MAKE SCHOOL

Hash Tables

CS 1.3 - Core Data Structures

MAKE SCHOOL

Linear Data Structures

MAKE SCHOOL

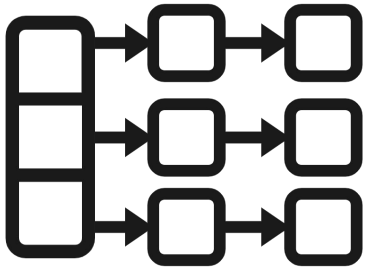
Array **Linked List** **Stack** **Queue**

A linear data structure has a specific order or sequence of its elements. There is a first element and a last element (or a top and a bottom).

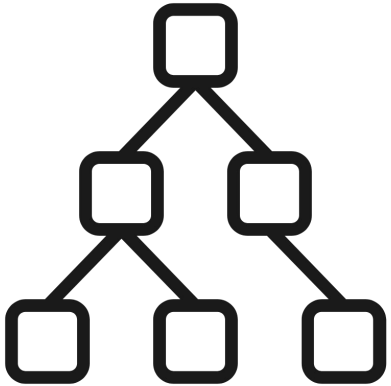
Definition from TutorialsPoint

MAKE SCHOOL

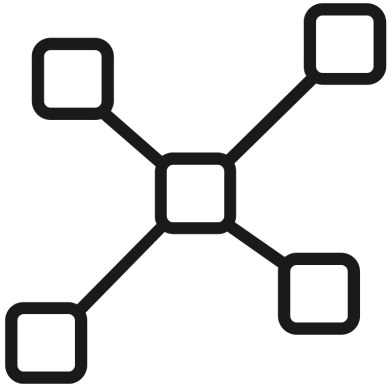
Non-Linear Data Structures



Hash Table



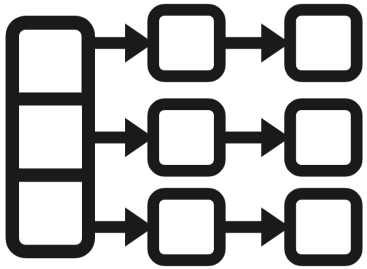
Tree



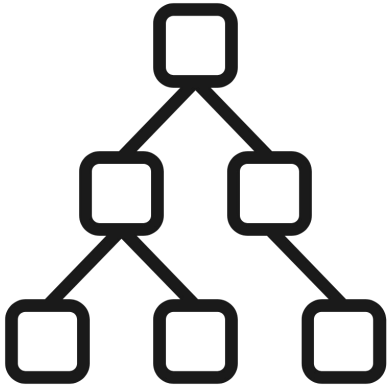
Graph

A non-linear data structure has no specific order or sequence of elements. Some structures can have multiple paths to connect to other elements.

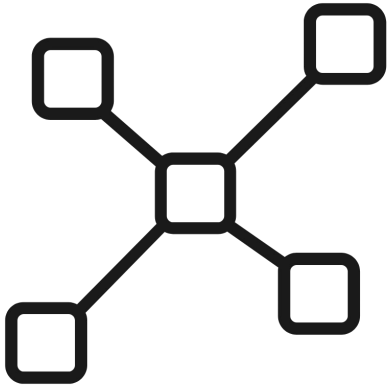
Definition from TutorialsPoint



Hash Table



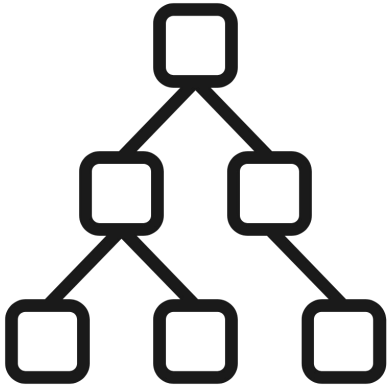
Tree



Graph

Non-linear data structures can support multi-level storage and often cannot be traversed in single run.

Definition from TutorialsPoint



Hash Table

Tree

Graph

For example, try tracing the 3 different paths we can follow from the top level to the bottom level of the tree above.

Students, draw anywhere on this slide!

What are examples of apps or technologies you have used that might use a non-linear data structure to organize their data?

MAKE SCHOOL

Remember Dictionaries? 👋

```
capitals = {“Argentina” : “Buenos Aires”,
            “Bolivia”   : “La Paz”,
            “Brazil”    : “Brasília” }
```

A dictionary data type is a collection which is unordered and mutable (changeable).

```
capitals = {“Argentina” : “Buenos Aires”,
            “Bolivia”   : “La Paz”,
            “Brazil”    : “Brasília” }
```

It is a set of key and value pairs.

key	value
“Paterson”	145710
“Jersey City”	261940
“Camden”	74002
“Atlantic City”	37999
“Newark”	281054

key	value
“Paterson”	145710
“Jersey City”	261940
“Camden”	74002
“Atlantic City”	37999
“Newark”	**281054**

“What is the population (value) of city (key) Newark?”

Lookup Operation

key	value
“Paterson”	145710
“Jersey City”	261940
“Camden”	74002
“Atlantic City”	37999
“Newark”	281054

“Irvington”	54233

“Add the city of Irvington and its 54,233 population.”

Insert Operation

key	value
“Paterson”	145710
“Jersey City”	261940
“Atlantic City”	37999
“Newark”	281054

“Camden”	74002

“Delete the city of Camden and its 74,002 population.”

Delete Operation

```
inventory = {“Jordan” : 1,
             “Yeezy”    : 8,
             “Air Max”  : 10 }
```

In Python, an easy way to create dictionaries using curly braces {}.

MAKE SCHOOL

Hash Tables

MAKE SCHOOL

What is a hash table?

A hash table is a data structure that maps **keys** to **values**.

A value can be **retrieved** from the hash table using its key.

Book metaphor

Book pages = array of buckets

Book index = hash function

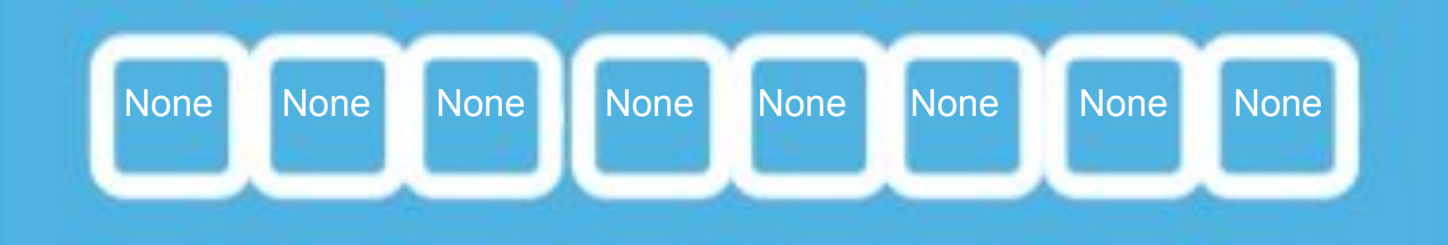
MAKE SCHOOL

How to Build a Hash Table:

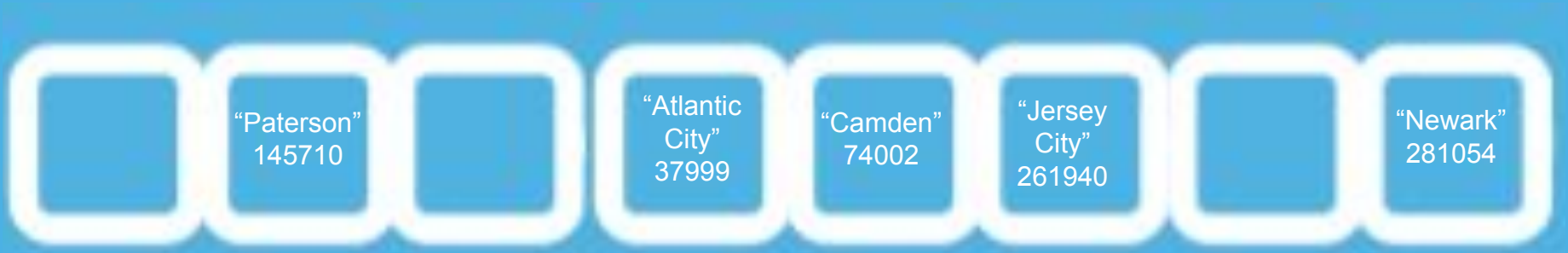
1. Create an array. Because we do not use arrays in Python, we create a list with "`None`" as the placeholder for each element.

   For example, when creating a Hash Table of size 8:

```
[None] * 8
```



key	value
“Paterson”	145710
“Jersey City”	261940
“Camden”	74002
“Atlantic City”	37999
“Newark”	281054



How to Build a Hash Table:

2️⃣ Create a **hash function** that turns each key into a number (called a **hash code**) that we can use to decide where in our array of buckets each key → value pair should be stored.

For example, we could make a hash function that looks at the **first letter** of each key and calculates how many characters away it is from the letter 'A'. Here are some examples: `hash("Apple") → 0`, `hash("Fig") → 5`, `hash("Kiwi") → 10`

For example:

Let's find the **hash code** for "Paterson"

"Paterson" starts with P, which is 15 characters away from the letter 'A'.

So we should try to store the pair "Paterson":145710 in **bucket 15** in our hash table… 👀🤞😬

key	value
"Paterson"	145710
"Jersey City"	261940
"Camden"	74002
"Atlantic City"	37999
"Newark"	281054

But wait… our Hash Table only has 8 buckets. Meaning it only has buckets with indexes ranging from index 0 to 7.

What can we do to change the value 15 to be within the range of 0 to 7?

MAKE SCHOOL

15 mod 8 = 7

We could use the mod operator.

Check for Understanding

Compute the index for the key:value pairs in the table use the hash function we discussed on the previous slide.

key	value	bucket
“Paterson”	145710	7
“Jersey City”	261940	
“Camden”	74002	
“Atlantic City”	37999	
“Newark”	281054	

So far we know...

To create a Hash Table we need:

1. Data in key → value pairs.
2. An array of buckets with a specific fixed size.
3. A hash function to determine where each key value pair will be stored in the array of buckets.

MAKE SCHOOL

Hash Collision

For example:

Let’s find an index for “Paterson.”

“**Paterson**” starts with P, which is 15 characters away from the letter A.

But what if we want to add “Princeton” to our hash table...

“**Princeton**” starts with P, which is 15 characters away from the letter A.

key	value
“Paterson”	145710
“Princeton”	31000
“Camden”	74002
“Atlantic City”	37999
“Newark”	281054

Check for Understanding

MAKE SCHOOL

Compute the bucket for the key:value pairs in the table use the hash function we discussed on the previous slide.

key	value	bucket
“Paterson”	145710	7
“Princeton”	31000	7
“Moorestown”	20355	4
“Montclair”	38676	4

MAKE SCHOOL

What is a hash collision?

Students, write your response!

MAKE SCHOOL

What is a hash collision?

A **hash collision** is when multiple keys land in the **same bucket** in the array. When determining which hash function to use for your data, you should avoid collisions to keep your hash table efficient.

🔗 **Example**: DJB2

MAKE SCHOOL

Handling Collision

Methods of Handling Collision

MAKE SCHOOL

1. Linear Probing
2. Chaining
3. Double Hashing (not covered in CS 1.3)

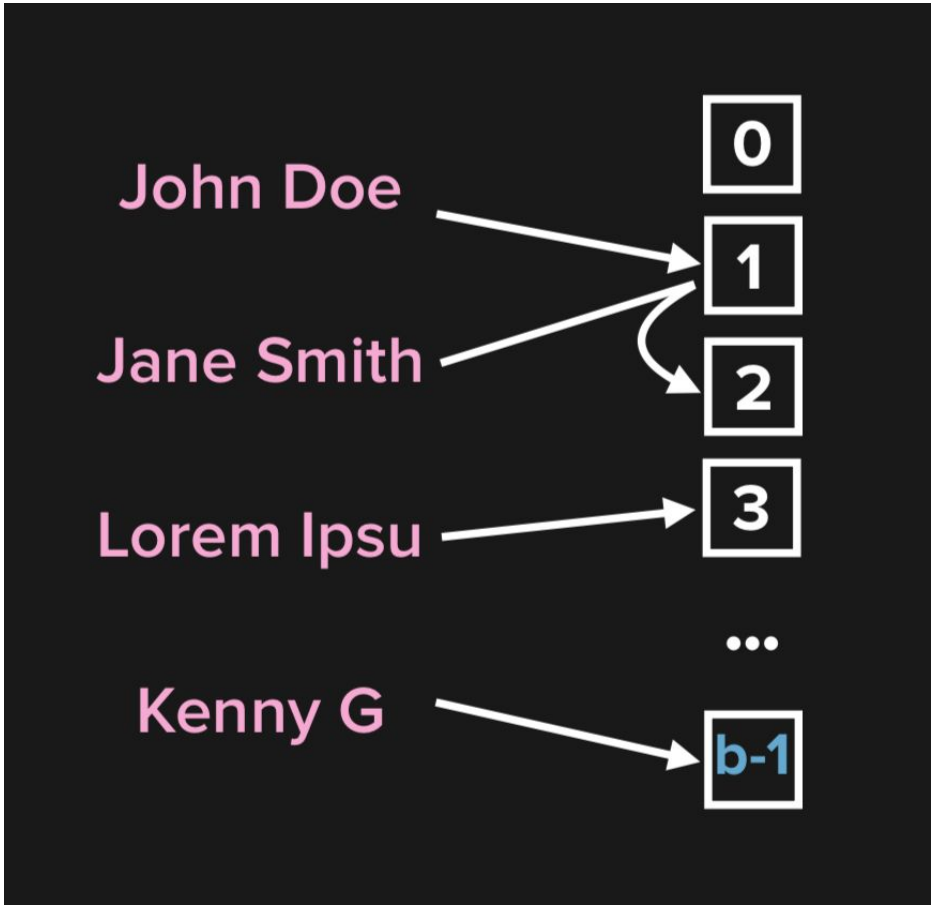
Linear Probing

Each bucket contains at most one entry

On collision - find next open bucket, add new entry there

To retrieve - find bucket, if that's not entry, try next bucket until you find entry or empty bucket

Python's `dict` uses probing



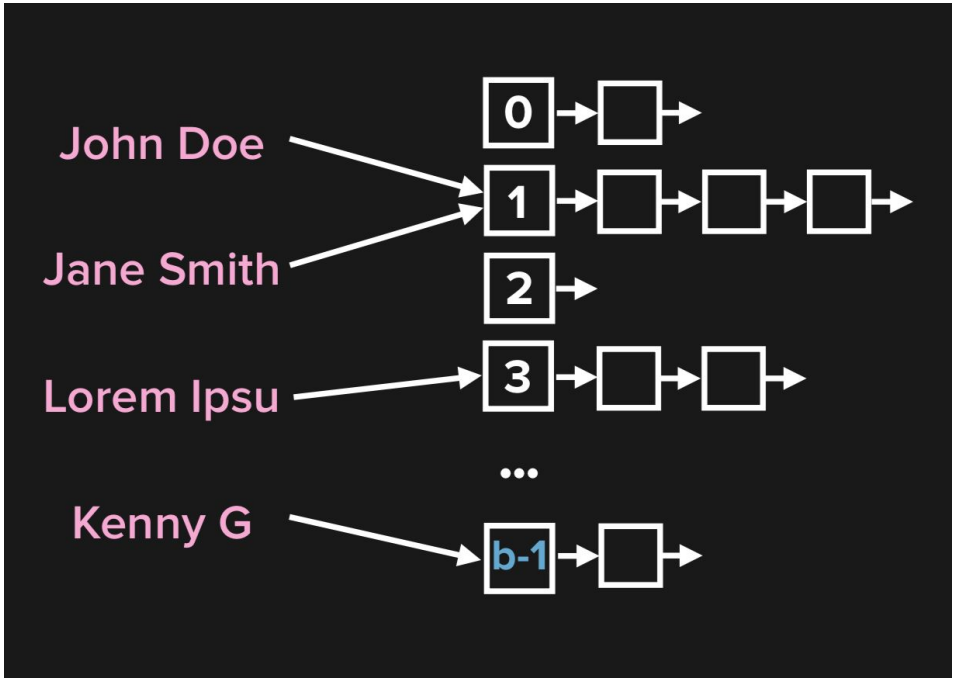
Chaining

Each bucket contains a **linked list** of entries

On collision - add new entry to the bucket's linked list

To retrieve - find bucket, find entry in linked list

We will use chaining with linked lists to implement our hash table



MAKE SCHOOL

Let’s Draw a Hash Table

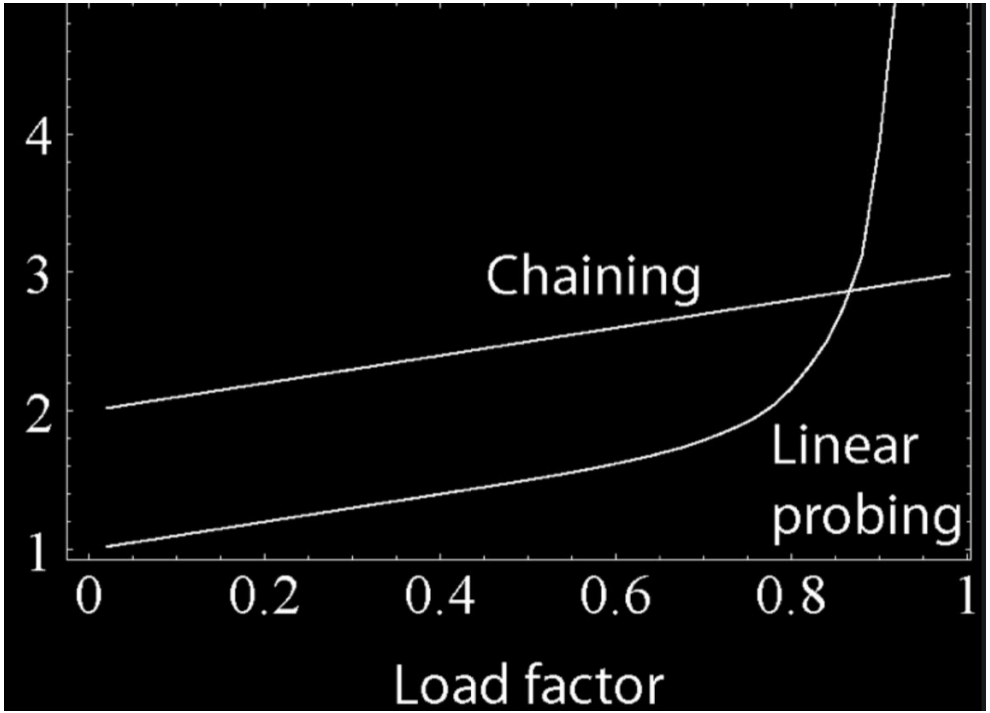
Load Factor

Load Factor = entries / buckets

Load Factor affects performance

Collision Resolution method also

affects performance



Resizing

MAKE SCHOOL

Once the load factor reaches a certain threshold and begins to impact performance we need to resize the hash table

1. Store what's already in our hash table
2. Make a new hash table with twice the number of buckets
3. Rehash our stored items

Amortized Time Complexity

MAKE SCHOOL

“Amortized time is the way to express the time complexity when an algorithm has the very bad time complexity only once in a while besides the time complexity that happens most of time”

O(1)*

https://medium.com/@satorusasozaki/amortized-time-in-the-time-complexity-of-an-algorithm-6dd9a5d38045

Time Complexity Analysis Review

MAKE SCHOOL

- items
- get
- set
- delete

Hash table with chaining animation

Time Complexity Analysis Review

MAKE SCHOOL

Operation	Amortized	Worst
items()	O(n)	O(n)
get()	O(1)	O(n)
set()	O(1)	O(n)
delete()	O(1)	O(n)

MAKE SCHOOL

Module 5: Hash Tables