Collections, Part Three

Friday Four Square!
Today at 4:15PM, Outside Gates

Lexicon

Lexicon

- A `Lexicon` is a container that stores a collection of words.
- No definitions are associated with the words; it is a “lexicon” rather than a “dictionary.”
- Contains operations for
  - Checking whether a word exists.
  - Checking whether a string is a prefix of a given word.

Tautonyms

- A **tautonym** is a word formed by repeating the same string twice.
  - For example: murmur, couscous, papa, etc.
- What English words are tautonyms?

`foreach`

- You can loop the elements of any collection class using the `foreach` macro:

```
foreach (type var in collection) {
    /* … do something with var … */
}
```

- `foreach` is ***not*** a part of standard C++; it's a *macro* that we've built to keep things simple.

Some Aa

http://upload.wikimedia.org/wikipedia/commons/f/f1/Aa_large.jpg

One Bulbul

http://travel.paintedstork.com/blog/image/yellow_browed_bulbul.jpg

More than One Caracara

http://www.greglasley.net/images/CO/Crested-Caracara-F3.jpg

Introducing the Dikdik

Anagrams

- Two phrases are **anagrams** of one another if they have the same letters, but in a different order.
- Examples:
  - Stanford University → A Trusty Finned Visor
  - Keith Schwarz → Zither Whacks
  - Dawson Zhou → Whoa! Zounds!
- **Question:** Given an English word, can we find all anagrams of that word?

Anagram Clusters

- An **anagram cluster** is a set of words that are all anagrams of one another.

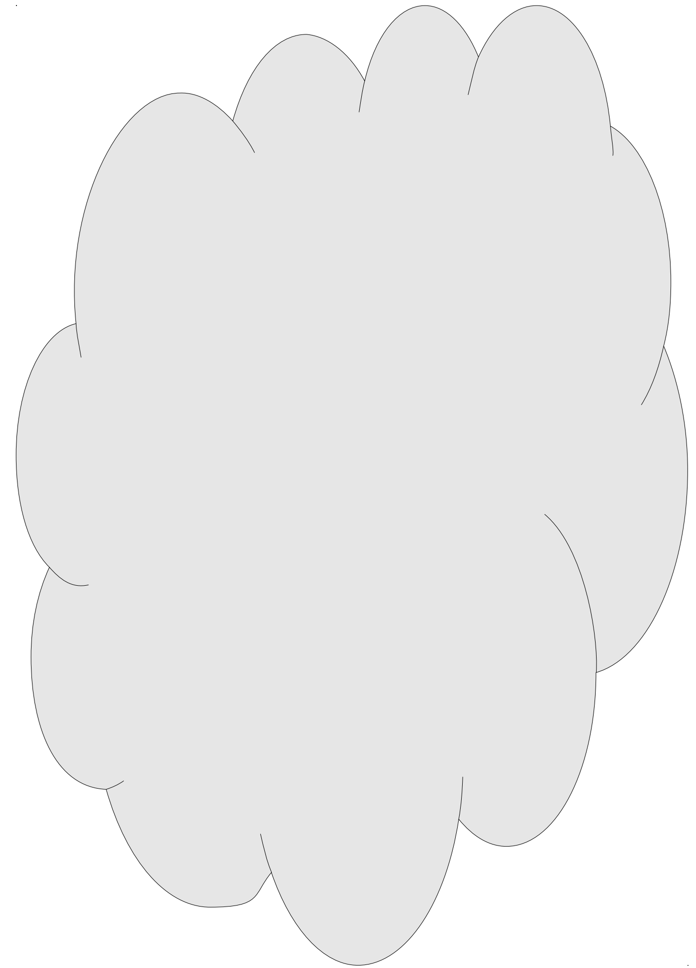
  stop ↔ tops ↔ pots ↔ spot ↔ opts ↔ post

- If we want to find all anagrams of a word, we can find its anagram cluster, then list off all the words in that cluster.
- Two questions:
  - How do we store an anagram cluster?
  - How do we find the anagram cluster associated with a given word?

Set

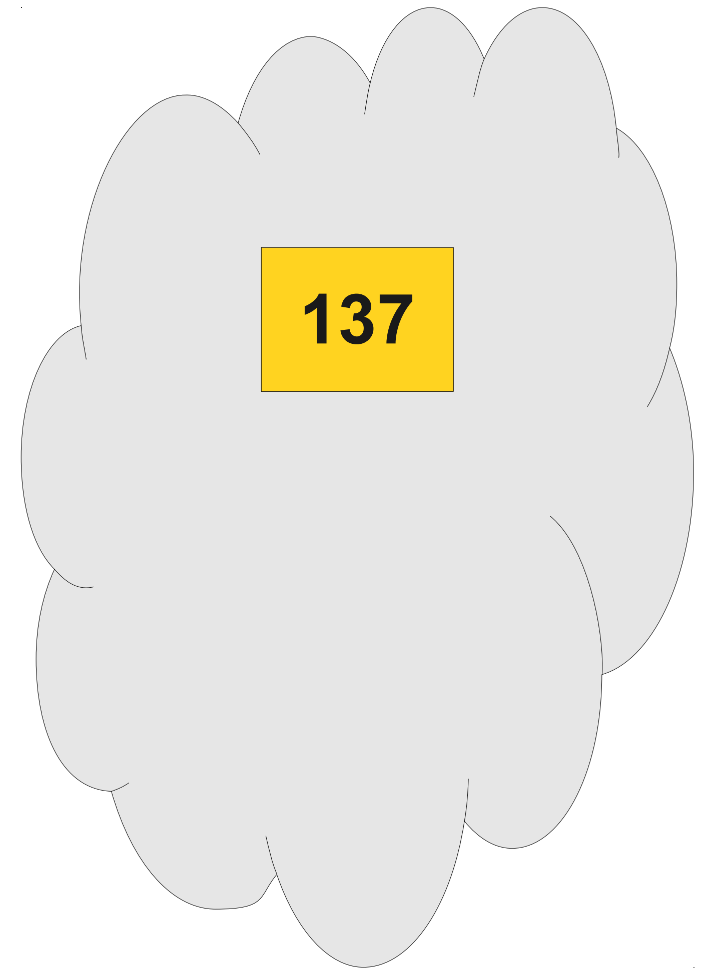
Set

- The **Set** represents an unordered collection of distinct elements.
- Elements can be added and removed, and you can check whether or not an element exists.



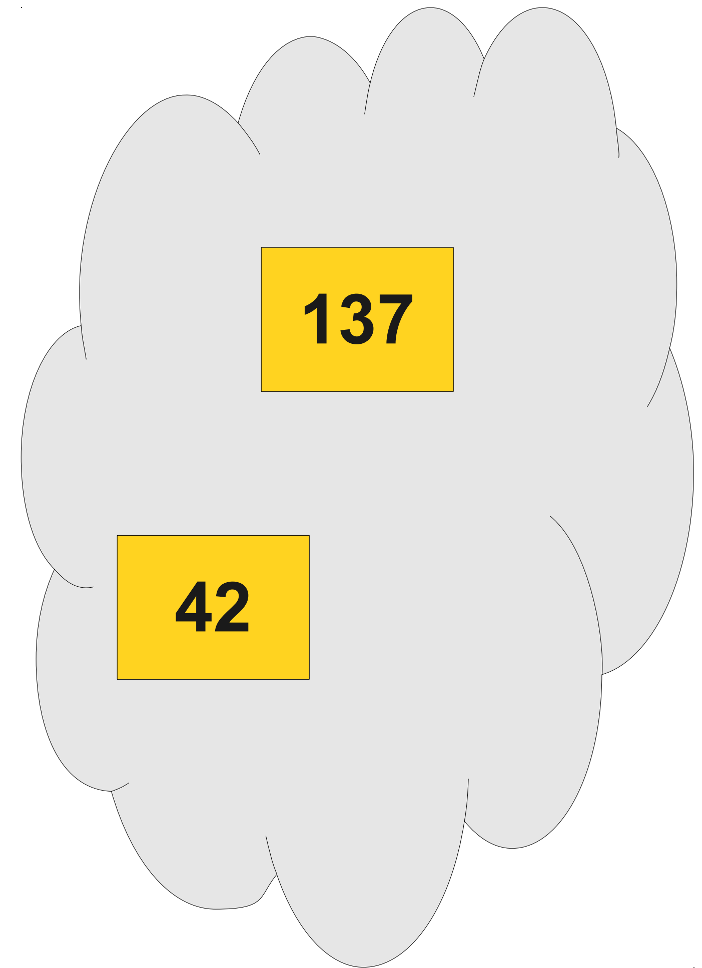
Set

- The `Set` represents an unordered collection of distinct elements.
- Elements can be added and removed, and you can check whether or not an element exists.



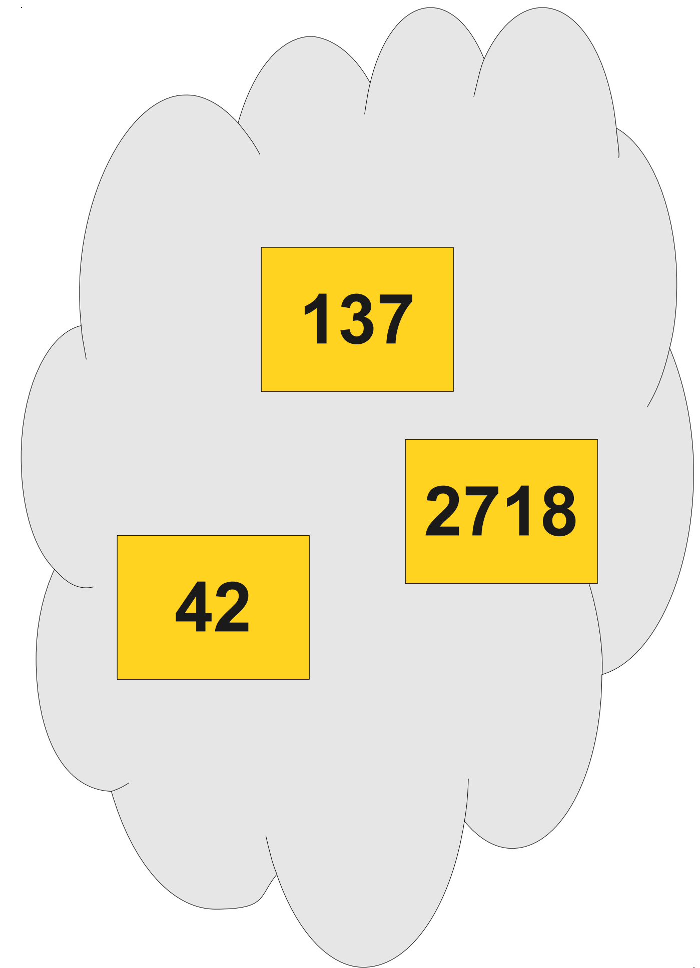
Set

- The **Set** represents an unordered collection of distinct elements.
- Elements can be added and removed, and you can check whether or not an element exists.



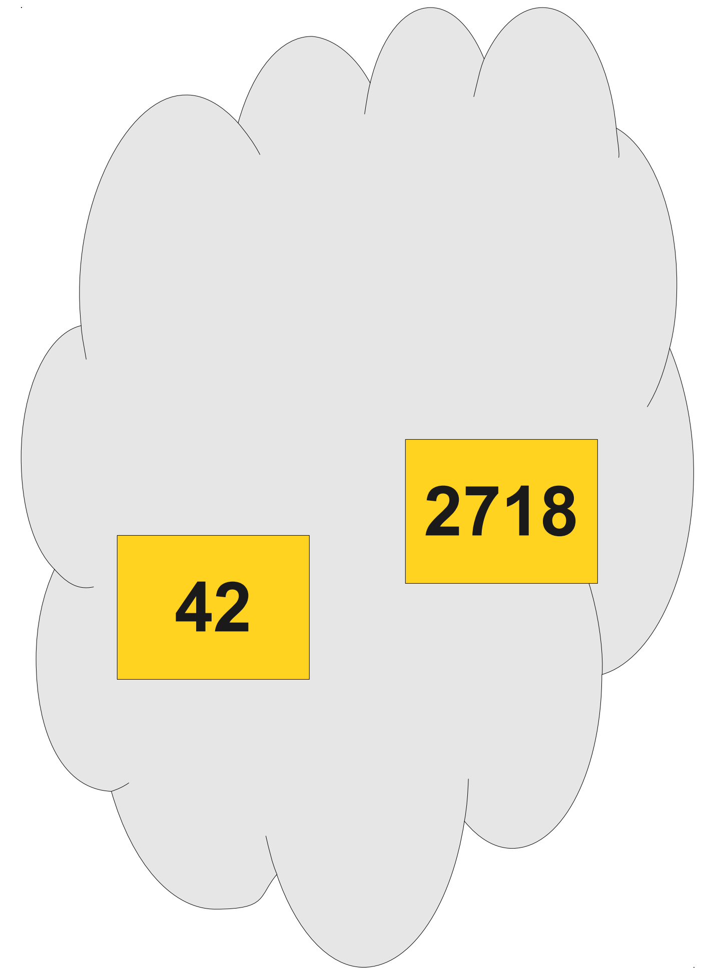
Set

- The **Set** represents an unordered collection of distinct elements.
- Elements can be added and removed, and you can check whether or not an element exists.



Set

- The **`Set`** represents an unordered collection of distinct elements.
- Elements can be added and removed, and you can check whether or not an element exists.



Operations on Sets

- You can add a value to a set by writing

```
set += value;
```

- You can remove a value from a set by writing

```
set -= value;
```

- You can check if a value exists by writing

```
set.contains(value)
```

- Many more operations available (union, intersection, difference, subset, etc.), so be sure to check the documentation.

Map

Map

- The `Map` class represents a set of key/value pairs.
- Each key is associated with a unique value.
- Given a key, can look up the associated value.

Map

- The **Map** class represents a set of key/value pairs.
- Each key is associated with a unique value.
- Given a key, can look up the associated value.

Map

- The **`Map`** class represents a set of key/value pairs.
- Each key is associated with a unique value.
- Given a key, can look up the associated value.

CS106B	Awesome!
Dikdik	Cute!

Map

- The **Map** class represents a set of key/value pairs.
- Each key is associated with a unique value.
- Given a key, can look up the associated value.

CS106B	Awesome!
Dikdik	Cute!
This Slide	Self Referential

Map

- The `Map` class represents a set of key/value pairs.
- Each key is associated with a unique value.
- Given a key, can look up the associated value.

CS106B	Awesome!
Dikdik	**Very** Cute!
This Slide	Self Referential

Using the Map

- You can create a map by writing

```
Map<KeyType, ValueType> map;
```

- You can add or change a key/value pair by writing

```
map[key] = value;
```

  If the key doesn't already exist, it is added.

- You can read the value associated with a key by writing

```
map[key]
```

  If the key doesn't exist, it is added and associated with a default value.

- You can check whether a key exists by calling

```
map.containsKey(key)
```

Sorting Letters

- One way to check whether two words are anagrams of one another is to reorder the letters into ascending order:

bleat → abelt

table → abelt

Sorting Letters

- One way to check whether two words are anagrams of one another is to reorder the letters into ascending order:

  bleat → abelt

  table → abelt

- **Idea**: Build a `Map<string, Set<string>>` to represent anagram clusters.
  - Each key is the letters of a word in sorted order.
  - Each value is the set of all words with those letters.

Ordering in `foreach`

- When using **`foreach`** to iterate over a collection:
  - In a **`Vector`**, **`string`**, or array, the elements are retrieved in order.
  - In a **`Map`**, the *keys* are returned in sorted order.
  - In a **`Set`** or **`Lexicon`**, the values are returned in sorted order.
  - In a **`Grid`**, the elements of the first row are returned in order, then the second row, etc. (this is called *row-major order*).

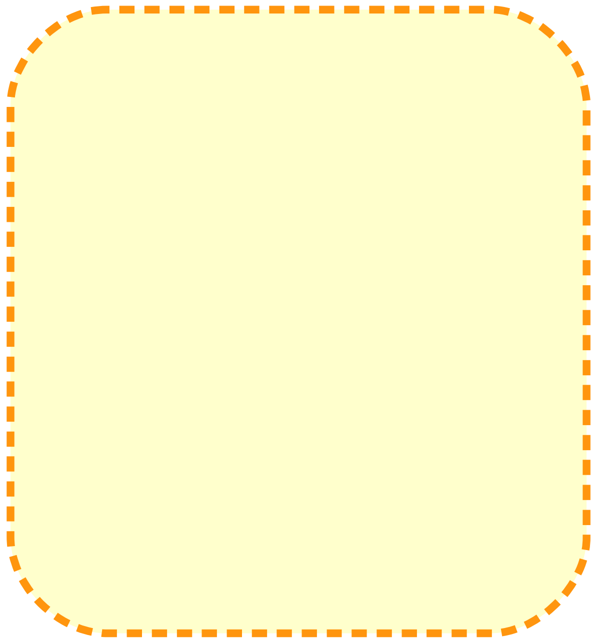
Counting Sort

Counting Sort

b	a	n	a	n	a

Counting Sort

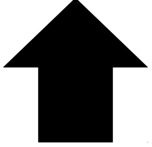
b	a	n	a	n	a

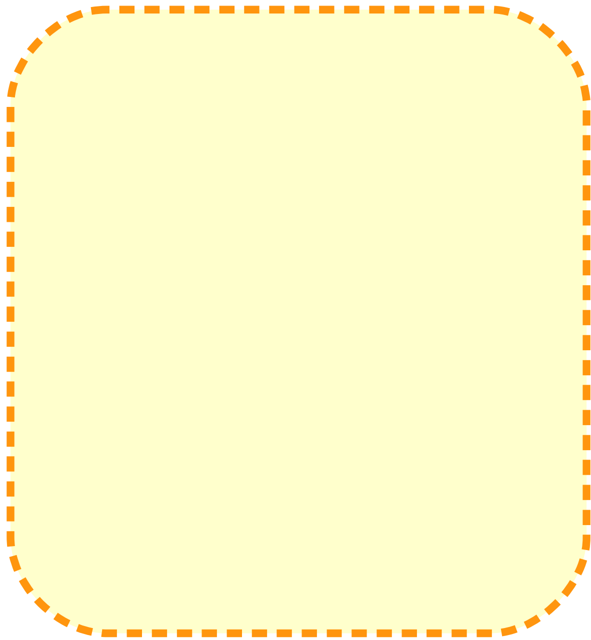


Map<char, int>

Counting Sort

b	a	n	a	n	a

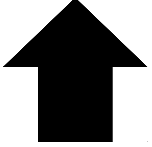


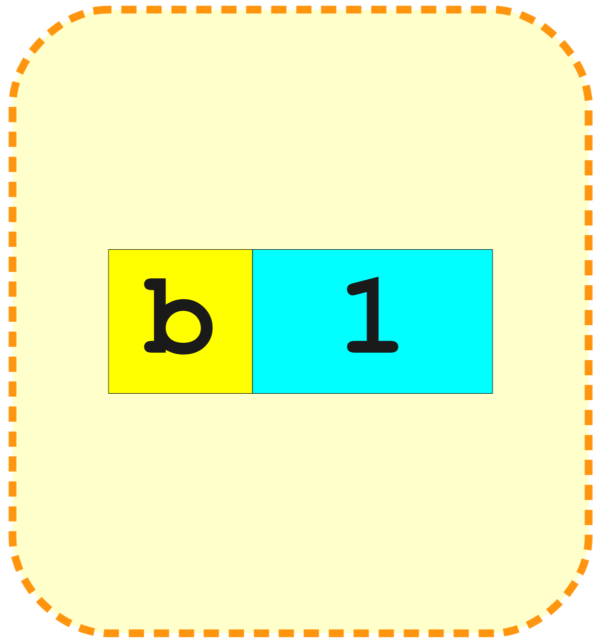


Map<char, int>

Counting Sort

b	a	n	a	n	a

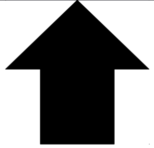


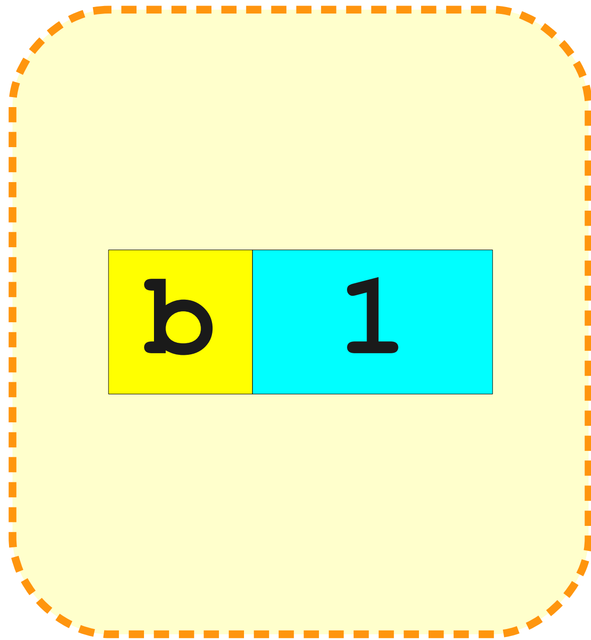


Map<char, int>

Counting Sort

b	a	n	a	n	a

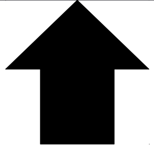




Map<char, int>

Counting Sort

b	a	n	a	n	a

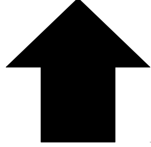


a	1
b	1

Map<char, int>

Counting Sort

b	a	n	a	n	a

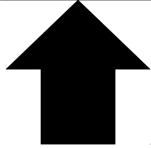


a	1
b	1

Map<char, int>

Counting Sort

b	a	n	a	n	a

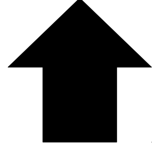


a	1
b	1
n	1

`Map<char, int>`

Counting Sort

b	a	n	a	n	a

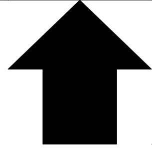


a	1
b	1
n	1

Map<char, int>

Counting Sort

b	a	n	a	n	a

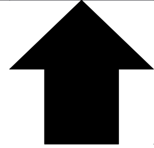


a	2
b	1
n	1

Map<char, int>

Counting Sort

b	a	n	a	n	a

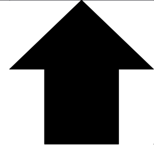


a	2
b	1
n	1

Map<char, int>

Counting Sort

b	a	n	a	n	a

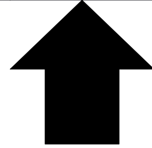


a	2
b	1
n	2

Map<char, int>

Counting Sort

b	a	n	a	n	a

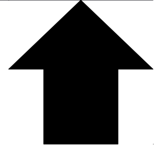


a	2
b	1
n	2

Map<char, int>

Counting Sort

b	a	n	a	n	a



a	3
b	1
n	2

Map<char, int>

Counting Sort

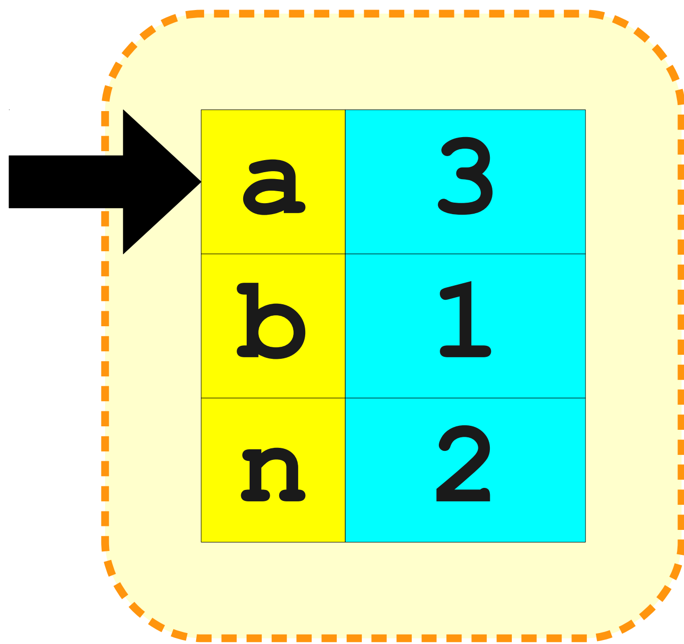
b	a	n	a	n	a

a	3
b	1
n	2

Map<char, int>

Counting Sort

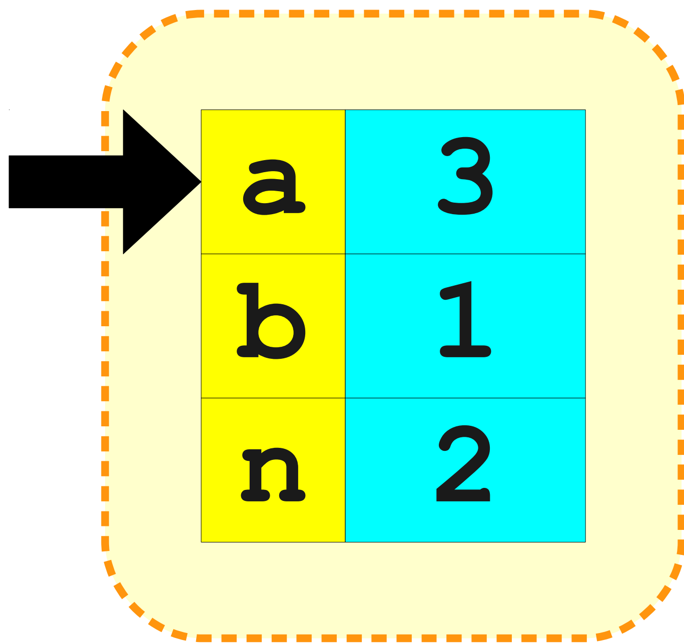
b	a	n	a	n	a



Map<char, int>

Counting Sort

b	a	n	a	n	a

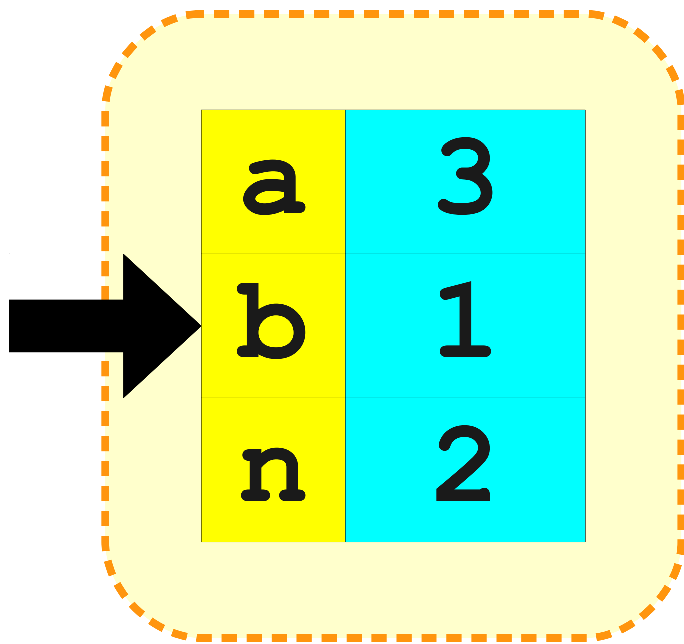


Map<char, int>

a	a	a

Counting Sort

b	a	n	a	n	a

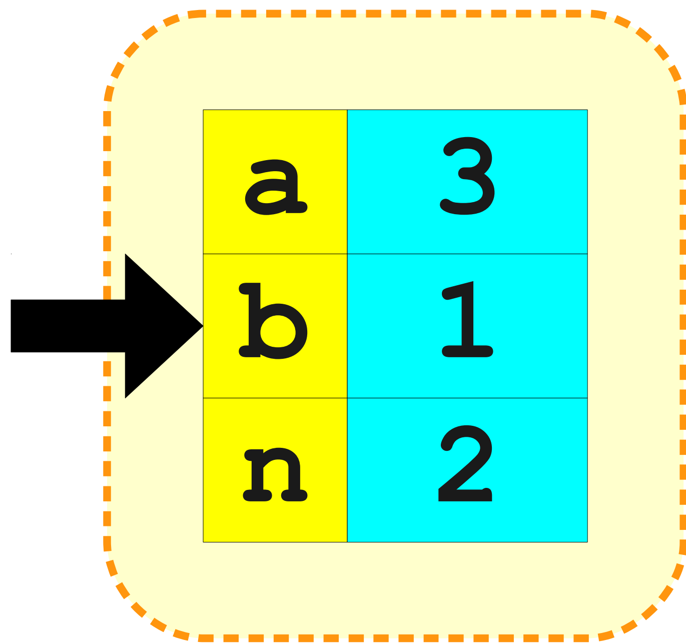


`Map<char, int>`

a	a	a

Counting Sort

b	a	n	a	n	a

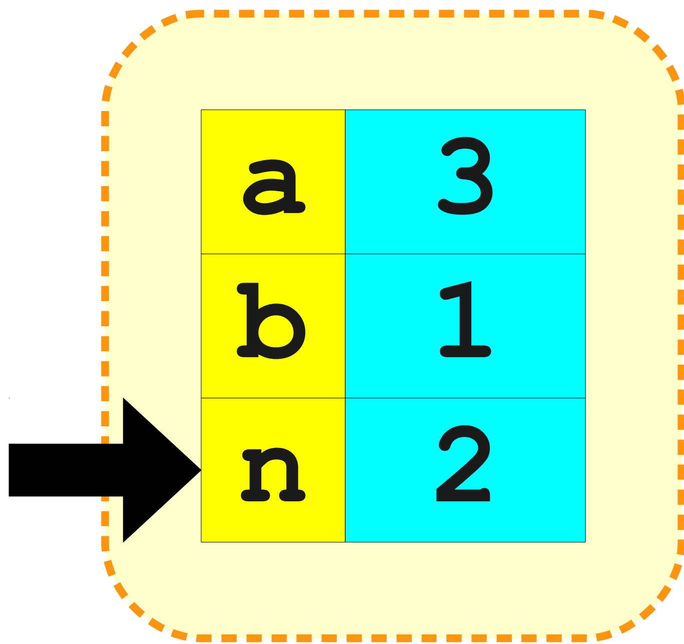


a	a	a	b

Map<char, int>

Counting Sort

b	a	n	a	n	a

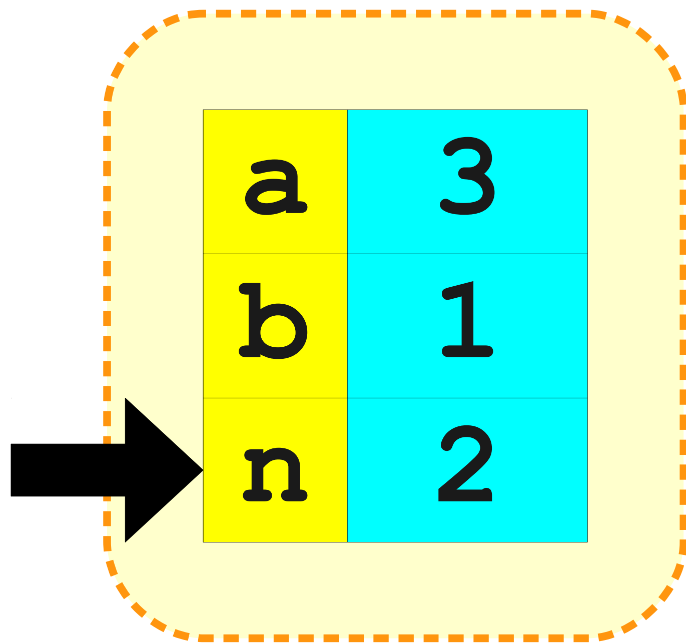


Map<char, int>

a	a	a	b

Counting Sort

b	a	n	a	n	a



a	a	a	b	n	n

Map<char, int>

Counting Sort

b	a	n	a	n	a

a	3
b	1
n	2

Map<char, int>

a	a	a	b	n	n

Next Time

- **`Queue`**
  - A data structure for waiting lines.
- **Password Security**
  - How do you properly store passwords?
  - And what on earth is a hash code?