Welcome!

Let’s know each other

My name is Tahiya Chowdhury. I use she/her pronouns. I will be facilitating this lecture.

Your turn

Let's go around the room and say

- Your name
- Preferred pronoun
- Quick fun fact about yourself (optional)

Agenda for today

- Lecture topic of the day: Hash Tables
- In class Activity
- Exercise problems (to be solved as Homework or Out of class activity)

Note.

All materials used in this lecture is available at this link:
https://github.com/Tahiya31/demo_lecture

This notebook has several **Try** sections that we will try during class as in-class activity.

Scenario

We all use passwords for across to different platforms.

Have you ever forgot a password of a website?

When you tried to recover it, the site sends you a link to reset it.

Scenario

We all use passwords for across to different platforms.

Have you ever forgot a password of a website?

When you tried to recover it, the site sends you a link to reset it.

Have you wondered why?

Scenario

We all use passwords for across to different platforms.

Have you ever forgot a password of a website?

When you tried to recover it, the site sends you a link to reset it.

Have you wondered why?

Answer

- The website doesn’t store the entire original password you chose.
- Just stores its hash, a transformed version of it.
- If the site’s database gets hacked, the hackers won’t be able to know your password but just the hash of your password.
- As hashes are one-way function, there is no way to trace back to your actual password.

How are passwords stored in a database?

If you store the (user_name, password) pairs to a database as they came along, every time you need to retrieve one, you will need to go through each entry, one at a time.

One way to avoid this is to store all our user names and passwords in a structured way.

So that you can retrieve it without having to go through each of them.

Let's assume

We are creating a website where we have multiple users with unique passwords.

Toy collection of stored login names and passwords on our website

Login Name	Password
gatsby	qazwsx
panoroma	qwerty
wabanaki	password1
saturn	1234567
diatom	987654

Let's assume

We are creating a website where we have multiple users with unique passwords.

Toy collection of stored login names and passwords on our website

Login Name	Password
gatsby	qazwsx
panoroma	qwerty
wabanaki	password1
saturn	1234567
diatom	987654

If User saturn wants to retrieve their password, how will the system retrieve it?

What kind of problem is this?

What kind of problem is this?

This is a search problem, where we want to quickly search and retrieve information.

How do we search for password of a specific user in this table?

Notice that this is not sorted.

If the elements were ordered, we could apply binary search resource 1 2

and search the user in **logarithmic time**.

How do we search for password of a specific user in this table?

Notice that this is not sorted.

If the elements were ordered, we could apply binary search resource 1 2

and search the user in **logarithmic time**.

Note

Here logarithmic time represents the running time of binary search given an input . For more on Big-O notations and run time complexity, check here.

How can we make this more efficient (faster)?

Have more information

about where the password might be located in the table

Have more information

about where the password might be located in the table

Option 1

You know exactly where to look

- Knowing the position of the element in an array of 100 elements allows us to quickly find the element using its index (array position). *(time complexity)*.

Have more information

about where the password might be located in the table

Option 1

You know exactly where to look

- Knowing the position of the element in an array of 100 elements allows us to quickly find the element using its index (array position). *(time complexity)*.

Option 2

You can structure the table in a way so that you don't need to go through each entry.

- Sorting an array allows us to find it without having to go through each element one by one. *(time complexity)*.

But what happens when we don't have location (or position) information?

How do we know the position?

That’s where hash table comes!

Key terms for today

- Hash Table
- Hash Function: Division Method
- Collision and Collision Resolution: Open Addressing, Linear Probing, Quadratic probing
- Chaining
- Big O, run time complexity

Hash Table

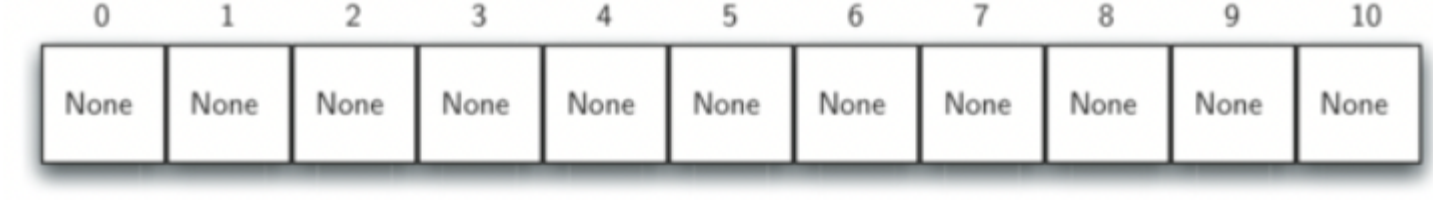
Hash Table

- Hash Table is a data structure that allows us to quickly search for an item.
- Think of a collection of items which are stored in such a way to make it easy to find them later.

- Each position of the hash table, often called a slot, can hold an item and is named by an integer value starting at 0

  Let's call this name **index**.

  For example, we will have a slot indexed 0, a slot indexed 1, a slot indexed 2, and so on. Initially, the hash table contains no items so every slot is empty.



This is a hash table of size $m = 11$

When an item needs to be placed in the table, we will need to find a slot for it.

The mapping between an item and the slot where that item belongs in the hash table is called the **hash function**.

Hash function

In a hash table, given some key, we apply a **hash function** to the key to find the index of the slot where the corresponding item would go.

If we have m slots, our hash function:

- will take any item in our collection
- return its index as an integer

This index tells us the slot name, and its range is between 0 and $m - 1$.

How can we design a hash function?

A hash function that maps each item to a unique slot is a **perfect hash function**.

Unfortunately there is no way to design a perfect hash function as

- items in the collection change is likely to change over time.
- will need to increase size of the hash table to accomodate each new item.
- waste of memory.

How can we design a hash function?

A hash function that maps each item to a unique slot is a **perfect hash function**.

Unfortunately there is no way to design a perfect hash function as

- items in the collection change is likely to change over time.
- will need to increase size of the hash table to accomodate each new item.
- waste of memory.

Okay, we accept that accomodating one item per slot is not feasible.

Then a practical solution will be to allow multiple items per slot if needed.

Goal of hash function design

- distribute the items evenly across different slots.
- easy to compute hash function
- Designing good hash function is hard.

Designing Hash Function

Strategy 1: Division Method (or Remainder Method)

Steps:

1. We pick a collection of items.
2. Our hash function h, will take one item at a time and divide it by the number of slots.
3. The remainder from the division will be the hash value and will determine the slot number.

Hash Function $h(item) = item\%11$

Now You Try!

Consider the following table. Our hash table has 11 slots.

Using Remainder method, can we get the hash values for these items?

Item	Hash Value (slot number)
26	
35	
27	
17	
42	
70	
57	
45	
86	
4	

Once the hash values are calculated

Let's put them in our hash table.

index	item
0	

Once the hash values are calculated

Let's put them in our hash table.

index	item
0	

Once we have the hash table filled with data, let's go back to our search problem.

How do we now find an item in this collection (hash table)?

Answer

Apply hash function to the item and we will get the index.

The rest is like finding an element in an array using index with time complexity.

Answer

Apply hash function to the item and we will get the index.

The rest is like finding an element in an array using index with time complexity.

Let's review chapter 6.5.1 to learn about a couple more hash functions such as *folding method*, *mid square method*, etc.

But ...

What if we have multiple items whose hash value is same and thus are assigned to the same slot in the hash table?

When two or more items are assigned to the same slot, it's called ..

Collision

How to deal with collision: Collision Resolution

When a second item gets assigned to a slot where an item already exists, we need to find a way to assign the second item in the table.

Collision Resolution Strategy 1: Open Addressing

What is the best way to get a spot to sit in a crowded coffee shop?

Answer

You find an available chair and just sit.

Steps

1. Start at the index we get from the hash function.
2. Is there a value already? Yes: then move to the next index (slots) in a sequential manner.
3. Continue until am empty slot is found.

Note

This may require going through the entire collection and then coming back to the starting index in a circle.

Open addressing tries to accommodate items by finding next open slot that is available.

Since we are checking one slot at a time, it is called **linear probing**.

Now you try!

This hash table has several slots where multplie items assigned. Let's use linear probing to find new slots for these items. Use the table below.

Item	Hash Value (slot number)	Collision Resolution with Linear Probing
26		
35		
27		
17		
42		
70		
57		
45		
86		
4		

How do we search for items in such tables?

Just like steps we used to build the table, we will use the same for search in the table.

Steps:

1. Start at the index we get from the hash function.
2. Is the value there? Yes, that is great.
3. Is the value there? No: then continue to the next slot until the value is found.

Disadvantage of Linear Probing

Clustering

We can probably see what will happen if a collision occurs at the same slot for multiple items,

all the slots near that specific slot will be filled eventually.

This will result in clustering where several slots after/near the occupied slot will start getting occupied in a row.

Chaining: Bonus method for resolving collision

If many items gets assigned to the same index location in hash table, then collision occurs.

Chaining will form a chain of all the items for that location and allow them to exist despite the collision.

Disadvantage.

- If there are many items that have been assigned to the same location, finding a specific item in that table becomes harder.
- We will first get to the location, but instead of an item, we will get a collection.
- For that collection we have to use separate search method to see whether that item is present in that collection.

Note

For a hash table we can calculate a metric called load factor, defined
$\lambda = numberofitems/tablesize.$

This convey how much chance we have for getting a collision.

- If λ is small, there is lower chance of collision.
- If λ is large, the table is getting filled up, which means higher chnace for collision..

Review

What did we learn today?

Can we think of more applications of hash table?

- For driver's license record's. Get information about the driver (ie. name, address, age) given the licence number.
- For internet search engines.
- For electronic library catalogs. Hash Table implementations allow for a fast find among the millions of materials stored in the library.
- For implementing password database for systems with multiple users. Hash Tables allow for a fast retrieval of the password which corresponds to a given username.

Exercise time!

Problem 1

In a hash table of size 13 which index positions would the following key pairs will map to?

A. 1, 10

B. 13, 0

C. 1, 0

D. 2, 3

Exercise time!

Problem 1

In a hash table of size 13 which index positions would the following key pairs will map to?

A. 1, 10

B. 13, 0

C. 1, 0

D. 2, 3

Let's try!

Problem 2

Suppose you are given the following set of keys to insert into a hash table that holds exactly 11 values:

113 , 117 , 97 , 100 , 114 , 108 , 116 , 105 , 99.

Which of the following best demonstrates the contents of the hash table after all the keys have been inserted using **open addressing with linear probing**?

A. 100, __, __, 113, 114, 105, 116, 117, 97, 108, 99

B. 99, 100, __, 113, 114, __, 116, 117, 105, 97, 108

C. 100, 113, 117, 97, 14, 108, 116, 105, 99, __, __

D. 117, 114, 108, 116, 105, 99, __, __, 97, 100, 113

Problem 2

Suppose you are given the following set of keys to insert into a hash table that holds exactly 11 values:

113 , 117 , 97 , 100 , 114 , 108 , 116 , 105 , 99.

Which of the following best demonstrates the contents of the hash table after all the keys have been inserted using **open addressing with linear probing**?

A. 100, __, __, 113, 114, 105, 116, 117, 97, 108, 99

B. 99, 100, __, 113, 114, __, 116, 117, 105, 97, 108

C. 100, 113, 117, 97, 14, 108, 116, 105, 99, __, __

D. 117, 114, 108, 116, 105, 99, __, __, 97, 100, 113

Let's try!

Problem 3

Insert the keys 17, 3, 9, 39, 5, 6, 28, and 22 into a hash table of size 11 given using hash function h(x) = x mod 11. Use Chaining method.

Problem 3

Insert the keys 17, 3, 9, 39, 5, 6, 28, and 22 into a hash table of size 11 given using hash function h(x) = x mod 11. Use Chaining method.

Let's try!

Resources

1. Hashing, Data Structures with Python
2. Binary Search, Data Structures with Python

Resources

1. Hashing, Data Structures with Python
2. Binary Search, Data Structures with Python

Checklist

- Homework: Submit the exercises from this lecture to Moodle.

Resources

1. Hashing, Data Structures with Python
2. Binary Search, Data Structures with Python

Checklist

- Homework: Submit the exercises from this lecture to Moodle.

See you next week!