COMP 250

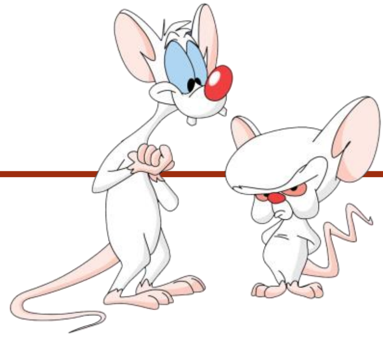
INTRODUC TER SCIENCE

Week 13-1

Giulia Alberini, Fall 2020

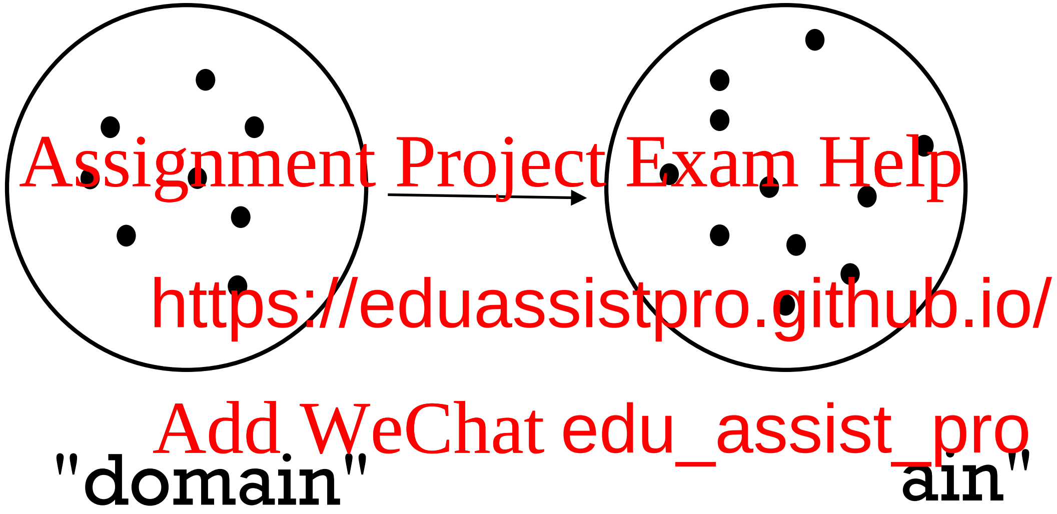
Slides adapted from Michael Langer's

WHAT ARE WE GOING TO DO IN THIS VIDEO?



- Maps

MAP (MATHEMATICS)



A map is a set of pairs $\{ (x, f(x)) \}$.

Each x in domain maps to exactly one $f(x)$ in codomain, but it can happen that $f(x1) = f(x2)$ for different $x1, x2$, i.e. many-to-one.

FAMILIAR EXAMPLES

Calculus 1 and 2 ("functions"):

Asymptotic complexity in CS:

t : input size → number of steps in a algorithm.

MAPS IN EVERYDAY LIFE

The term "map" commonly refers to a 2D spatial representation of a region of the earth's surface.

map(x, y) : position in image → position in 2D Montreal

COLOR MAP

The color map representing the USA election results in 2020.

vote_result : US_state → {D, R}

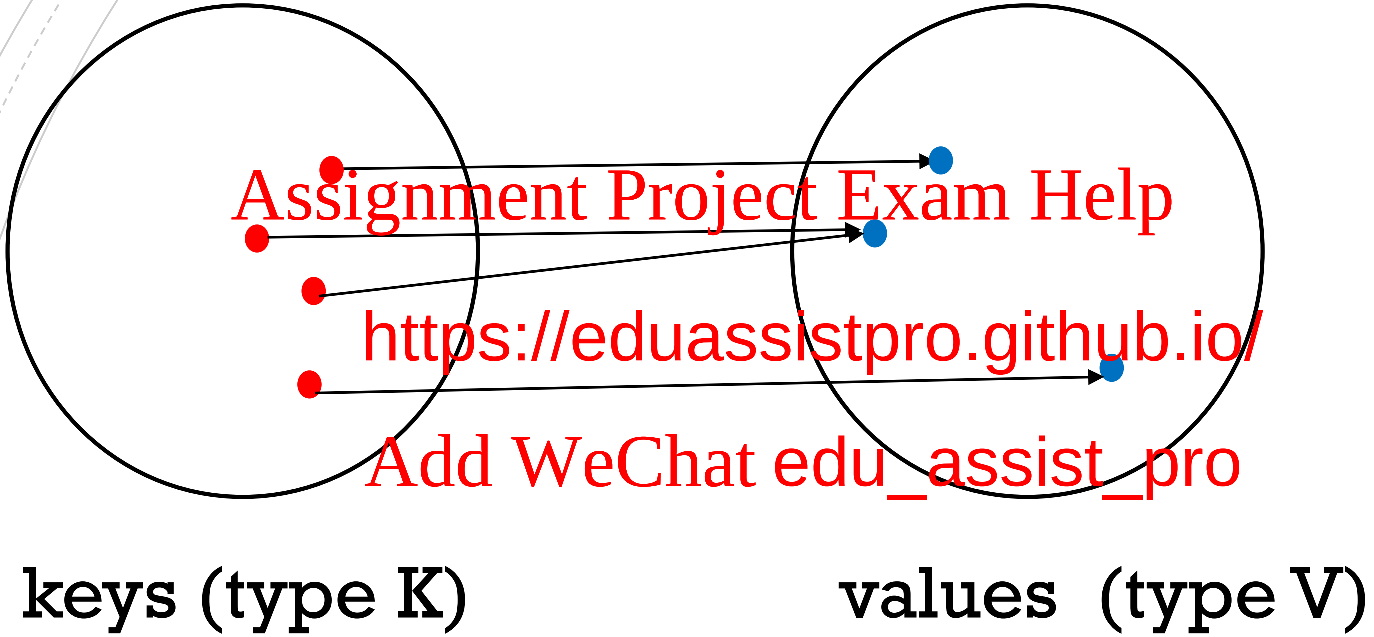
RESTAURANT MENU

menu : dish_name → price

INDEX IN A BOOK

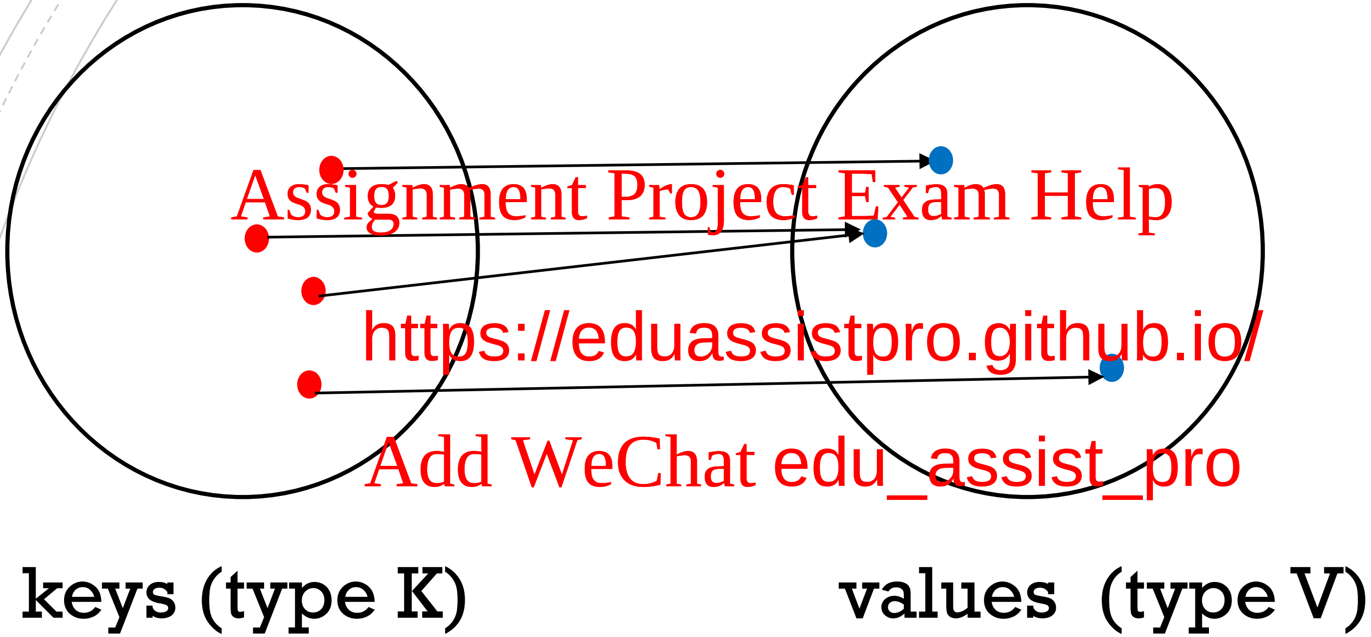
index : term → list of pages

MAP (ADT)



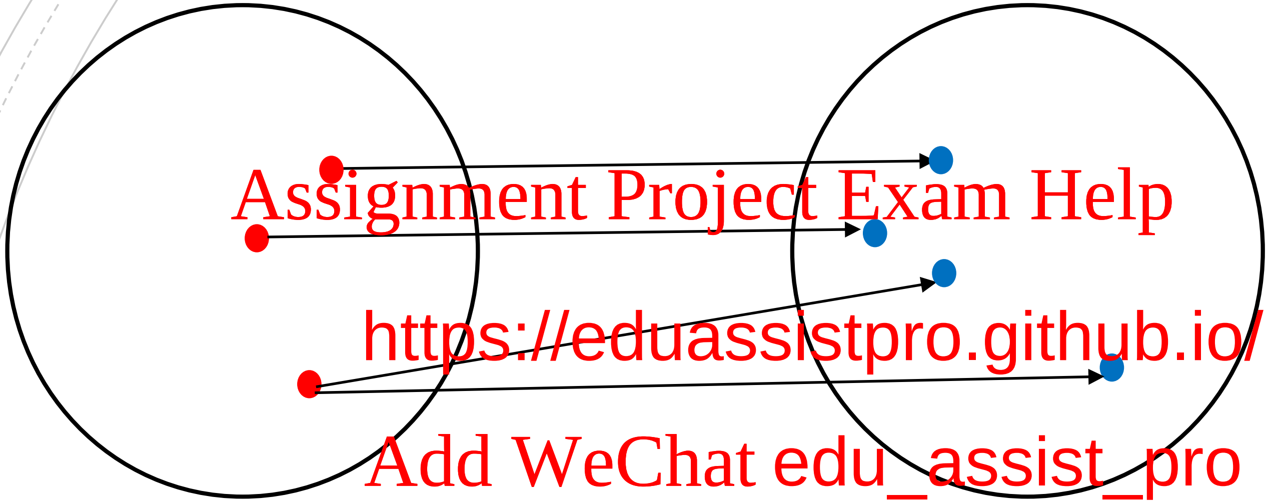
A map is a set of (key, value) pairs.
For each key, there is at most one value.

MAP (ADT)



Note that it is possible for two keys to map to the same value.

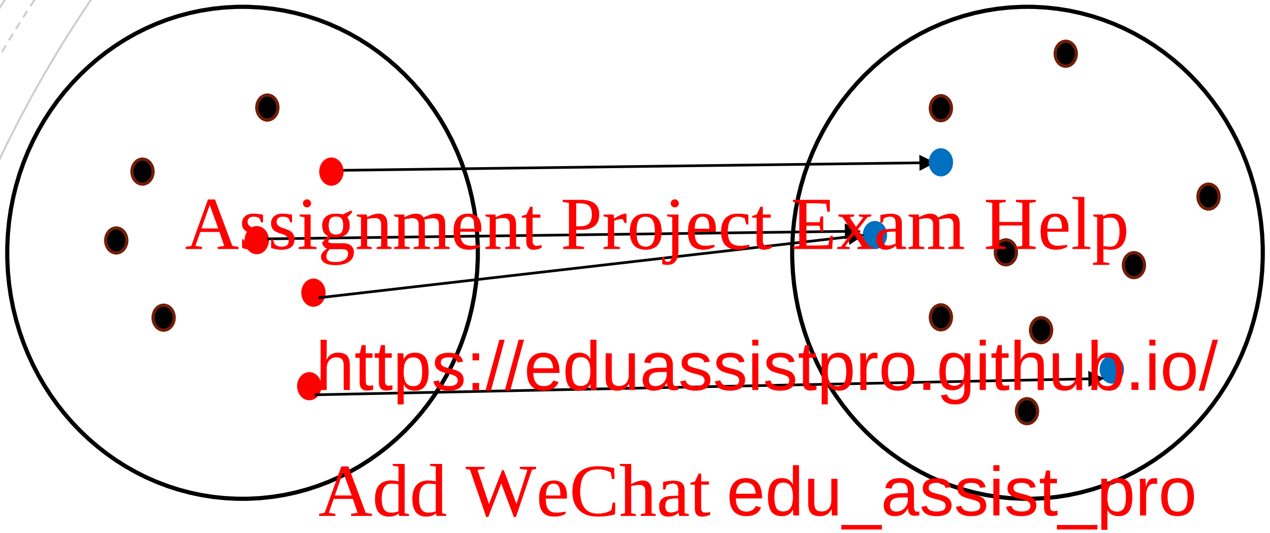
MAP (ADT)



keys (type K) values (type V)

It is NOT allowed for one key to map to two different values! The example above is NOT a map.

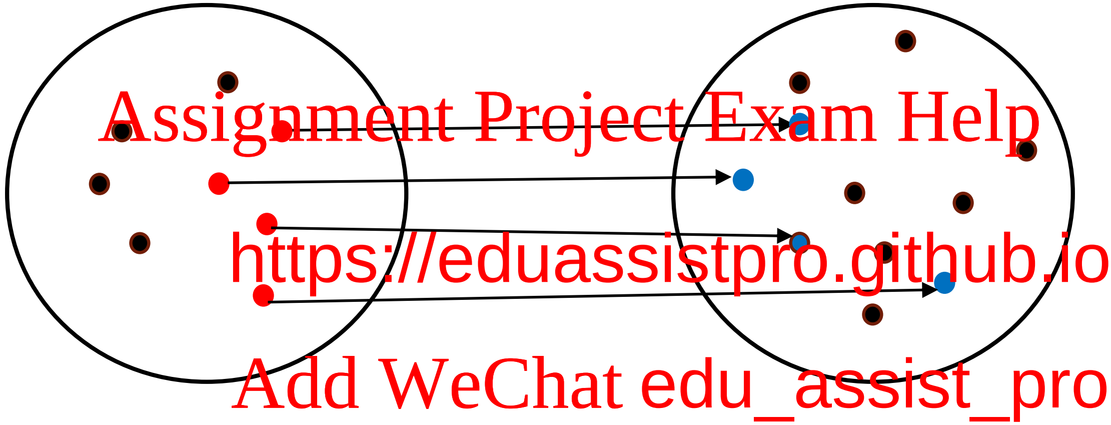
MAP ENTRIES



The black dots here indicate objects (or potential objects) of type K or V that are *not* in the map.

Each (key, value) pair is called an *entry*. In this example, there are four entries.

EXAMPLE



Student IDs

Student records

In COMP 250 this semester, the above mapping has ~650 entries.

Most McGill students are not taking COMP 250 this semester.

Student ID also happens to be part of the student record.

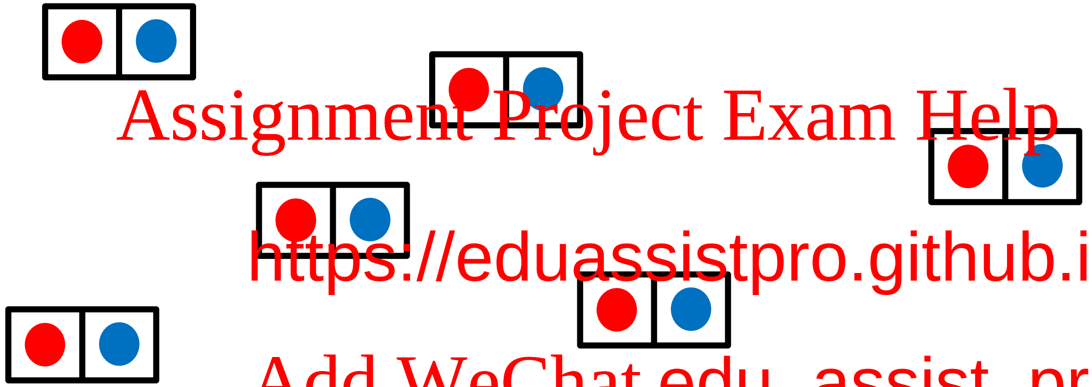
MAP ADT

put(key, value) // Add the entry (key, value) to the map. If the map previously contained an entry with key, the old value is replaced by the specified value.

get(key) // Returns the value to which the specified key is mapped. Why not get(key, value) ?

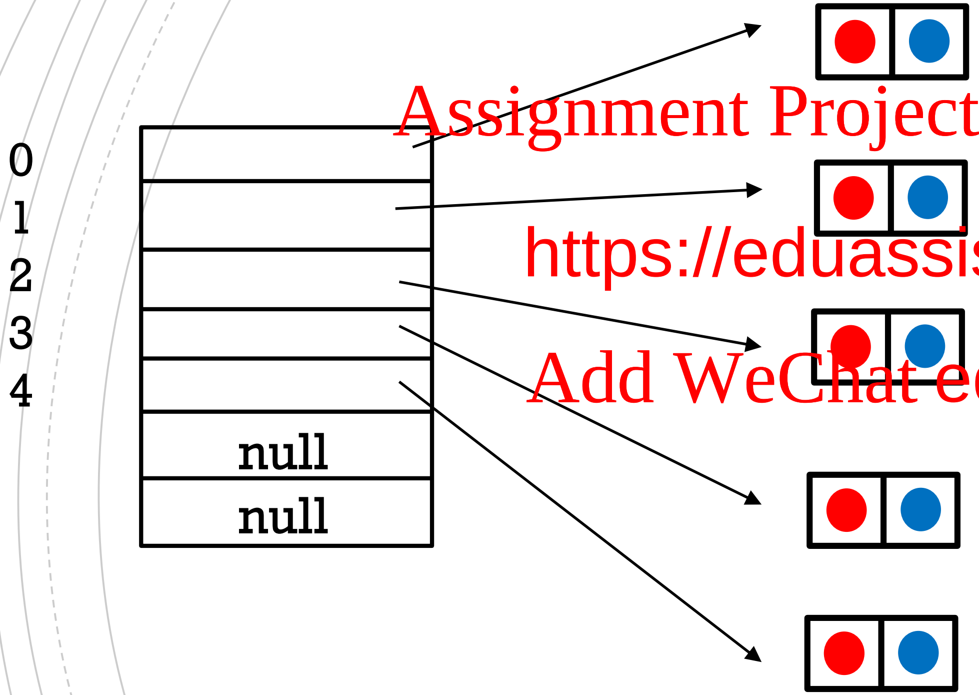
remove(key) // Removes the entry with the specified key. Returns true if the entry was removed, false otherwise.

DATA STRUCTURES FOR MAPS



How to organize a set of (key, value) pairs, i.e. entries ?

ARRAY LIST



How can we implement the following methods?

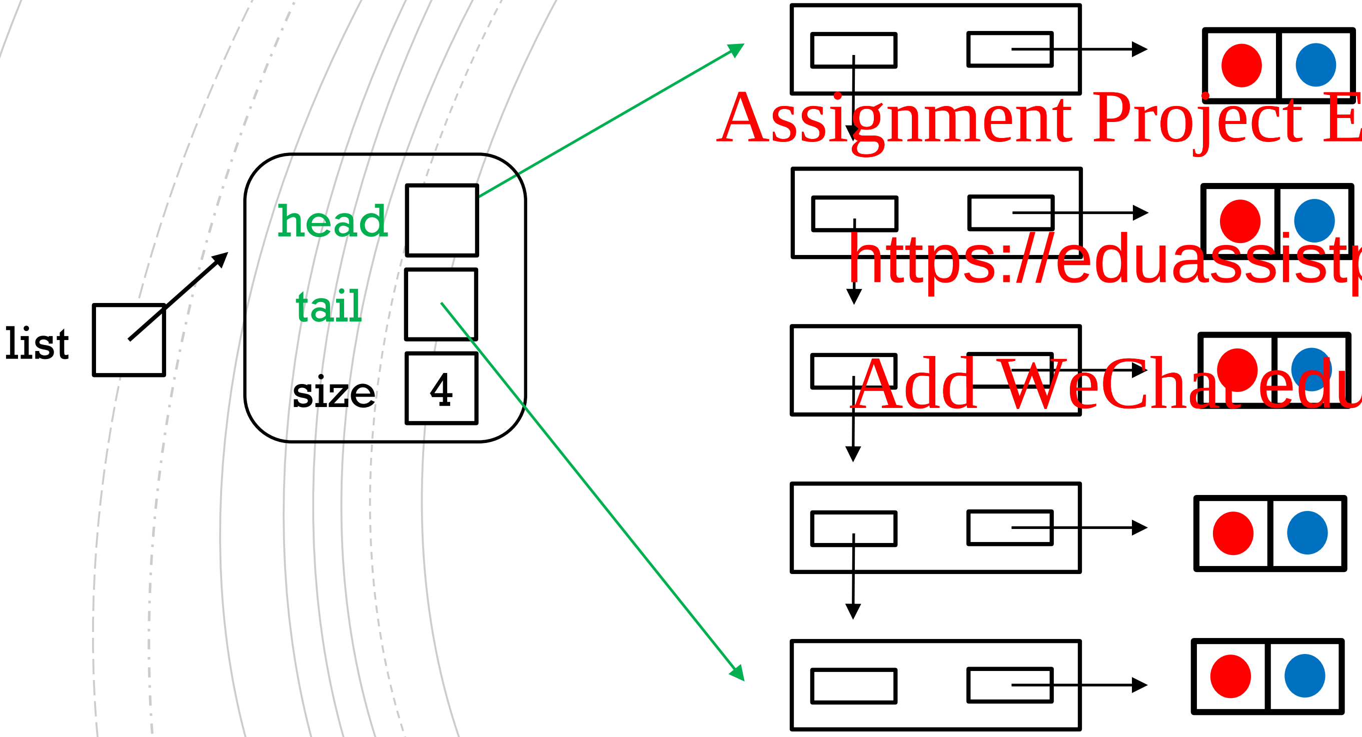
ut(key, value)

et(key)

(key)

put() would be O(1), while get() and remove() would be O(n)

SINGLY (OR DOUBLY) LINKED LIST



How can we implement the following methods?

ut(key, value)

et(key)

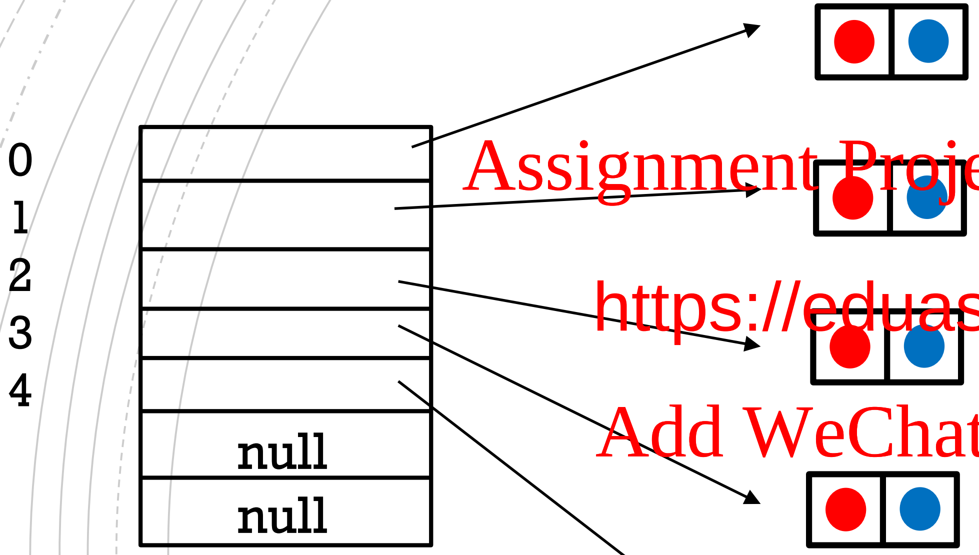
(key)

put() would be O(1), while get() and remove() would be O(n)

LET'S ADD ASSUMPTIONS

- Special case #1: what if keys are *comparable* ?

ARRAY LIST (SORTED BY KEY)



How can we implement the following methods?

(key, value)

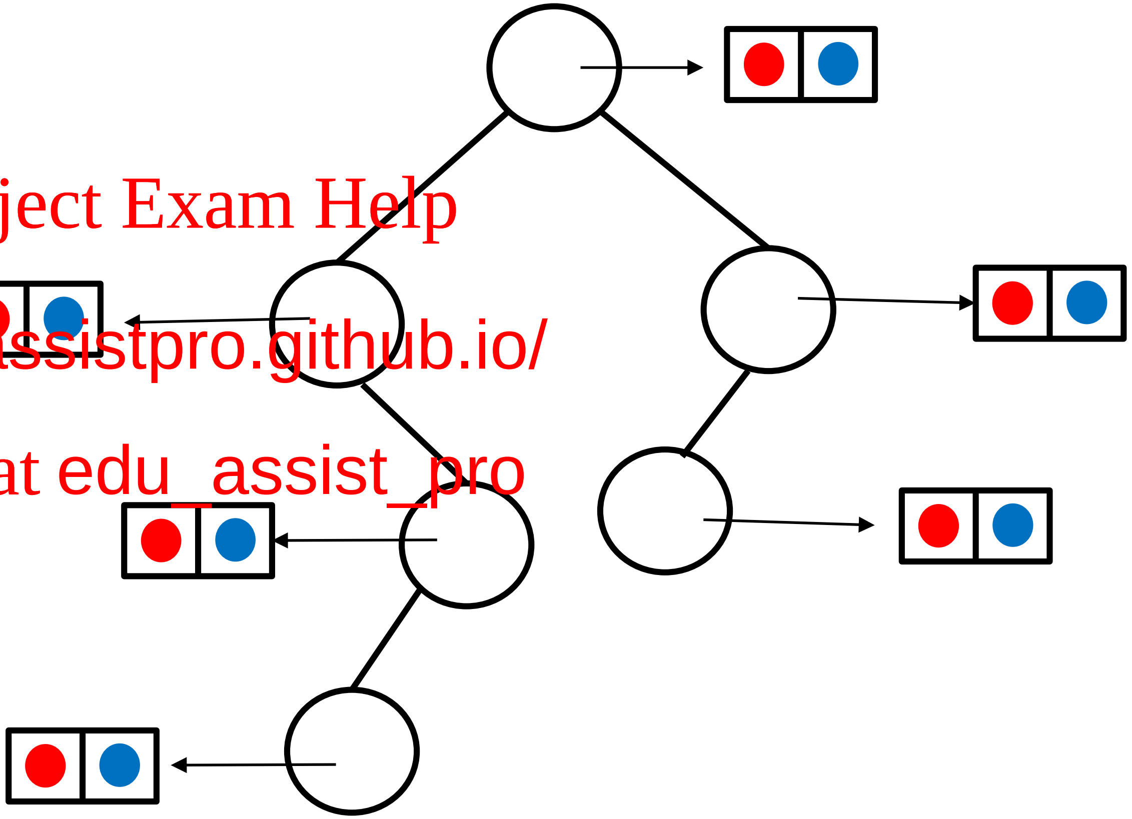
(key)

put() and remove() would be O(n), while get() could be performs in time O(log n) using binary search

BINARY SEARCH TREE (SORTED BY KEY)

How can we implement the following methods?

put(key, value)
get(key)
remove(key)

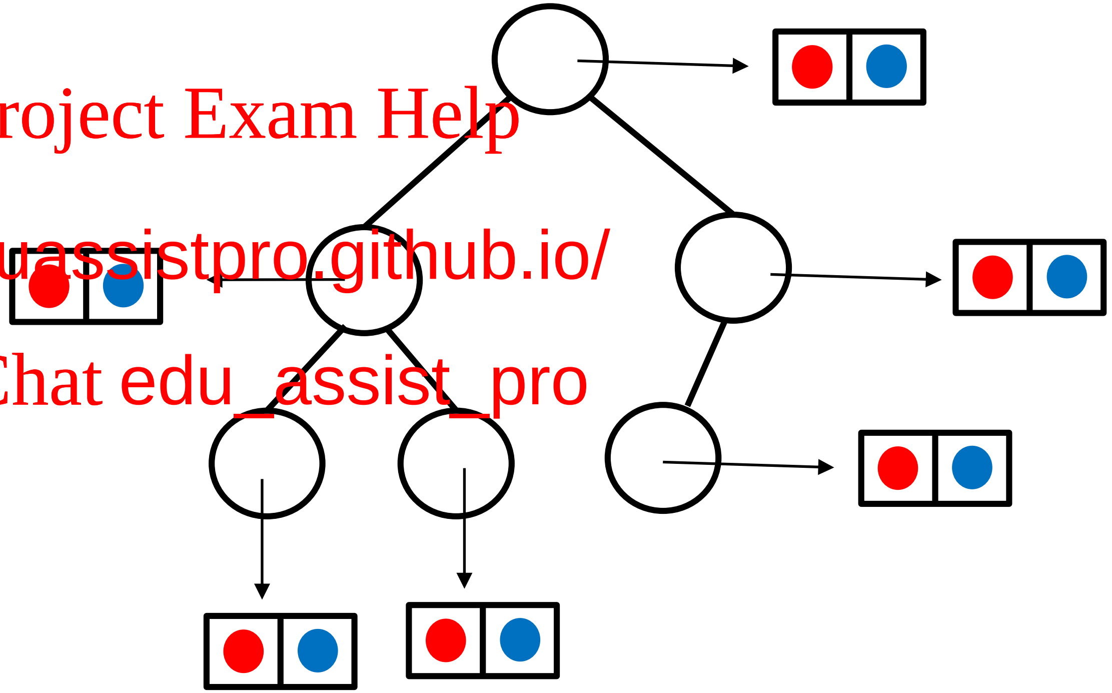


The performance of put(), get() and remove() depends on the tree. If we have a balances tree, then these operations would all take time O(log n) in worst case. You will learn more about balanced tree in COMP 251.

MINHEAP (PRIORITY DEFINED BY KEY)

How can we implement the following methods?

put(key, value)
get(key)
remove(key)



The performance of put() would be O(log n). Implementing get() would require traversing the tree, so it would be O(n). Implementing remove() would be a little weird for heaps…

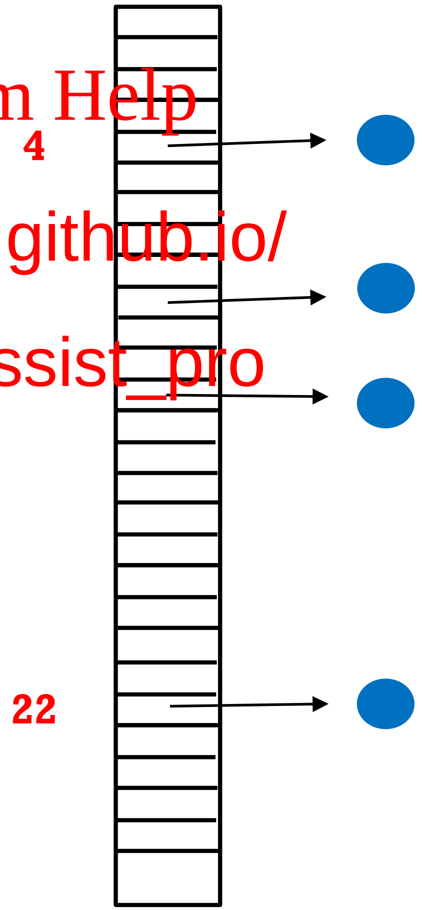
LET'S ADD ASSUMPTIONS

- ~~Special case #1: what if keys are *comparable* ?~~
- Special case #2: what if keys are unique positive integers in small range ?

ARRAYS OF VALUES

Then, we could u type V (value) a access.

This would not work well if keys are 9 digit student IDs.



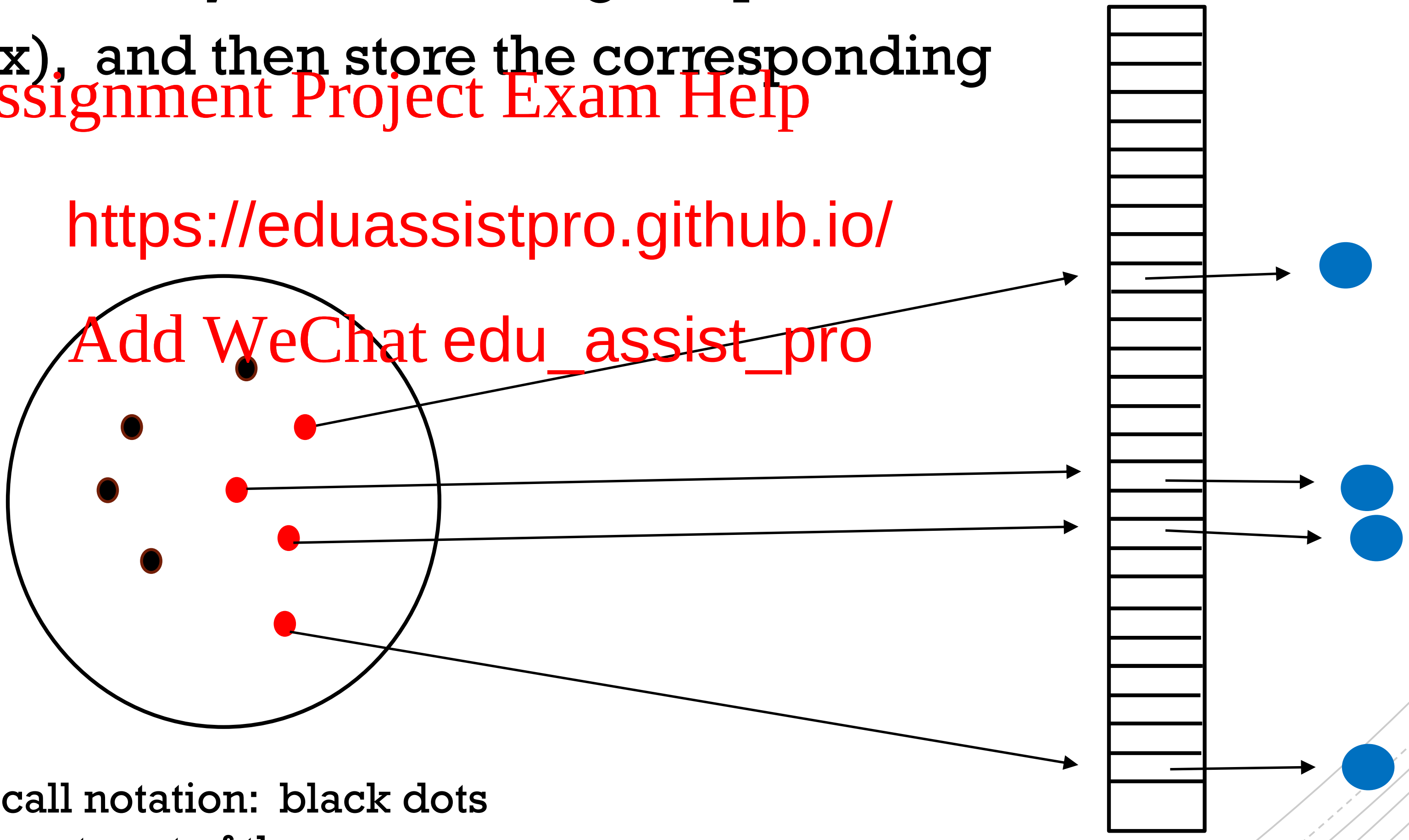
IN GENERAL

- Keys might not be comparable.
- Keys might be not

  e.g. Keys might be strings or so pe.

STRATEGY IN THE GENERAL CASE

Try to define a map from keys to *small* range of positive integers (array index), and then store the corresponding values in the array.



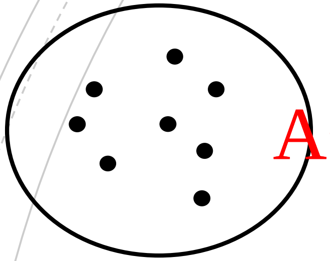
Recall notation: black dots are not part of the map.

IN THIS VIDEO

Define a map from keys to *large* range of positive integers

Such map is called *h*

JAVA'S `Object.hashcode()`



$$\xrightarrow[\text{(not many-to-1)}]{\text{1-to-1}} \{0, 1, 2, ..., 2^{24} - 1\}$$

objects in a Java program (runtime)

o *dress* in JVM memory

(

By default, "obj1 == obj2" means "obj1.hashcode() == obj2.hashcode()"

JAVA'S `String.hashcode()`

https://docs.oracle.com/javase/7/docs/api/java/lang/String.html

EXAMPLE HASH CODE FOR STRINGS

(not used in Java)

$$s.length-1$$

$$(\quad)\qquad\qquad[i]$$

e.g.

$$h("eat") = h("ate") = h("tea")$$

ASCII values of 'a', 'e', 't' are 97, 101, 116.

JAVA'S `String.hashcode()`

$$\texttt{s.hashCode()} \equiv \sum_{i=0}^{s.length-1} s[i] * x^{s.length-1-i}$$

here $x = 31$.

JAVA'S `String.hashcode()`

$$s.\text{hashCode}() \equiv \sum_{i=0}^{s.length-1} s[i]\, x^{s.length-1-i}$$

here $x = 31.$

e.g. `s = "eat"` **then** `s.hashcode()` $= 101 * 31^2 + 97 * 31 + 116$

`'e'`	`'a'`	`'t'`
`s[0]`	`s[1]`	`s[2]`

JAVA'S `String.hashcode()`

$$\text{s.hashCode()} \equiv \sum_{i=0}^{s.length-1} s[i]\, x^{s.length-1-i}$$

here $x = 31$.

e.g. `s = "ate"` then `s.hashcode()` $= 97 * 31^2 + 116 * 31 + 101$

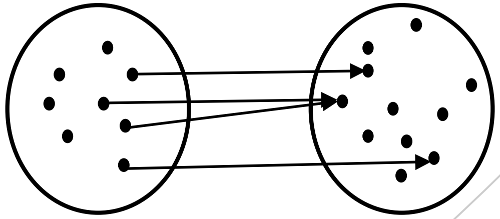
`'a'`	`'t'`	`'e'`
`s[0]`	`s[1]`	`s[2]`

JAVA'S `String.hashcode()`

$$\texttt{s.hashCode()} \equiv \sum_{i=0}^{s.length-1} s[i] * (31)^{s.length-1-i}$$

If `s1.hashCode() == s2.` `()` **then what can we conclude about** `s1.equals(s2)` **?**

`s1` ***may or may not*** **be the same string as** `s2`**.**

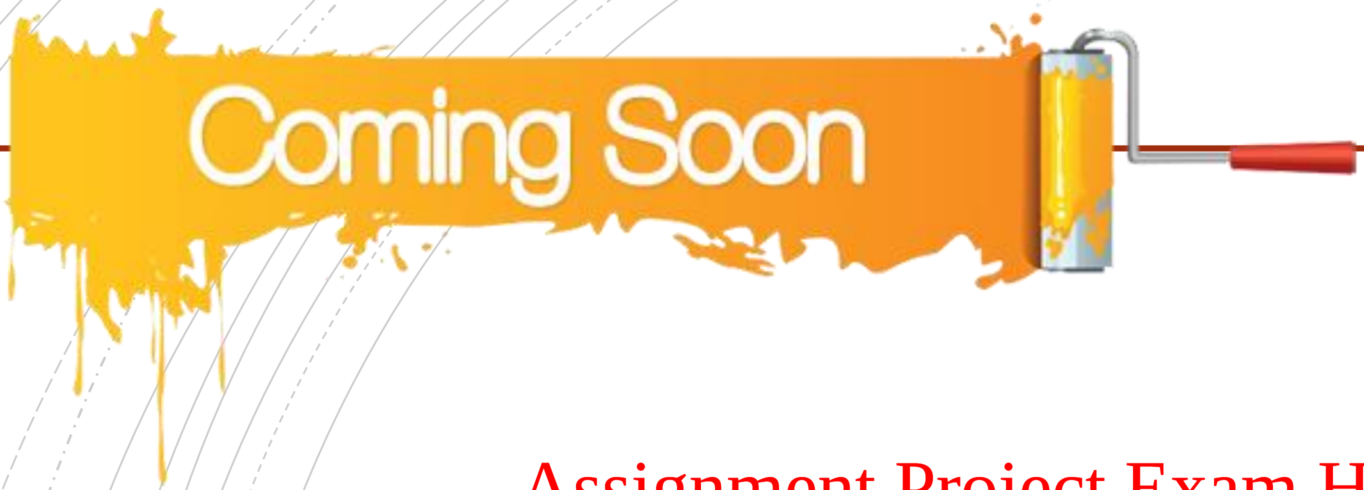


JAVA'S `String.hashcode()`

$$\texttt{s.hashCode()} \equiv \sum_{i=0}^{s.length-1} s[i] * (31)^{s.length-1-i}$$

If `s1.hashCode() != s2.` `()` **then what can we conclude about** `s1.equals(s2)` **?**

`s1` **is a different string then** `s2`**.**



In the next

- Hash Ma