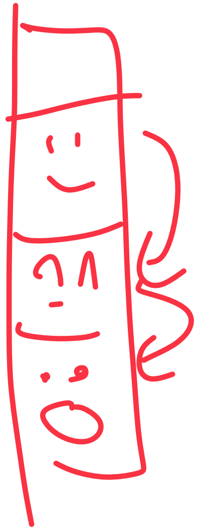
Algorithms and Data Structures for Data Science
Hashing 2

CS 277
Brad Solomon

April 10, 2024



UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Department of Computer Science

Learning Objectives

Review fundamentals of hash tables

Introduce closed hashing approaches to hash collisions

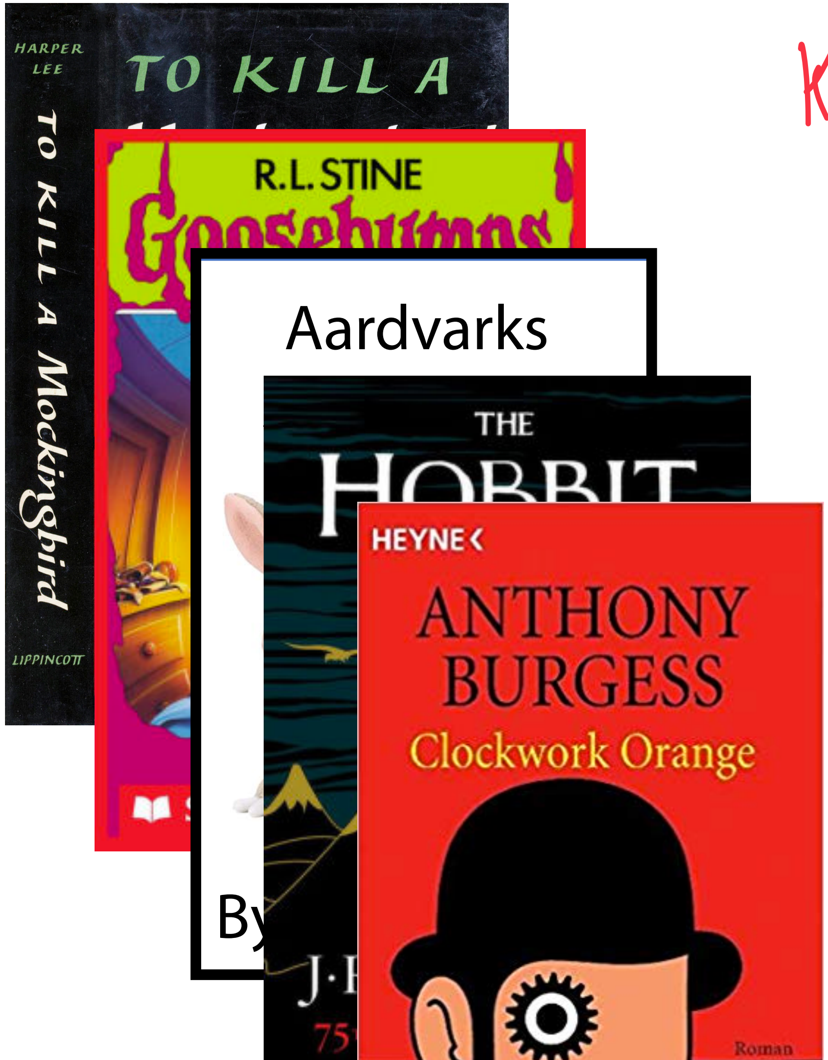
Determine when and how to resize a hash table

Data Structure Review

Data as key, value pairs is an extremely common format in data science

Key - search for

Value - what I return



A Hash Table based Dictionary

List [indexed by #]

```
d = {}
d[k] = v
print(d[k])
```

"List" [indexed by key]

A **Hash Table** consists of three things:

1. A hash function

↳ Key → integer

convert to

↳ hash value

hash index

dictionary[["A", ~~"B"~~]]

"C"

↳ ("A", "B")

2. A data storage structure

↳ Look up/Store data as a list

3. A method of addressing *hash collisions*

Trivia point about

lists vs tuples

Open vs Closed Hashing

Addressing hash collisions depends on your storage structure.

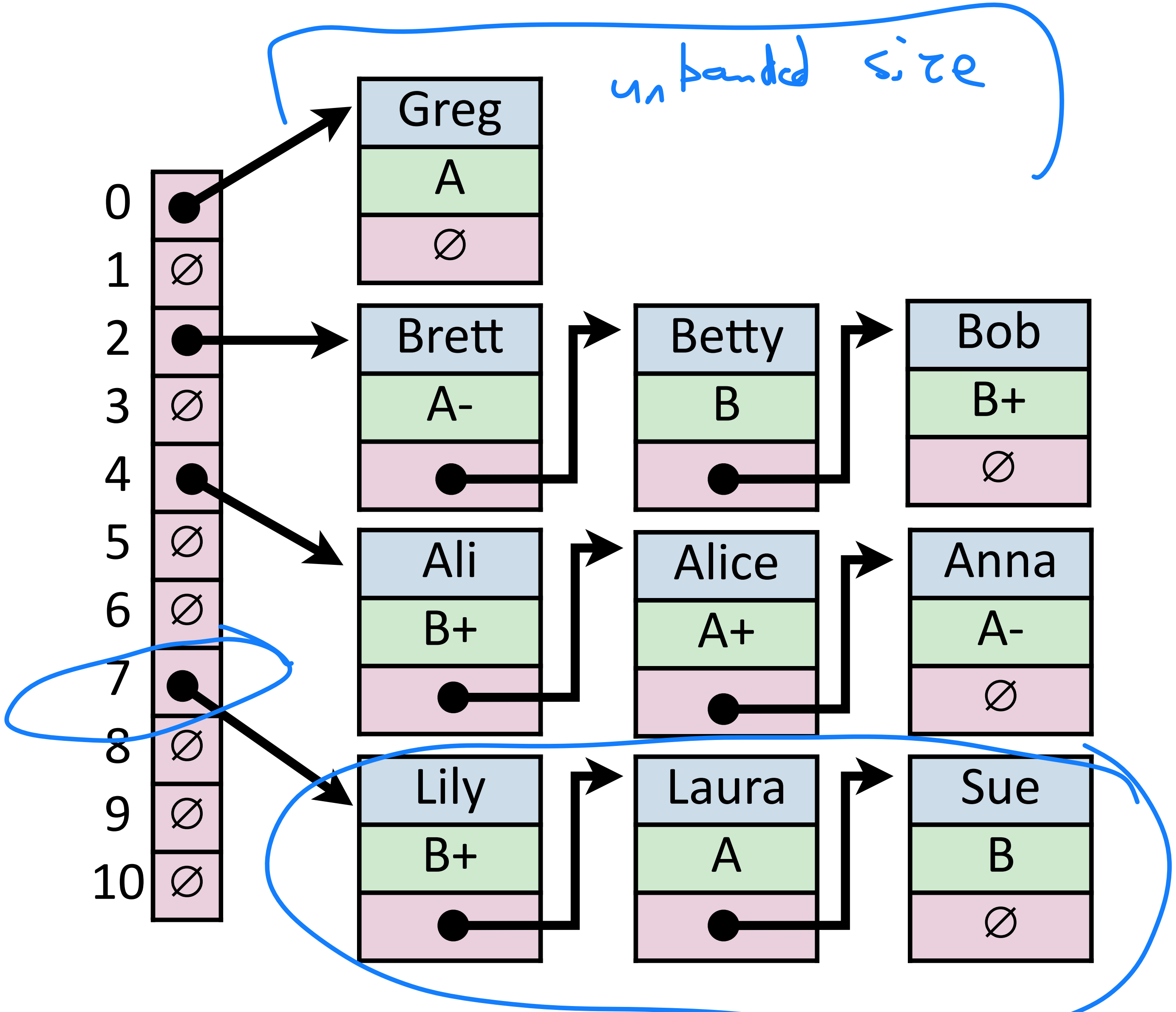
- **Open Hashing:** store *k,v* pairs externally

- **Closed Hashing:** store *k,v* pairs in the hash table

Hash Table (Separate Chaining)

Linked list = open hashing

Key	Value	Hash
Bob	B+	2
Anna	A-	4
Alice	A+	4
Betty	B	2
Brett	A-	2
Greg	A	0
Sue	B	7
Ali	B+	4
Laura	A	7
Lily	B+	7



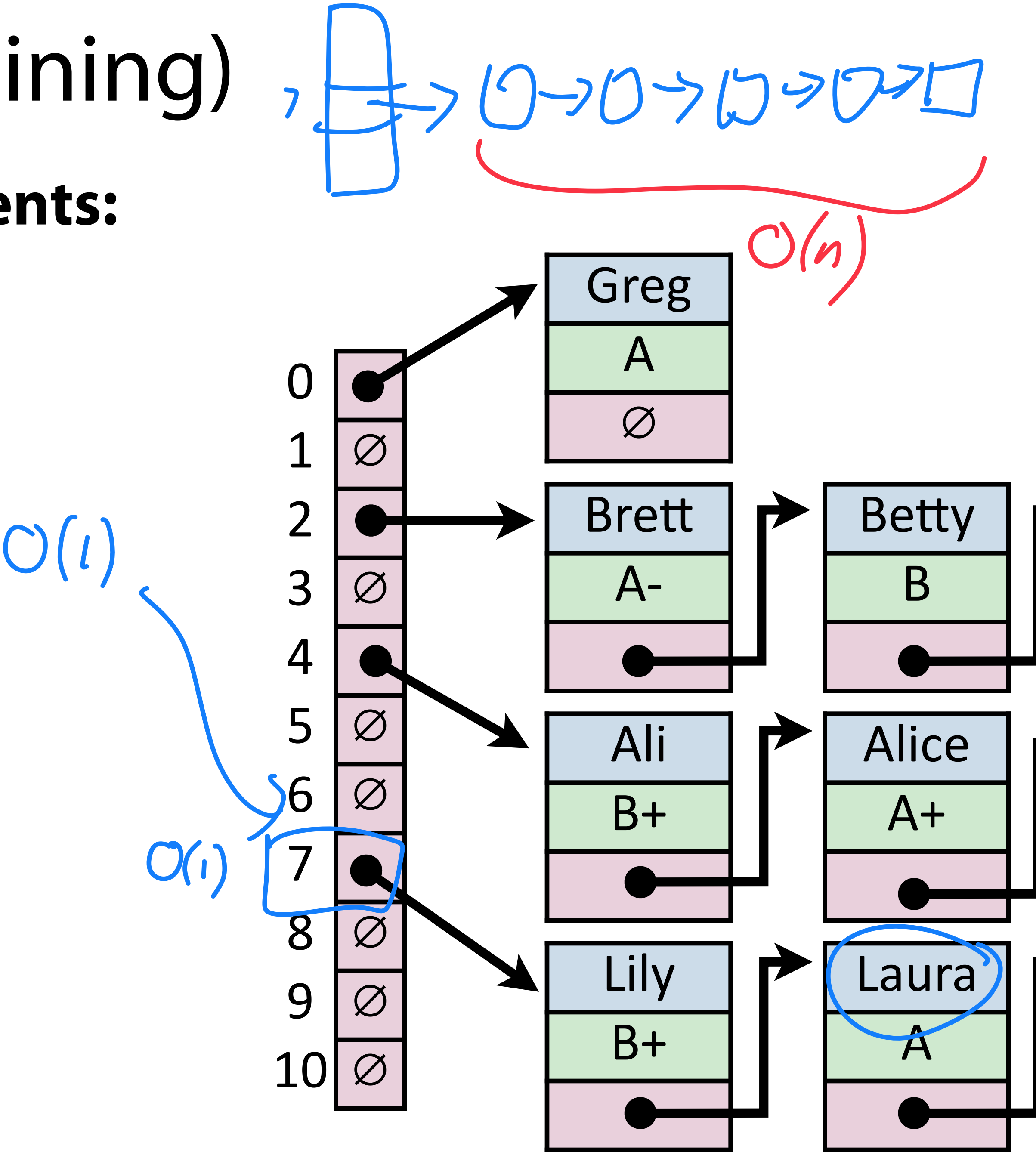
Hash Table (Separate Chaining)

For hash table of size *m* and *n* elements:

Find runs in: $O(n)$

Insert runs in: $O(1)$

Remove runs in: $O(n)$



Hash Table

Worst-Case behavior is bad — but what about randomness?

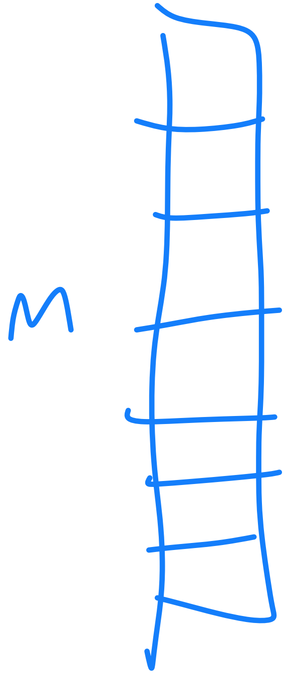
1) **Fix *h***, our hash, and assume it is good for ***all keys***:

Simple Uniform Hashing Assumption (SUHA)

Simple Uniform Hashing Assumption

Given table of size m, a simple uniform hash, h, implies

$$\forall k_1, k_2 \in U \text{ where } k_1 \neq k_2 ,\ Pr(h[k_1] = h[k_2]) = \boxed{\frac{1}{m}}$$



Uniform: keys are equally likely to hash to any position

↳ $\frac{1}{m}$ chance to be at each position

Independent: All keys hash independently of each other

Separate Chaining Under SUHA

Given table of size m and n inserted objects

Claim: Under SUHA, expected length of chain is $\frac{n}{m}$

linked lists

Separate Chaining Under SUHA

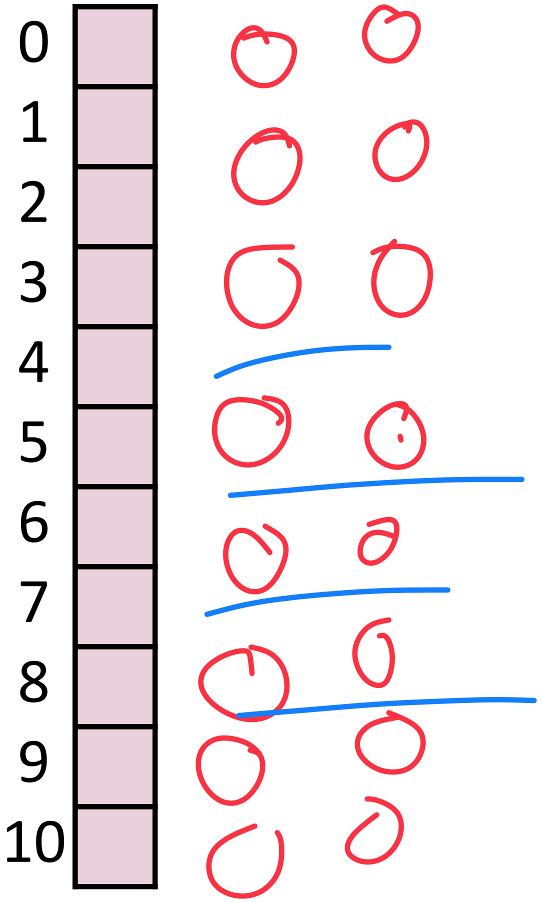
Under SUHA, a hash table of size *m* and *n* elements:

$\alpha = n/m$

Find runs in: $O(1+\alpha)$.

Insert runs in: $O(1)$.

Remove runs in: $O(1+\alpha)$.



(Example of closed hashing)

Collision Handling: Probe-based Hashing

S = { 1, 8 , 15}

h(k) = k % 7

|S| = n

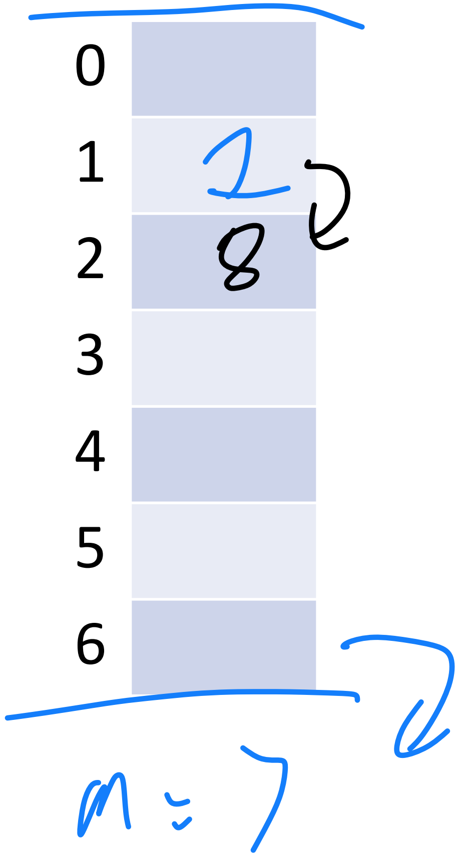
|Array| = m

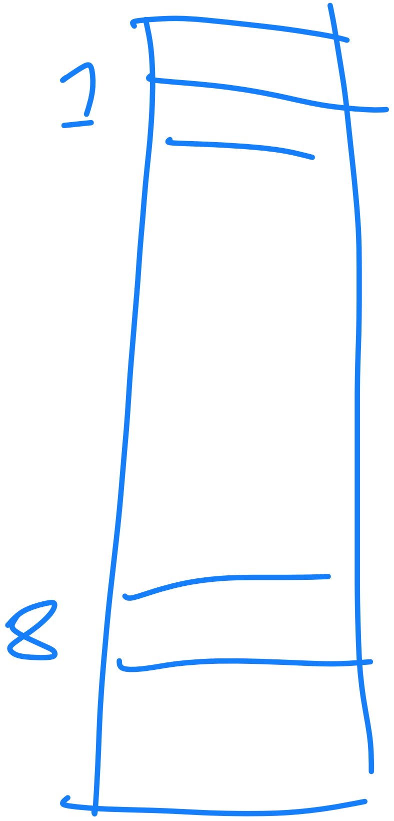
Resize if collide is a "perfect hash" solution m = 13

1 % 7 = 1

8 % 7 = 1

8 at the "next available space"



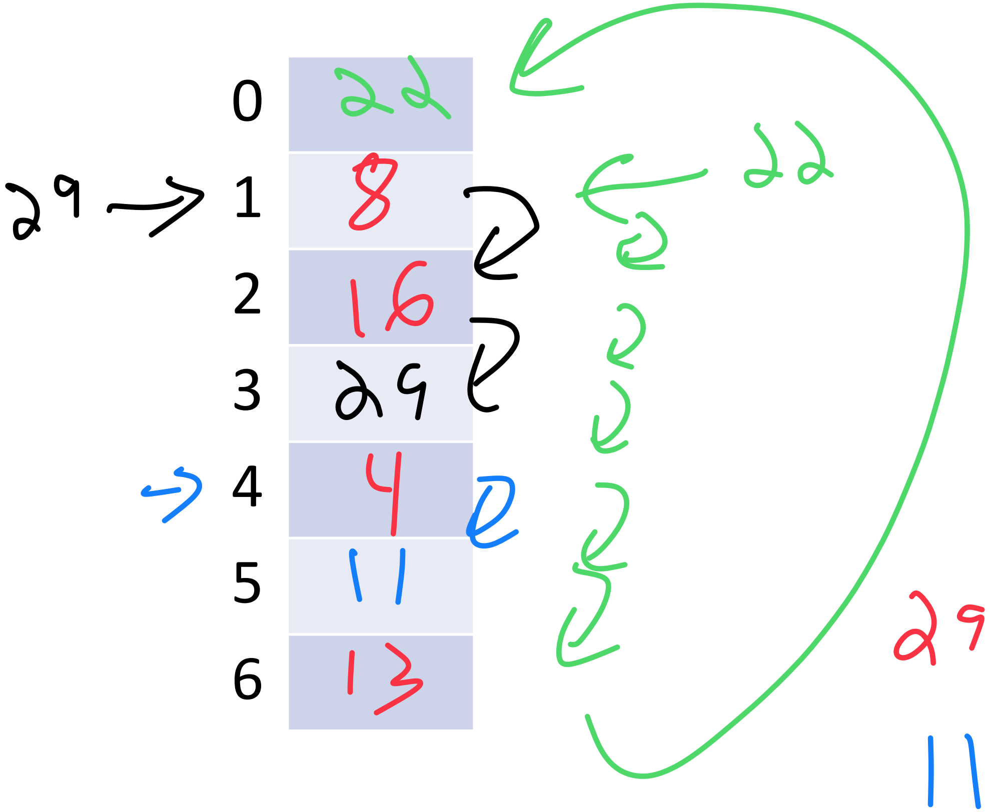


(Example of closed hashing)

Collision Handling: Linear Probing

S = { 16, 8, 4, 13, 29, 11, 22 } **|S| = n**

h(k) = k % 7 **|Array| = m**



0	22
1	8
2	16
3	29
4	4
5	11
6	13

h(k, i) = (k + i) % 7

Try h(k) = (k + 0) % 7, if full…

Try h(k) = (k + 1) % 7, if full…

Try h(k) = (k + 2) % 7, if full…

Try …

22 % 7 = 1

29 % 7 = 2

11 % 7 = 4

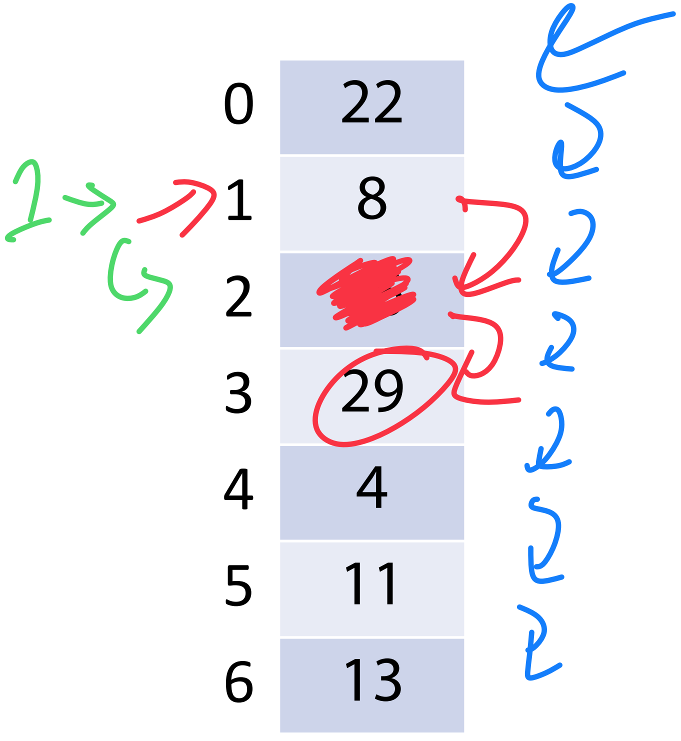
(Example of closed hashing)

Collision Handling: Linear Probing

Find(1)

S = { 16, 8, 4, 13, 29, 11, 22 } **|S| = n**

h(k, i) = (k + i) % 7 **|Array| = m**



0	22
1	8
2	[redacted]
3	29
4	4
5	11
6	13

_find(29)

1) Hash(29) h(29) = 1

2) Check all "next available spaces"

↳ stop when found value

↳ Stop when we have looked at every space

↳ stop if we see a blank space

_find(70)

70%7=0

(Example of closed hashing)

Collision Handling: Linear Probing

S = { 16, 8, 4, 13, 29, 11, 22 } **|S| = n**

h(k, i) = (k + i) % 7 **|Array| = m**

Tombstone		
0	← i → 0	22
0	1	8
1	2	9 ~~16~~
0	3	29
0	4	4
0	5	11
0	6	13

something once existed here

Thats blank but...

it wasnt always blank

↑ list of bools of size M

_remove(16)

1) hash key h(16) = 2

2) Look for 16 [find]

3) Remove 16

by rehashing everything but 16

or

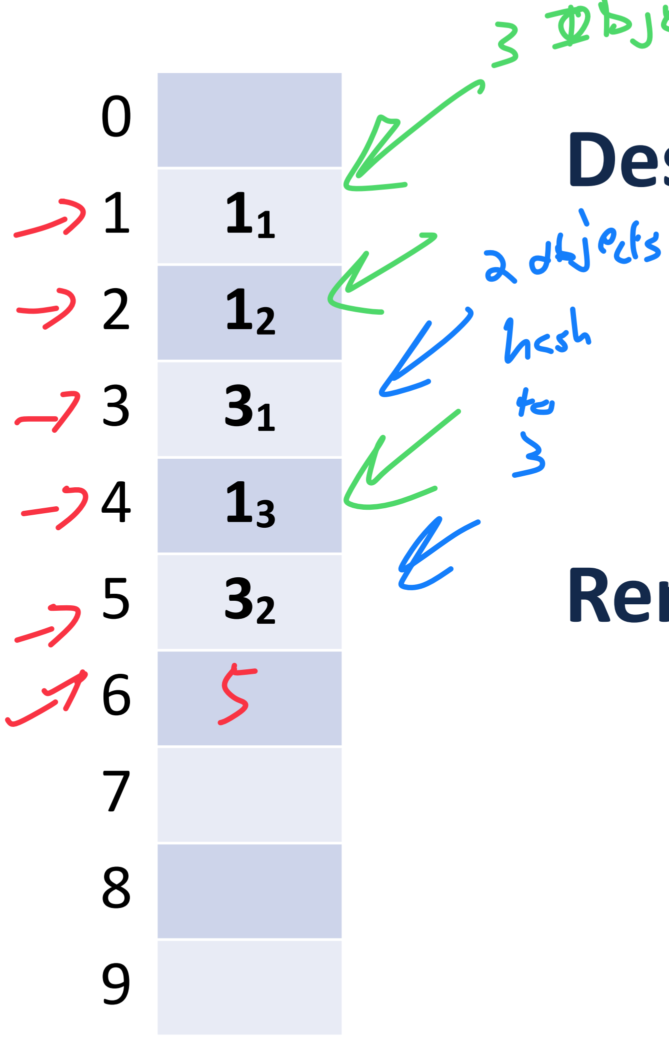
by adding a single tombstone bit

Find (29)

A Problem w/ Linear Probing

Primary clustering:

3 objects hash to 1



0	
1	1_1
2	1_2
3	3_1
4	1_3
5	3_2
6	5
7	
8	
9	

2 objects hash to 3

Description: Collisions create long runs of filled in positions

↳ Even with 1/m probability of hashing to any #, the next item has a 6/10 chance of being at index 6

Remedy:

↳ We need a better "next available"

(Example of closed hashing)

Collision Handling: Quadratic Probing

S = { 16, 8, 4, 13, 29, 12, 22 }

h(k) = k % 7

|S| = n

|Array| = m

22 ⇒ 1

1+1	2
1+4	5
1+9	1+9=10=3

0	12
1	8
2	16
3	22
4	4
5	29
6	13

29 → 1

12 → 5

1+1

1+4

5+1

(5+9)%7

(5+4)%7 = 2

h(k, i) = (k + i*i) % 7

Try h(k) = (k + 0) % 7, if full…

Try h(k) = (k + 1*1) % 7, if full…

Try h(k) = (k + 2*2) % 7, if full…

Try …

(5+9)%7 = 0

14

12 %7 = 5

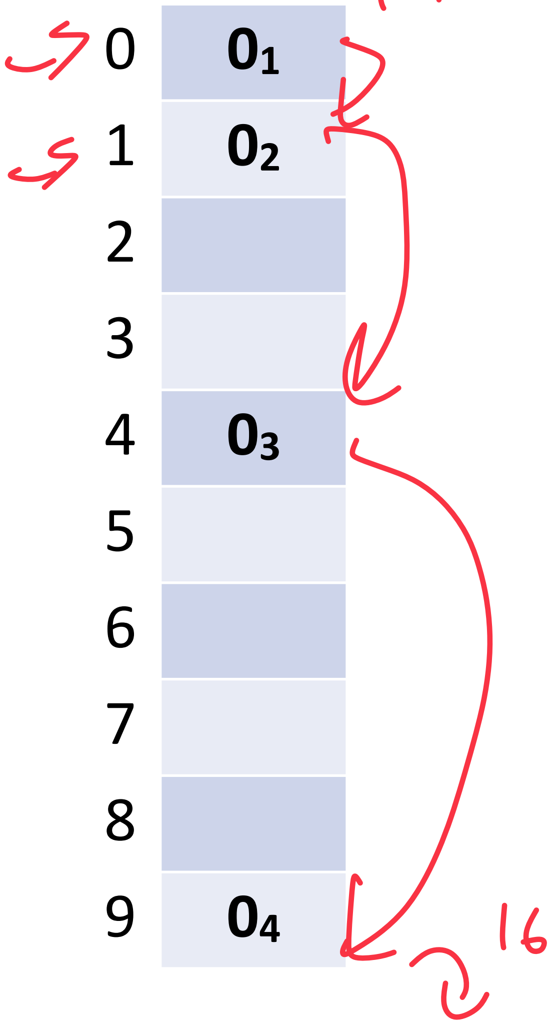
5+1

(5+4)%7 = 2

A Problem w/ Quadratic Probing

Secondary clustering:

Hashed to 0 four times



0	o₁
1	o₂
2	
3	
4	o₃
5	
6	
7	
8	
9	o₄

Description: Individual collisions still make long chains

Remedy: Be less consistent (but still deterministic)

(Example of closed hashing)

Collision Handling: Double Hashing

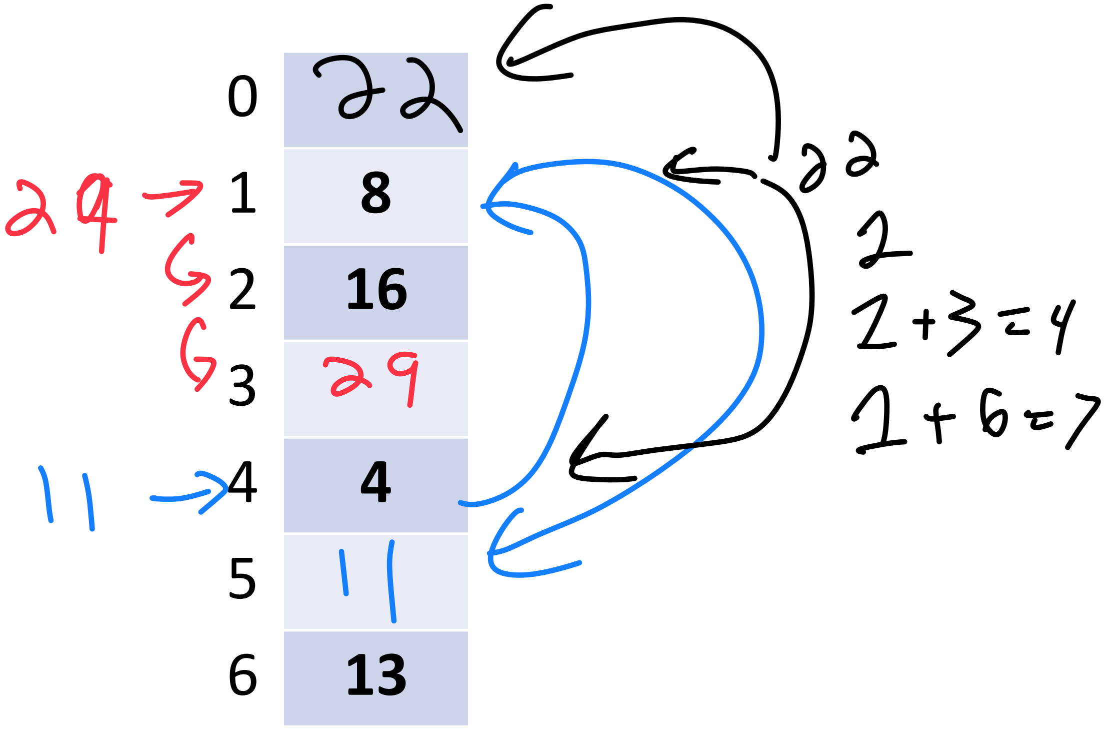
S = { 16, 8, 4, 13, 29, 11, 22 }

$h_1(k) = k \% 7$ 1 4 1

$h_2(k) = 5 - (k \% 5)$ 1 4 3

|S| = n

|Array| = m



0	22
1	8
2	16
3	29
4	4
5	11
6	13

$h(k, i) = (h_1(k) + i*h_2(k)) \% 7$

Try $h(k) = (k + \underline{0}*h_2(k)) \% 7$, if full...

Try $h(k) = (k + \underline{1*h_2(k))} \% 7$, if full...

Try $h(k) = (k + \underline{2*h_2(k)) \% 7}$, if full...

Try ...

$4 + 4*1 = 8 \% 7 = 1$

$4 + 4*2 = 12 \% 7 = 5$

Running Times

(Expectation under SUHA)

(Don't memorize these equations, no need.)

Open Hashing:

insert: $O(1)$.

find/ remove: $1 + \alpha$.

Closed Hashing:

insert: $\frac{1}{1 - \alpha}$.

find/ remove: $\frac{1}{1-\alpha}$.

items

$= n/m$ total spaces

$\alpha =$ how full our hash table is

$\underline{1} + \alpha + \alpha^2 + \alpha^3 + \alpha^4 + \ldots$

collide once twice = taylor series

Running Times (Don't memorize these equations, no need.)

The expected number of probes for find(key) under SUHA

Linear Probing:

- Successful: **½(1 + 1/(1-α))**
- Unsuccessful: **½(1 + 1/(1-α))²**

Double Hashing:

- Successful: **1/α * ln(1/(1-α))** $1/.0000002$
- Unsuccessful: **1/(1-α)**

$0 \le \alpha < 1$

Open hash
α is unbounded

Separate Chaining:

- Successful: **1 + α/2**
- Unsuccessful: **1 + α**

$\alpha = n/m = 200000$

$1/m$ = percentage full in table

Instead, observe:

- As α increases: My runtime increases

(The table gets slower)

- If α is constant: :)

→ our runtime is constant

Running Times

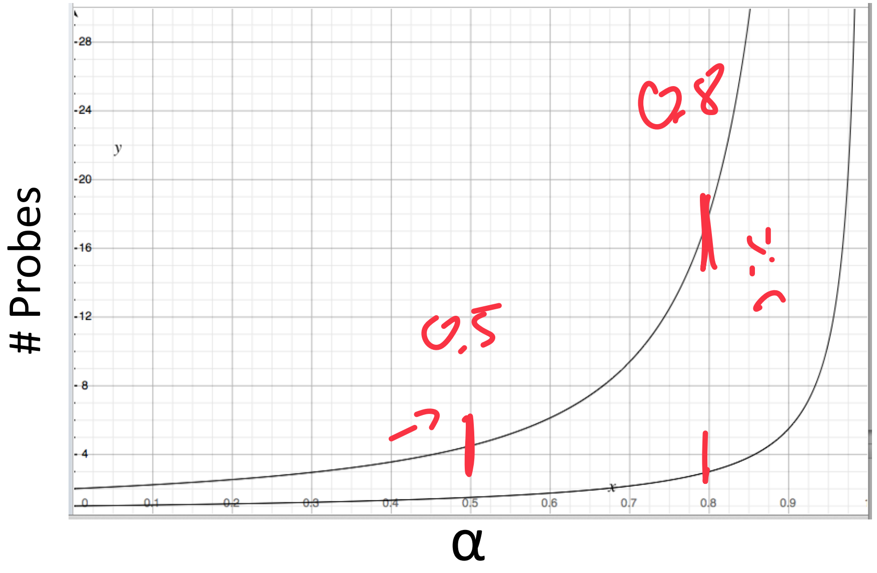
The expected number of probes for find(key) under SUHA

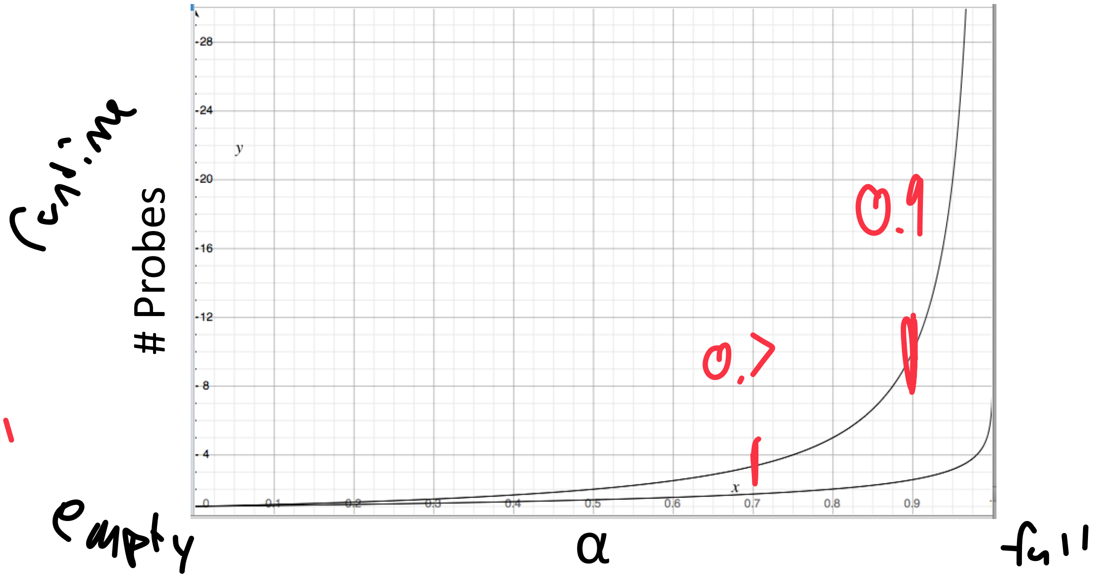
Linear Probing:

- Successful: **½(1 + 1/(1-α))**
- Unsuccessful: **½(1 + 1/(1-α))²**

Double Hashing:

- Successful: **1/α * ln(1/(1-α))**
- Unsuccessful: **1/(1-α)**





When do we resize? 0.7 - 0.9

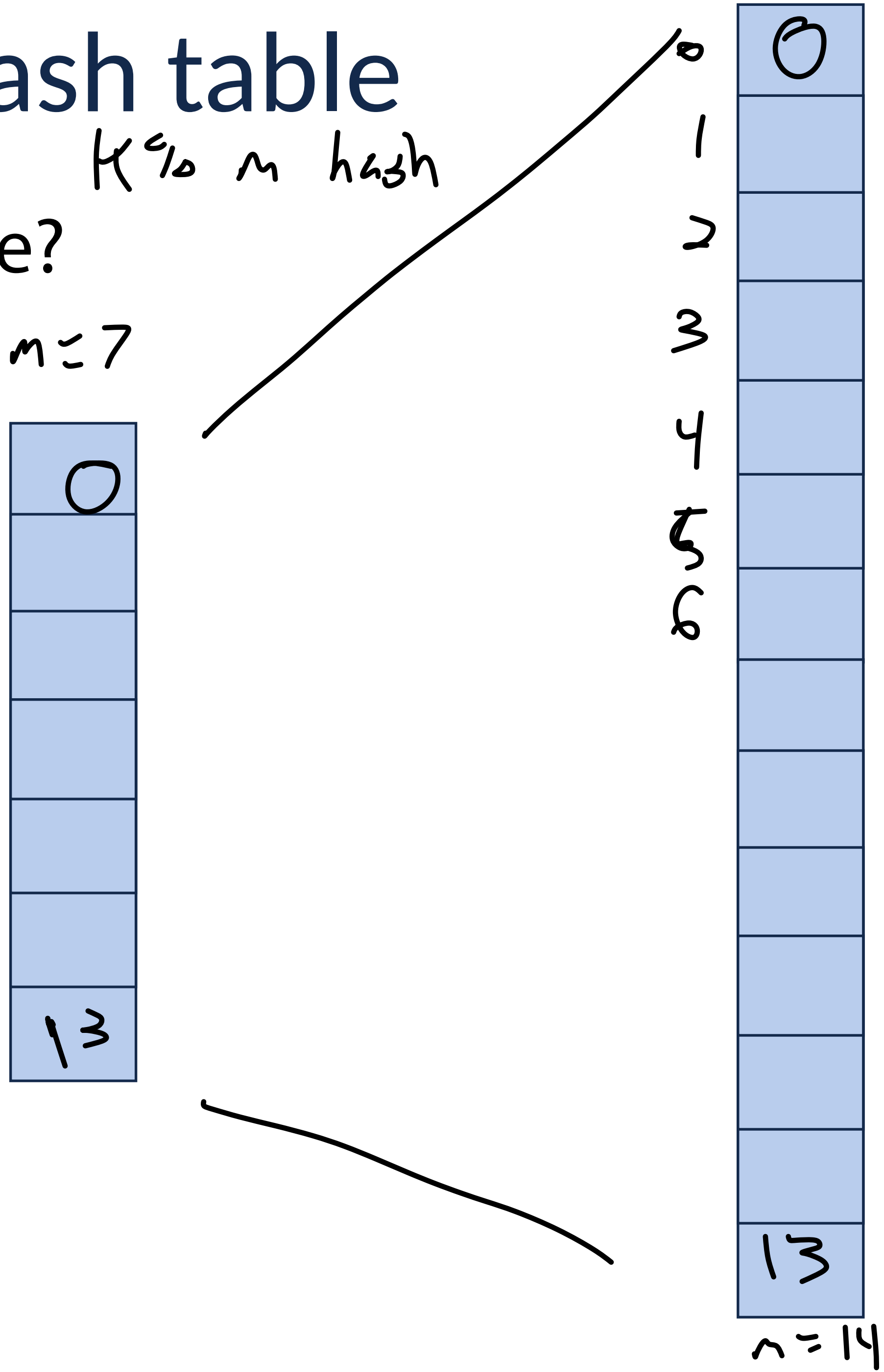
↳ Not when full but when 70-90% full

Resizing a hash table

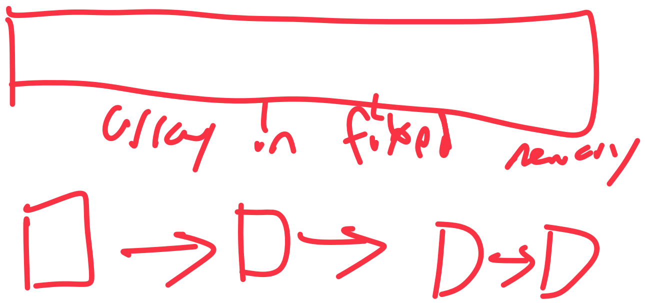
k % m hash

How do you resize?

1) Allocate new list
2) Rehash every item



Which collision resolution strategy is better?



- Big Records: Separate chaining
- Structure Speed: Double hashing
  ↳ collisions make table slow

What structure do hash tables implement?

↳ Dictionaries in Python are hash tables

What constraint exists on hashing that doesn't exist with BSTs?

↳ Probabilistic SUHA assumption
↳ Big O is O(n), real world is O(1)*

↓ O(log n)

Better guaranteed

Why talk about BSTs at all?

↳ Trees can get closest match | Hash table only does exact lookup

Running Times Review

Balanced BST

	Hash Table	AVL	Linked List
Find	Expectation*: $O(1)^*$ Worst Case: $O(n)$	$O(\log n)$	$O(n)$
Insert	Expectation*: Worst Case: $O(1)$	$O(\log n)$	$O(1)$
Remove	Expectation*: $O(1)^*$ Worst Case: $O(n)$	$O(\log n)$	$O(n)$
Storage Space			

Where do we go from here?

Hash tables were a much needed detour (and allows for next week’s lab to be a review session)

Still to discuss:

- Graph Algorithms
- Sets and hash table extensions
- ‘Bonus topics’

Assignments remaining:

- This week’s MP is last MP
- This week’s lab is last lab

MP Algorithm

A trio of independent graph algorithm projects designed as a capstone

Learning Objectives:

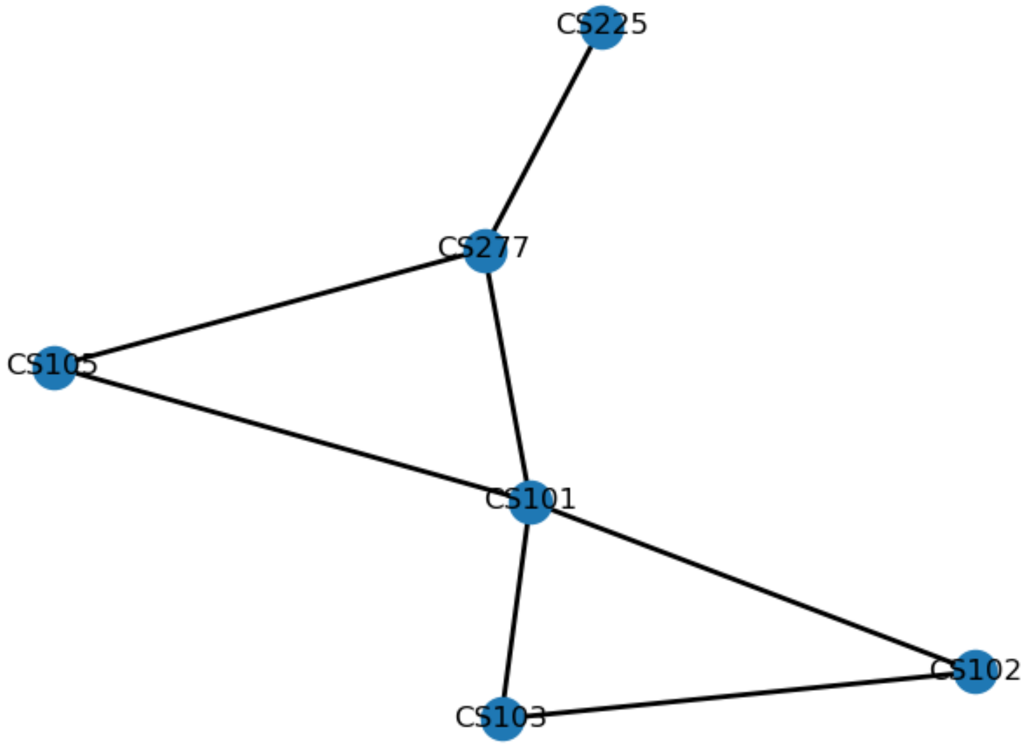
Practice parsing different data formats into graphs

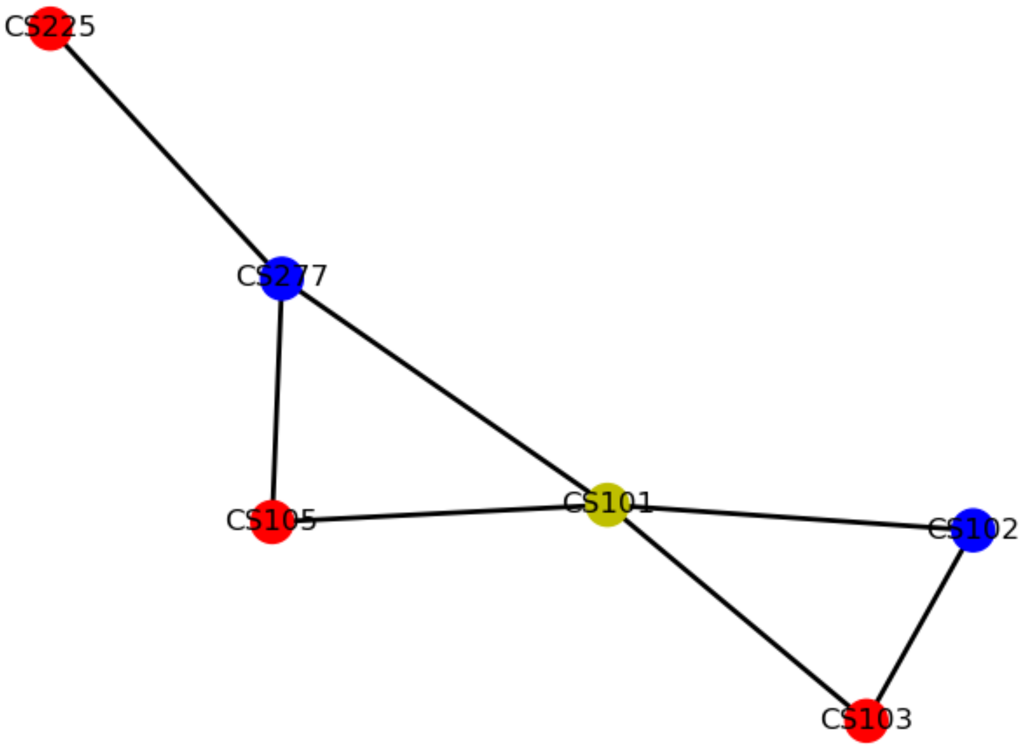
Practice fundamentals of accessing and modifying graphs in NetworkX

Create a greedy heuristic algorithm to solve a complex problem

Part 1: Graph Coloring

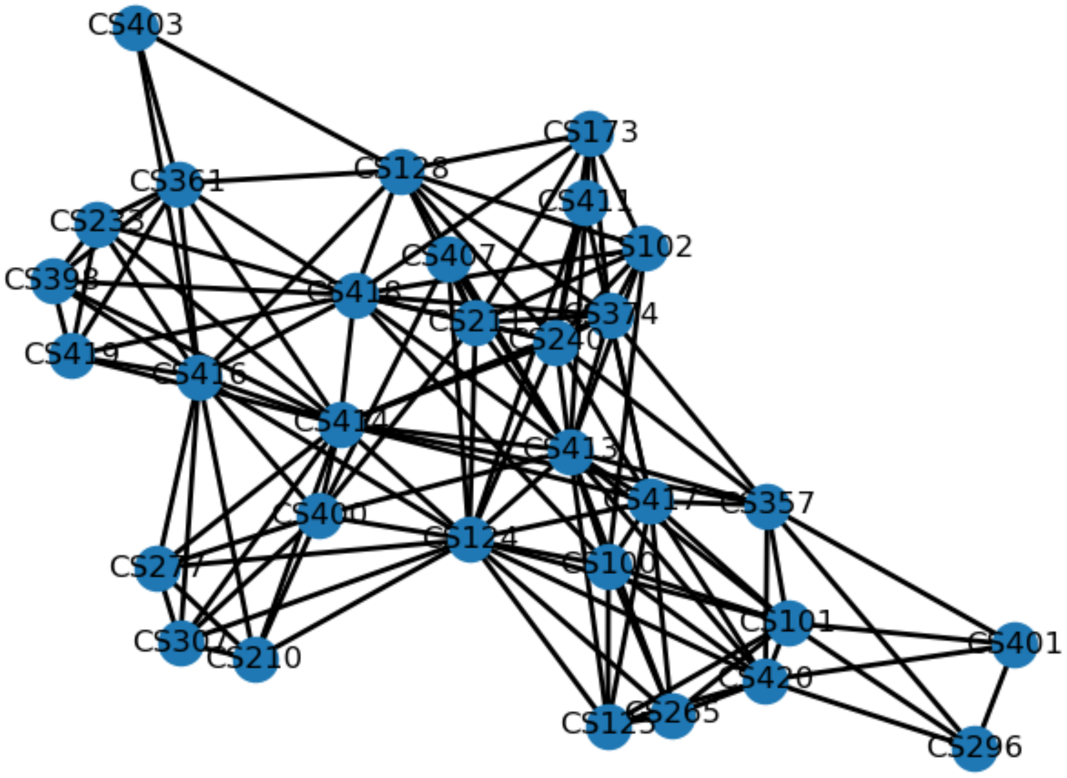
We want to assign a color label to every node in the graph such that no two neighbors have the same color

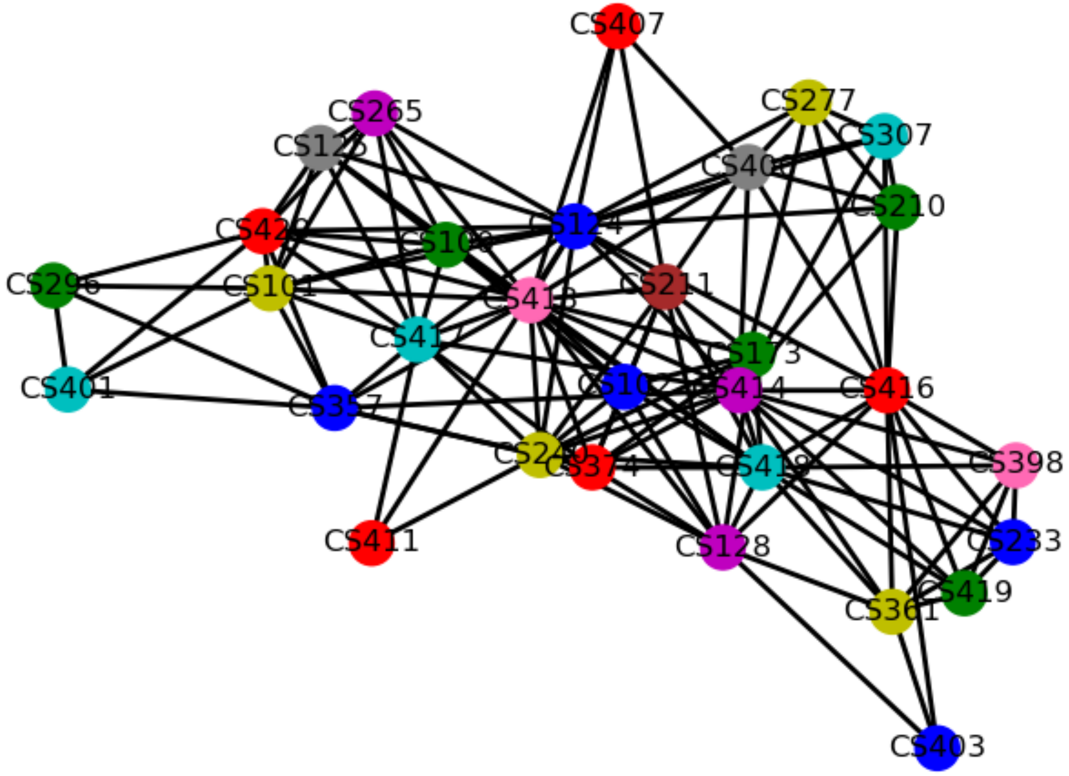




Part 1: Graph Coloring

If we want to minimize the number of colors, this can get very computationally intensive very quickly…





Part 1: Graph Coloring

We will do this using a greedy heuristic of our own design:

Given a graph, a list of vertices, and a list of colors

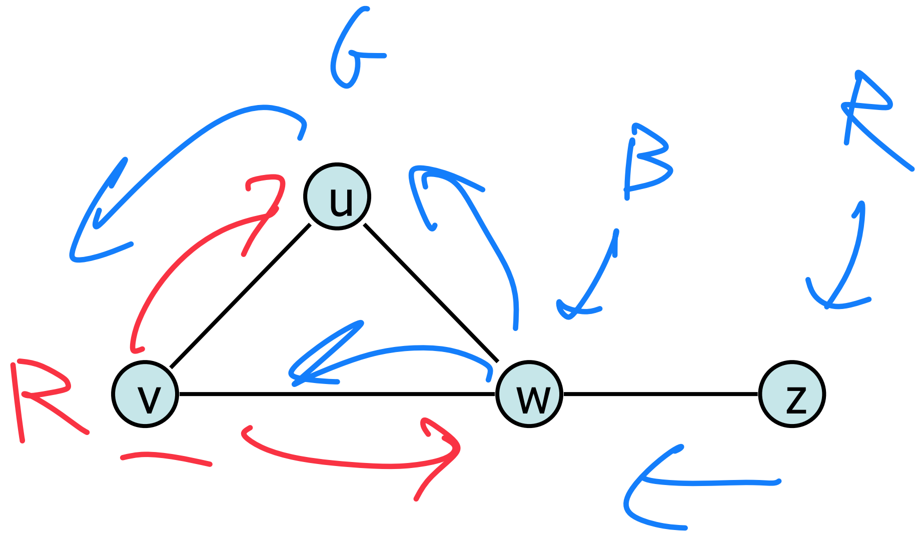
For each node in a specified order:

- Check the color of every neighbor
- Label the node the first unused color

Ex:

Nodes: V, U, W, Z

Color: R, G, B



Part 1: Graph Coloring

We will do this using a greedy heuristic of our own design:

Given a graph, a list of vertices, and a list of colors

For each node in a specified order:

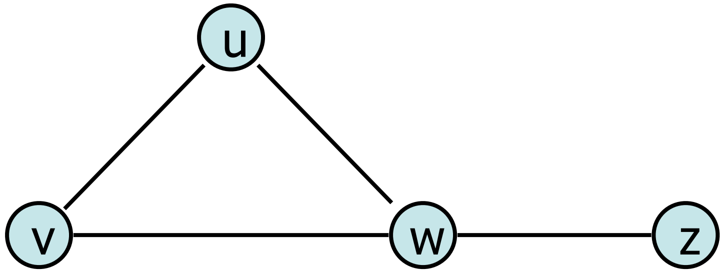
    Check the color of every neighbor

    Label the node the first unused color

Ex:

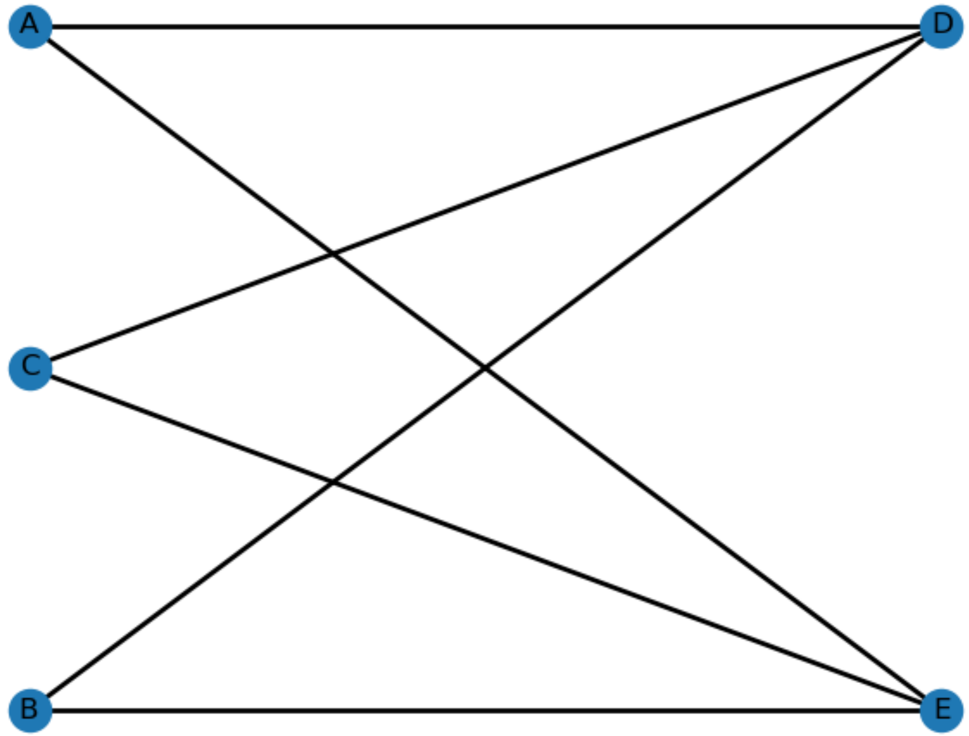
Nodes: W, V, U, Z

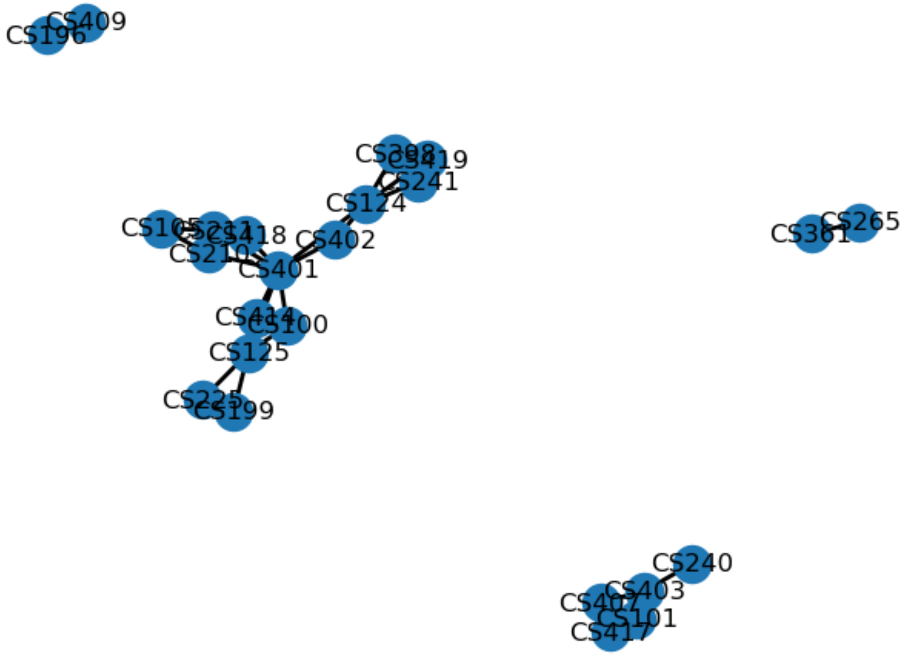
Color: R, G, B



Part 1: Graph Coloring

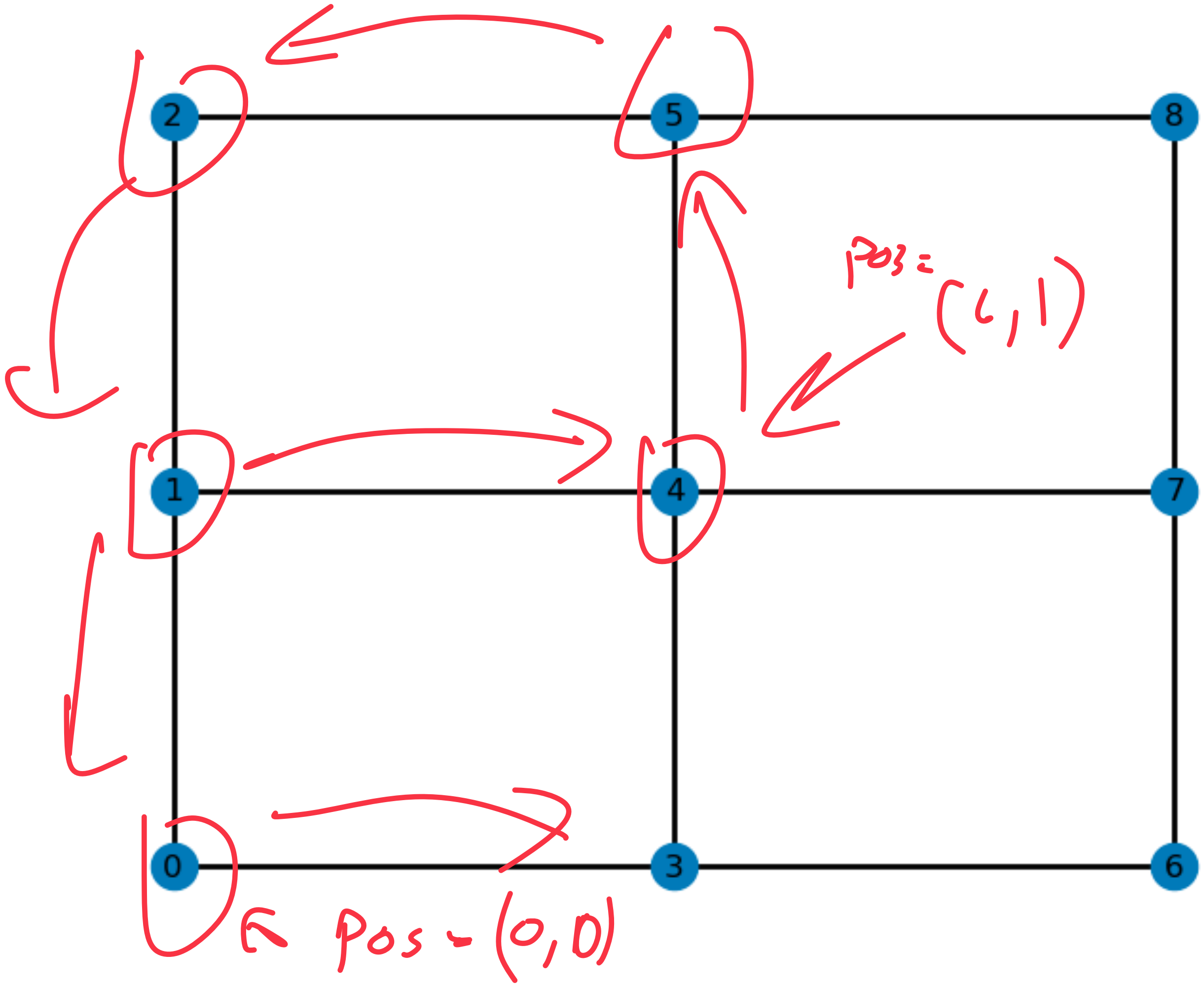
You are also responsible for making two different types of graphs:





Part 2: Pirate Walk (on a graph!)

Given a grid graph (with node attribute 'pos'), a start node, and a path string, record the list of vertices the path goes through in order.



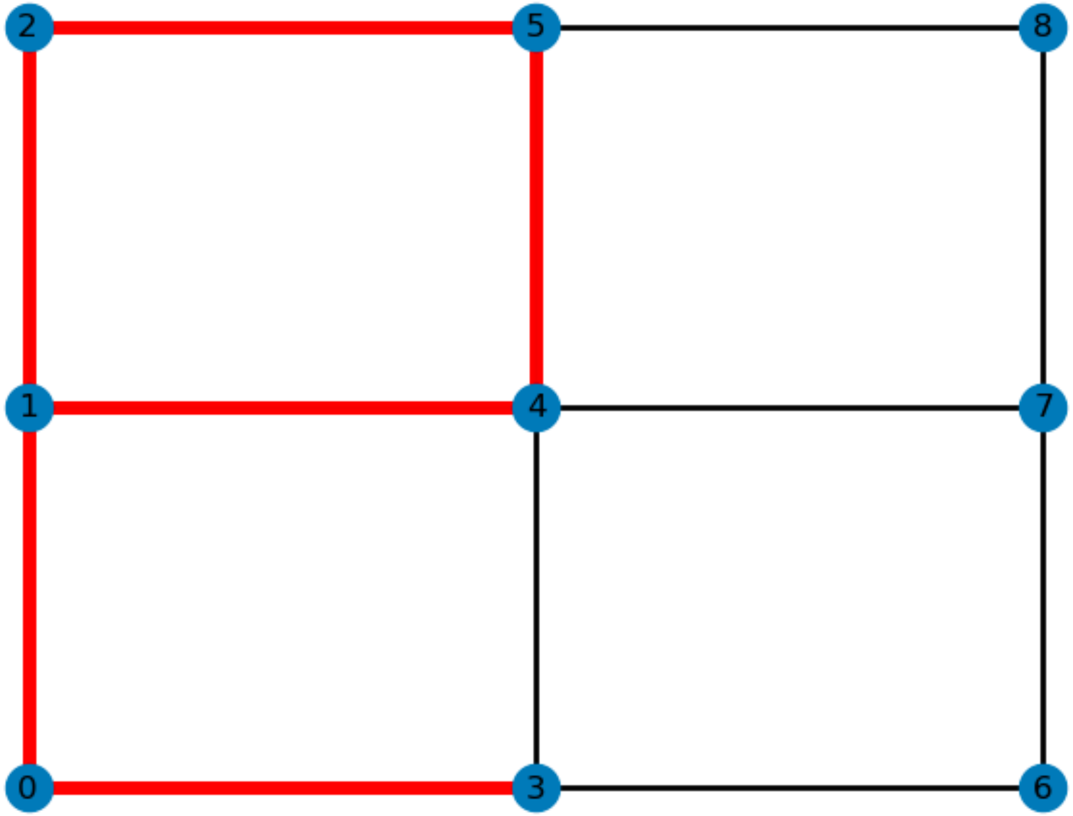
Start: 1

Path: "ENWSSE"

Output: 1, 4, 5, 2, 1, 0, 3

Part 2: Pirate Walk (on a graph!)

Given a grid graph (with node attribute 'pos'), a start node, and a path string, record the list of vertices the path goes through in order.



Start: 1

Path: "ENWSSE"

Output: [1, 4, 5, 2, 1, 0, 3]

Part 3: OpenFlights Flight Paths

Given two csv files (vertex & edge), build a weighted NetworkX graph

645,"Haugesund Airport","Haugesund","Norway","HAU","ENHD",59.34529876709,5.2083601951599,86,1,"E","Europe/Oslo","airport","OurAirports"

11092,"Larned Pawnee County Airport","Larned","United States",\N,"KLQR",38.20859909,-99.08599854,2012,-5,"A",\N,"airport","OurAirports"

293,"Djerba Zarzis International Airport","Djerba","Tunisia","DJE","DTTJ",33.875,10.775500297546387,19,1,"E","Africa/Tunis","airport","OurAirports"

"KLQR","DTTJ"

“ENHD","KLQR"

“ENHD","DTTJ"

Solve each problem your own way

Part 3 (and to a lesser degree the overall assignment) has less structure than past assignments — by design!

Use what you've learned in the class previously to build graphs from different inputs

You can (and are encouraged to) freely discuss your approach to solving these problems