CSCI 3230 Data Structures

Maps and Hash Tables

Weitian Tong, Ph.D.

Department of Computer Science
Georgia Southern University
Website: www.weitiantong.com
Email:wtong@georgiasouthern.edu

Table of contents

Map

Map

A map models a searchable collection of key-value entries

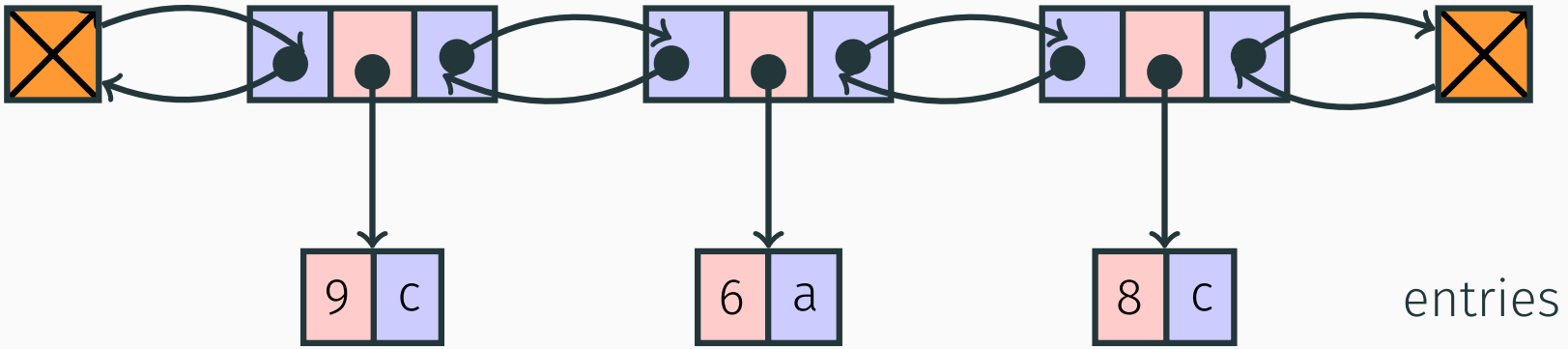
The main operations of a map are for searching, inserting, and deleting items

Multiple entries with the same key are not allowed

```
1  Map<K,V> {
2      size();
3      boolean isEmpty();
4      get(K key);
5      put(K key, V value);
6      remove(K key);
7  }
```

A Simple List-Based Map

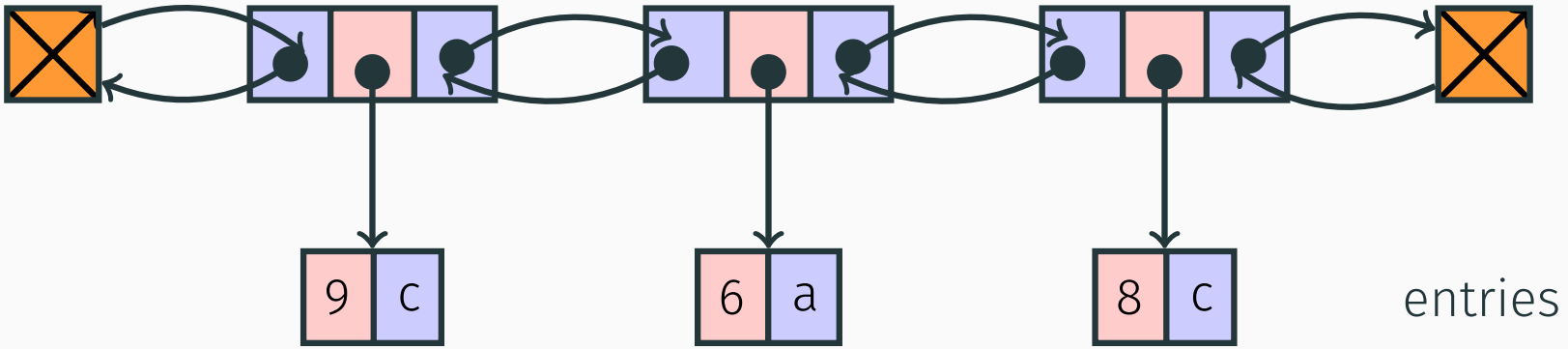
We can implement a map using an unsorted list




```
1 Algorithm get(k):
2 B = S.header
3 while B.hasNext() do
4   p = B.next() // the next position in B
5   if p.element().getKey() = k then
6     return p.element().getValue()
7 return null // there is no entry with key equal to k
```

A Simple List-Based Map: continue

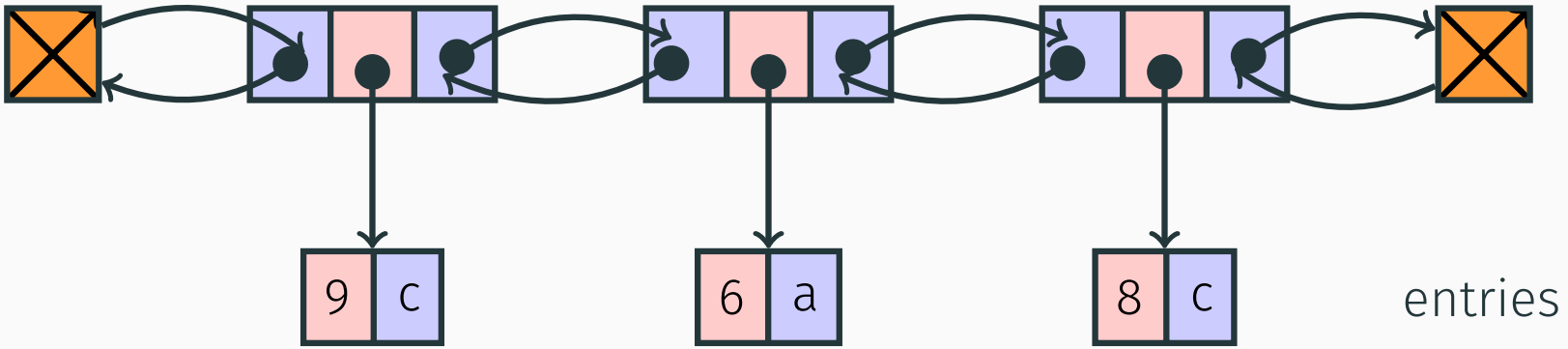
We can implement a map using an unsorted list




```
1  Algorithm put(k,v):
2  B = S.header
3  while B.hasNext() do
4    p = B.next()
5    if p.element().getKey() = k  then
6      t = p.element().getValue()
7      S.set(p,(k,v))
8      return t  // return the old value
9  S.addLast((k,v))
10 n = n + 1
11 return null
```

A Simple List-Based Map: continue

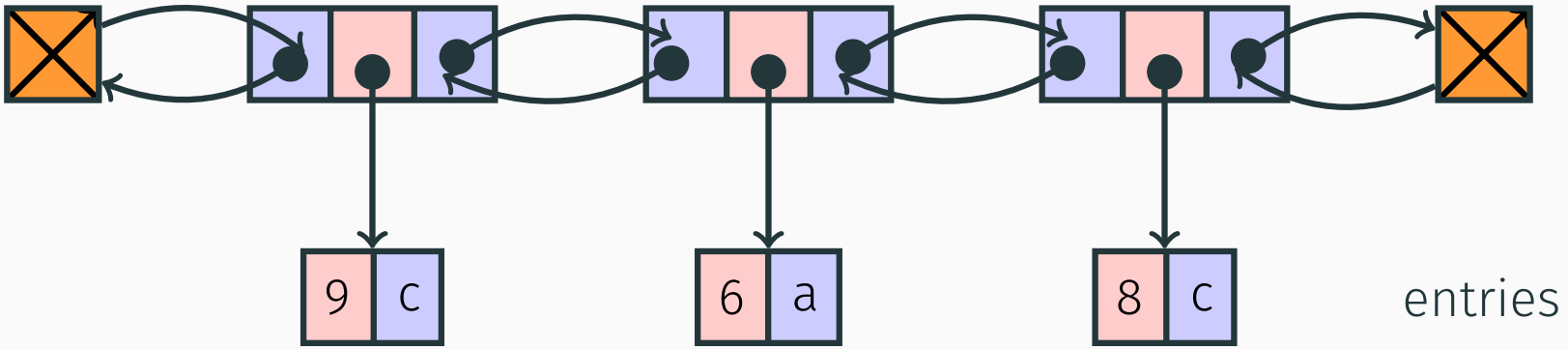
We can implement a map using an unsorted list




```
1  Algorithm remove(k):
2  B =S.header
3  while B.hasNext() do
4    p = B.next()
5    if p.element().getKey() = k  then
6      t = p.element().getValue()
7      S.remove(p)
8      n = n − 1   // decrement number of entries
9      return t  // return the removed value
10 return null
```

A Simple List-Based Map: continue

We can implement a map using an unsorted list



Performance:

- put, get and remove 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

How to make search operation faster in a Map?

Hash Table

Hash Tables

A **hash table** is a data structure to make map operations faster.

While worst-case is still $O(n)$, average case is typically $O(1)$.

Main idea:

- Indexing into an array takes $O(1)$ time.
- Mapping keys into integers (or indecies).

Examples:

- Search a word in a dictionary
- Search a phone number in your contact list.

Hash Functions and Hash Tables

The integer $h(x)$ is called the hash value of key x.

Hash table

A hash table for a given key type consists of

- Hash function h
- Array (called table) of size N

When implementing a map with a hash table, **the goal is to store item (k, v) at index $i = h(k)$.**

Hash function

A hash function h maps keys of a given type to integers in a fixed interval $[0, N - 1]$.

Example: $h(x) = x \mod N$

Hash Functions

A hash function

$$h(x) = h_2(h_1(x))$$

is usually specified as the composition of two functions:

- **Hash code**:

$$h_1 : keys \rightarrow integers$$

- **Compression function**:

$$h_2 : integers \rightarrow [0, N-1]$$

The **goal** of the hash function is to "disperse" the keys in an apparently random way

Hash codes

Hash code: $h_1 : keys \rightarrow integers$

Hash codes

Hash code: $h_1 : keys \to integers$

- Memory address
- Binary representation (Integer cast)
- Component sum
  - We partition the bits of the key into components of fixed length (e.g., 16 or 32 bits) and we sum the components (ignoring overflows)
- Polynomial accumulation
  - We partition the bits of the key into a sequence of components of fixed length (e.g., 8, 16 or 32 bits) $a_0, a_1, \ldots, a_{n-1}$
  - We evaluate the polynomial

$$p(z) = a_0 + a_1 z + a_2 z^2 + \ldots + a_{n-1} z^{n-1}$$

    at a fixed value z, ignoring overflows

Compression functions

Compression function:

$$h_2 : integers \rightarrow [0, N-1]$$

- **Division**: $h_2(y) = y \mod N$

  The size N of the hash table is usually chosen to be a prime

- **Multiply, Add and Divide (MAD)**: $h_2(y) = (ay + b) \mod N$

  a and b are nonnegative integers such that $a \mod N \neq 0$

Hash Table

Collision Handling

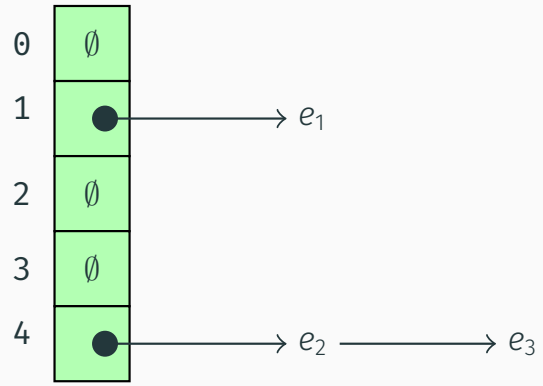
Collision Handling: Separate Chaining

Collisions: different elements are mapped to the same cell

Collision Handling: Separate Chaining

Collisions: different elements are mapped to the same cell

Separate Chaining: let each cell in the table point to a linked list of entries that map there



Separate chaining is simple, but requires additional memory outside the table

Collision Handling: Open addressing

Open addressing: the colliding item is placed in a different cell

- **Linear probing**: handles collisions by placing the colliding item in the next (circularly) available table cell
- **Double hashing**: uses a secondary hash function d(k) and handles collisions by placing an item in the first available cell of the series $(h(k) + jd(k)) \mod N$ for $j = 0, \ldots, N - 1$

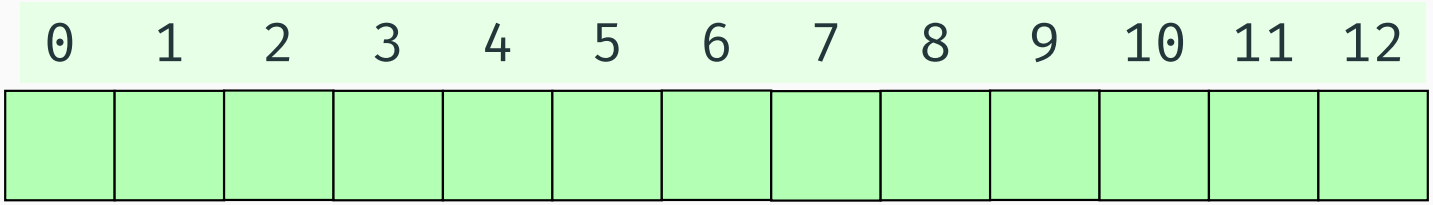
Collision Handling: Open addressing

Open addressing: the colliding item is placed in a different cell

- **Linear probing**: handles collisions by placing the colliding item in the next (circularly) available table cell
- **Double hashing**: uses a secondary hash function d(k) and handles collisions by placing an item in the first available cell of the series $(h(k) + jd(k)) \mod N$ for $j = 0, \ldots, N-1$

Example for Linear probing:

- $h(x) = x \mod 13$
- Insert keys 18, 41, 22, 44, 59, 32, 31, 73, in this order



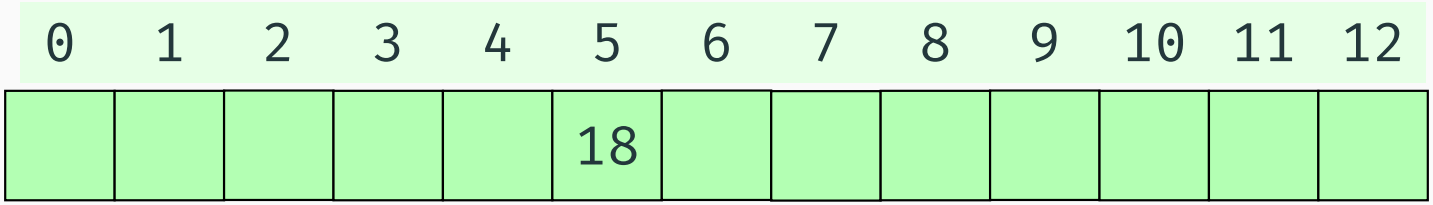
Collision Handling: Open addressing

Open addressing: the colliding item is placed in a different cell

- **Linear probing**: handles collisions by placing the colliding item in the next (circularly) available table cell
- **Double hashing**: uses a secondary hash function d(k) and handles collisions by placing an item in the first available cell of the series $(h(k) + jd(k)) \mod N$ for $j = 0, \ldots, N-1$

Example for Linear probing:

- $h(x) = x \mod 13$
- Insert keys 18, 41, 22, 44, 59, 32, 31, 73, in this order



$18 \mod 13 = 5$

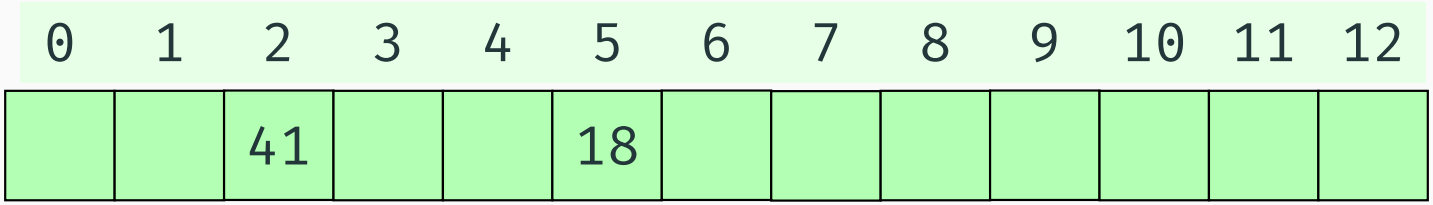
Collision Handling: Open addressing

Open addressing: the colliding item is placed in a different cell

- **Linear probing**: handles collisions by placing the colliding item in the next (circularly) available table cell
- **Double hashing**: uses a secondary hash function d(k) and handles collisions by placing an item in the first available cell of the series $(h(k) + jd(k)) \mod N$ for $j = 0, \ldots, N-1$

Example for Linear probing:

- $h(x) = x \mod 13$
- Insert keys 18, 41, 22, 44, 59, 32, 31, 73, in this order



$41 \mod 13 = 2$

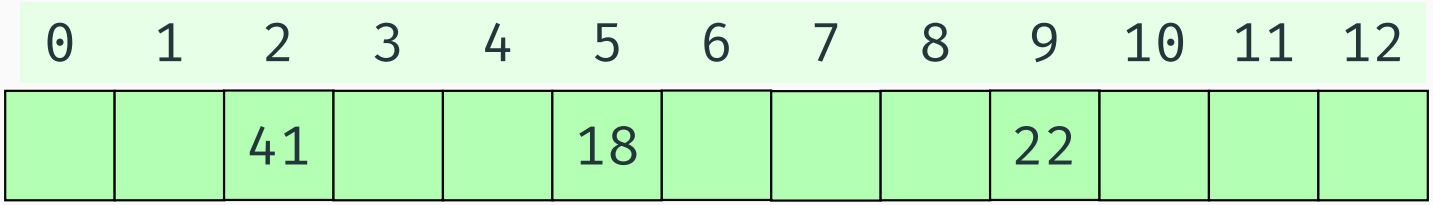
Collision Handling: Open addressing

Open addressing: the colliding item is placed in a different cell

- **Linear probing**: handles collisions by placing the colliding item in the next (circularly) available table cell
- **Double hashing**: uses a secondary hash function d(k) and handles collisions by placing an item in the first available cell of the series $(h(k) + jd(k)) \mod N$ for $j = 0, \ldots, N-1$

Example for Linear probing:

- $h(x) = x \mod 13$
- Insert keys 18, 41, 22, 44, 59, 32, 31, 73, in this order



$22 \mod 13 = 9$

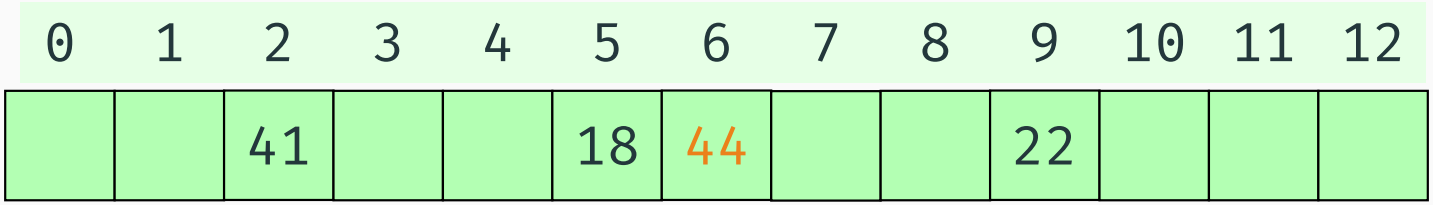
Collision Handling: Open addressing

Open addressing: the colliding item is placed in a different cell

- **Linear probing**: handles collisions by placing the colliding item in the next (circularly) available table cell
- **Double hashing**: uses a secondary hash function d(k) and handles collisions by placing an item in the first available cell of the series $(h(k) + jd(k)) \mod N$ for $j = 0, \ldots, N-1$

Example for Linear probing:

- $h(x) = x \mod 13$
- Insert keys 18, 41, 22, 44, 59, 32, 31, 73, in this order



$44 \mod 13 = 5$

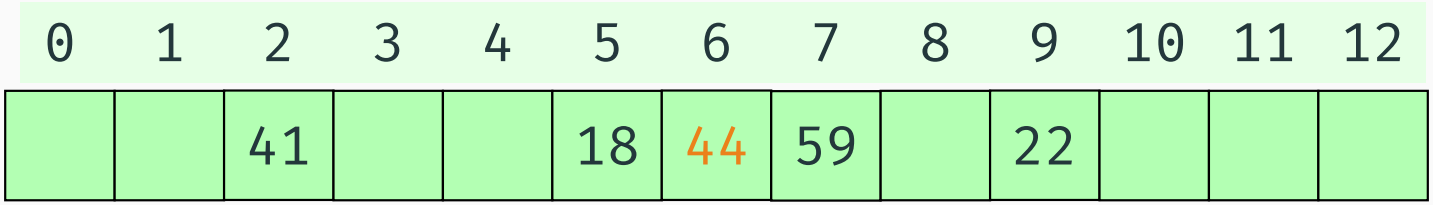
Collision Handling: Open addressing

Open addressing: the colliding item is placed in a different cell

- **Linear probing**: handles collisions by placing the colliding item in the next (circularly) available table cell
- **Double hashing**: uses a secondary hash function d(k) and handles collisions by placing an item in the first available cell of the series $(h(k) + jd(k)) \mod N$ for $j = 0, \ldots, N-1$

Example for Linear probing:

- $h(x) = x \mod 13$
- Insert keys 18, 41, 22, 44, 59, 32, 31, 73, in this order



$59 \mod 13 = 7$

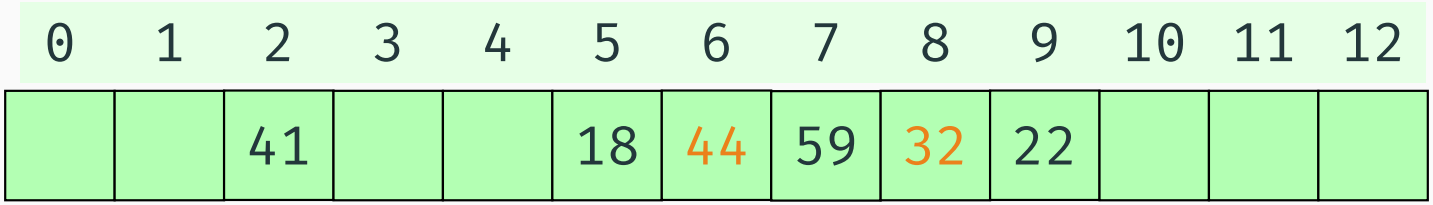
Collision Handling: Open addressing

Open addressing: the colliding item is placed in a different cell

- **Linear probing**: handles collisions by placing the colliding item in the next (circularly) available table cell
- **Double hashing**: uses a secondary hash function d(k) and handles collisions by placing an item in the first available cell of the series $(h(k) + jd(k)) \mod N$ for $j = 0, \ldots, N-1$

Example for Linear probing:

- $h(x) = x \mod 13$
- Insert keys 18, 41, 22, 44, 59, 32, 31, 73, in this order



$32 \mod 13 = 6$

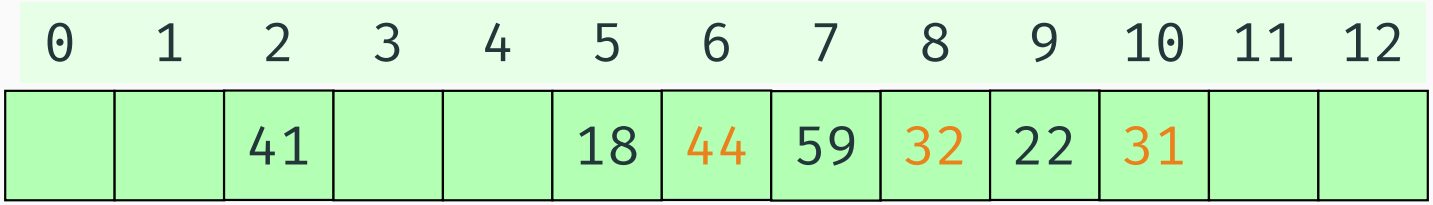
Collision Handling: Open addressing

Open addressing: the colliding item is placed in a different cell

- **Linear probing**: handles collisions by placing the colliding item in the next (circularly) available table cell
- **Double hashing**: uses a secondary hash function d(k) and handles collisions by placing an item in the first available cell of the series $(h(k) + jd(k)) \mod N$ for $j = 0, \ldots, N-1$

Example for Linear probing:

- $h(x) = x \mod 13$
- Insert keys 18, 41, 22, 44, 59, 32, 31, 73, in this order



$31 \mod 13 = 5$

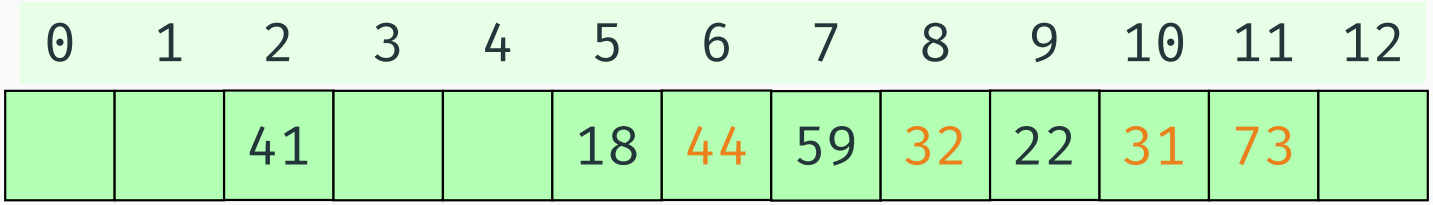
Collision Handling: Open addressing

Open addressing: the colliding item is placed in a different cell

- **Linear probing**: handles collisions by placing the colliding item in the next (circularly) available table cell
- **Double hashing**: uses a secondary hash function d(k) and handles collisions by placing an item in the first available cell of the series $(h(k) + jd(k)) \mod N$ for $j = 0, \ldots, N-1$

Example for Linear probing:

- $h(x) = x \mod 13$
- Insert keys 18, 41, 22, 44, 59, 32, 31, 73, in this order



$73 \mod 13 = 8$

Collision Handling: Open addressing

Open addressing: the colliding item is placed in a different cell

- **Linear probing**: handles collisions by placing the colliding item in the next (circularly) available table cell
- **Double hashing**: uses a secondary hash function d(k) and handles collisions by placing an item in the first available cell of the series $(h(k) + jd(k)) \mod N$ for $j = 0, \ldots, N-1$

Example for Linear probing: How about delete 44?

- $h(x) = x \mod 13$
- 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
		41			18	44	59	32	22	31	73	

Collision Handling: Open addressing

Open addressing: the colliding item is placed in a different cell

- **Linear probing**: handles collisions by placing the colliding item in the next (circularly) available table cell
- **Double hashing**: uses a secondary hash function d(k) and handles collisions by placing an item in the first available cell of the series $(h(k) + jd(k)) \mod N$ for $j = 0, \ldots, N-1$

Example for Linear probing: How about delete 44?

- $h(x) = x \mod 13$
- 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
		41			18	44	59	32	22	31	73	

Collision Handling: Open addressing

Open addressing: the colliding item is placed in a different cell

- **Linear probing**: handles collisions by placing the colliding item in the next (circularly) available table cell
- **Double hashing**: uses a secondary hash function d(k) and handles collisions by placing an item in the first available cell of the series $(h(k) + jd(k)) \mod N$ for $j = 0, \ldots, N-1$
- The secondary hash function d(k) cannot have zero values
- The table size N must be a prime to allow probing of all the cells

Common choice: $d(k) = q - k \mod q$, where $q < N$, q is a prime

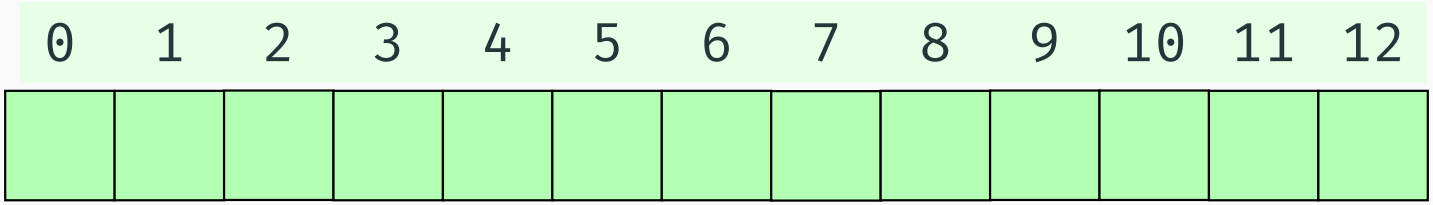
Collision Handling: Open addressing

Open addressing: the colliding item is placed in a different cell

- **Linear probing**: handles collisions by placing the colliding item in the next (circularly) available table cell
- **Double hashing**: uses a secondary hash function d(k) and handles collisions by placing an item in the first available cell of the series $(h(k) + jd(k)) \mod N$ for $j = 0, \ldots, N-1$

Example for Double Hashing:

- $N = 13, h(k) = k \mod 13, d(k) = 7 - k \mod 7$
- Insert keys 18, 41, 22, 44, 59, 32, 31, 73, in this order



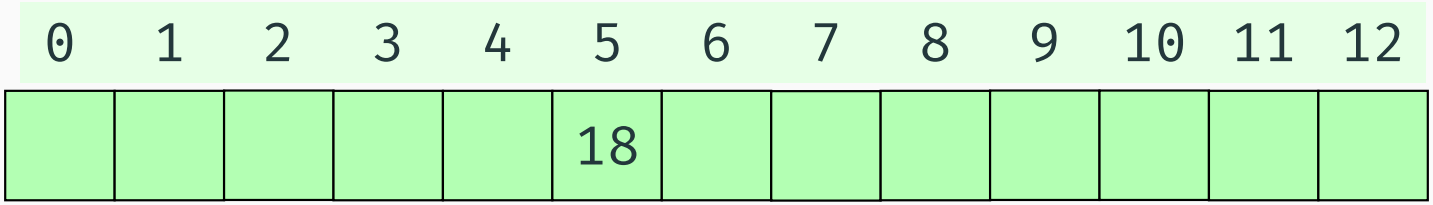
Collision Handling: Open addressing

Open addressing: the colliding item is placed in a different cell

- **Linear probing**: handles collisions by placing the colliding item in the next (circularly) available table cell
- **Double hashing**: uses a secondary hash function d(k) and handles collisions by placing an item in the first available cell of the series $(h(k) + jd(k)) \mod N$ for $j = 0, \ldots, N-1$

Example for Double Hashing:

- $N = 13, h(k) = k \mod 13, d(k) = 7 - k \mod 7$
- Insert keys 18, 41, 22, 44, 59, 32, 31, 73, in this order



$18 \mod 13 = 5, d(18) = 3$

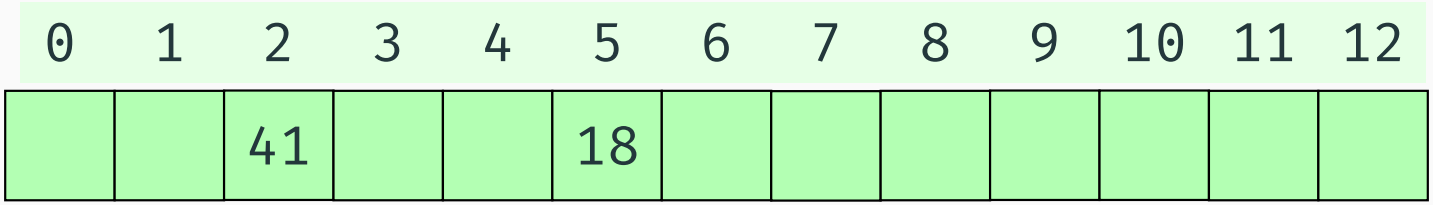
Collision Handling: Open addressing

Open addressing: the colliding item is placed in a different cell

- **Linear probing**: handles collisions by placing the colliding item in the next (circularly) available table cell
- **Double hashing**: uses a secondary hash function d(k) and handles collisions by placing an item in the first available cell of the series $(h(k) + jd(k)) \mod N$ for $j = 0, \ldots, N-1$

Example for Double Hashing:

- $N = 13, h(k) = k \mod 13, d(k) = 7 - k \mod 7$
- Insert keys 18, 41, 22, 44, 59, 32, 31, 73, in this order



$41 \mod 13 = 2, d(41) = 1$

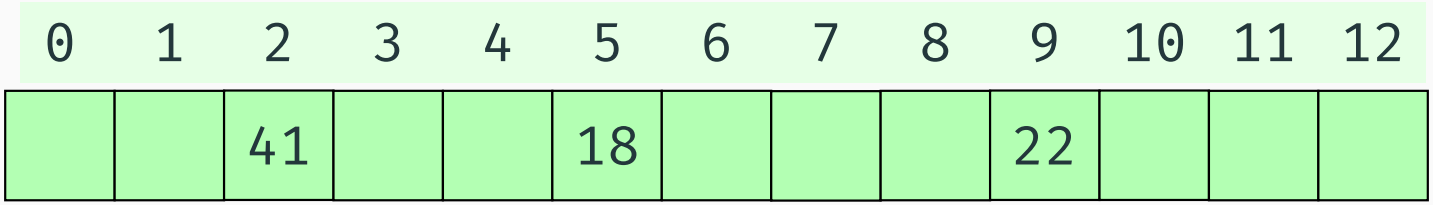
Collision Handling: Open addressing

Open addressing: the colliding item is placed in a different cell

- **Linear probing**: handles collisions by placing the colliding item in the next (circularly) available table cell
- **Double hashing**: uses a secondary hash function d(k) and handles collisions by placing an item in the first available cell of the series $(h(k) + jd(k)) \mod N$ for $j = 0, \ldots, N-1$

Example for Double Hashing:

- $N = 13, h(k) = k \mod 13, d(k) = 7 - k \mod 7$
- Insert keys 18, 41, 22, 44, 59, 32, 31, 73, in this order



$22 \mod 13 = 9, d(22) = 6$

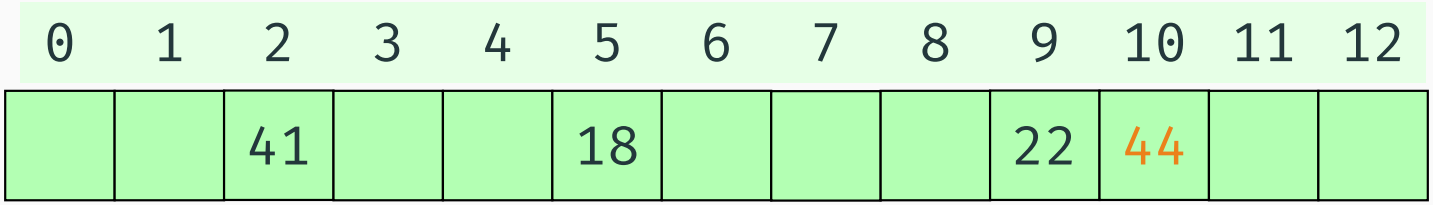
Collision Handling: Open addressing

Open addressing: the colliding item is placed in a different cell

- **Linear probing**: handles collisions by placing the colliding item in the next (circularly) available table cell
- **Double hashing**: uses a secondary hash function d(k) and handles collisions by placing an item in the first available cell of the series $(h(k) + jd(k)) \mod N$ for $j = 0, \ldots, N-1$

Example for Double Hashing:

- $N = 13, h(k) = k \mod 13, d(k) = 7 - k \mod 7$
- Insert keys 18, 41, 22, 44, 59, 32, 31, 73, in this order



$44 \mod 13 = 5, d(44) = 5$

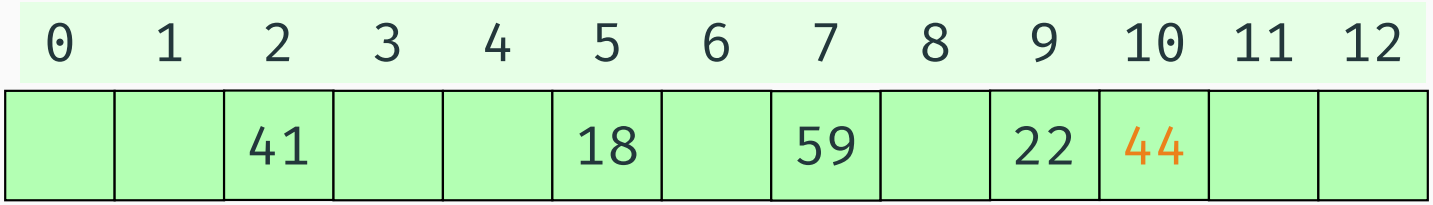
Collision Handling: Open addressing

Open addressing: the colliding item is placed in a different cell

- **Linear probing**: handles collisions by placing the colliding item in the next (circularly) available table cell
- **Double hashing**: uses a secondary hash function d(k) and handles collisions by placing an item in the first available cell of the series $(h(k) + jd(k)) \mod N$ for $j = 0, \ldots, N-1$

Example for Double Hashing:

- $N = 13, h(k) = k \mod 13, d(k) = 7 - k \mod 7$
- Insert keys 18, 41, 22, 44, 59, 32, 31, 73, in this order



$59 \mod 13 = 7, d(59) = 4$

Collision Handling: Open addressing

Open addressing: the colliding item is placed in a different cell

- **Linear probing**: handles collisions by placing the colliding item in the next (circularly) available table cell
- **Double hashing**: uses a secondary hash function d(k) and handles collisions by placing an item in the first available cell of the series $(h(k) + jd(k)) \mod N$ for $j = 0, \ldots, N-1$

Example for Double Hashing:

- $N = 13, h(k) = k \mod 13, d(k) = 7 - k \mod 7$
- 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
		41			18	32	59		22	44		

$32 \mod 13 = 6, d(32) = 3$

Collision Handling: Open addressing

Open addressing: the colliding item is placed in a different cell

- **Linear probing**: handles collisions by placing the colliding item in the next (circularly) available table cell
- **Double hashing**: uses a secondary hash function d(k) and handles collisions by placing an item in the first available cell of the series $(h(k) + jd(k)) \mod N$ for $j = 0, \ldots, N-1$

Example for Double Hashing:

- $N = 13, h(k) = k \mod 13, d(k) = 7 - k \mod 7$
- 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
31		41			18	32	59		22	44		

$31 \mod 13 = 5, d(31) = 4$

Collision Handling: Open addressing

Open addressing: the colliding item is placed in a different cell

- **Linear probing**: handles collisions by placing the colliding item in the next (circularly) available table cell
- **Double hashing**: uses a secondary hash function d(k) and handles collisions by placing an item in the first available cell of the series $(h(k) + jd(k)) \mod N$ for $j = 0, \ldots, N-1$

Example for Double Hashing:

- $N = 13, h(k) = k \mod 13, d(k) = 7 - k \mod 7$
- 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
31		41			18	32	59	73	22	44		

$73 \mod 13 = 8, d(73) = 4$

Performance of Hashing

- Worst case: searches, insertions and removals take $O(n)$ time
- The worst case occurs when all the keys collide
- The load factor $\alpha = n/N$ affects the performance of a hash table
- Assuming that the hash values are like random numbers, it can be shown that the expected number of probes for an insertion with open addressing is $1/(1-\alpha)$

The expected running time of all the map ADT operations in a hash table is $O(1)$

In practice, hashing is very fast provided $\alpha << 100\%$

Thank you!

Questions?