keys 10, 22,31, 4, 15, 28, 17, 88, 59 into a hash table of length m=11 using open addressing with the auxiliary hash function $h(k) = k$. Illustrate the result of inserting these keys using linear probing, using quadratic. probing, and using double hashing with $h_1(k) = k$ and $h_2(k) = 1 + (k \ mod \ (m \ - 1))$.