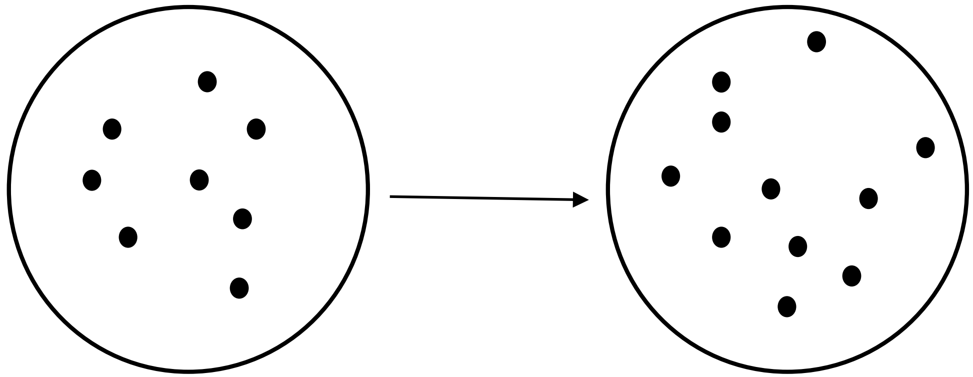
COMP 250

Lecture 25

maps

Nov. 7, 2016

Map (Mathematics)



“domain” “codomain”

A map is a set of pairs { (x, f(x)) }.
The set of { f(x) } is called the “range”.
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

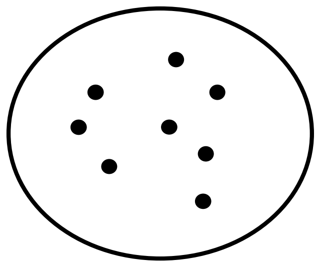
Calculus 1 and 2 (“functions”):

f : real numbers → real numbers

Asymptotic complexity in CS:

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

Object.hashcode() map in Java

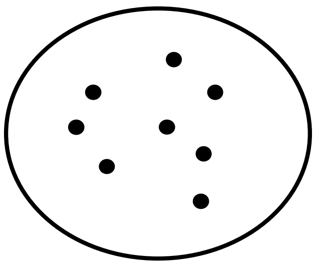


$\xrightarrow[\text{(not many-to-1)}]{\text{1-to-1}}$ $\{ 0, 1, 2, \ldots, 2^{24} - 1\}$

objects in a Java program (runtime)

object's *base address* in JVM memory (24 bits)

Object.hashcode() map in Java



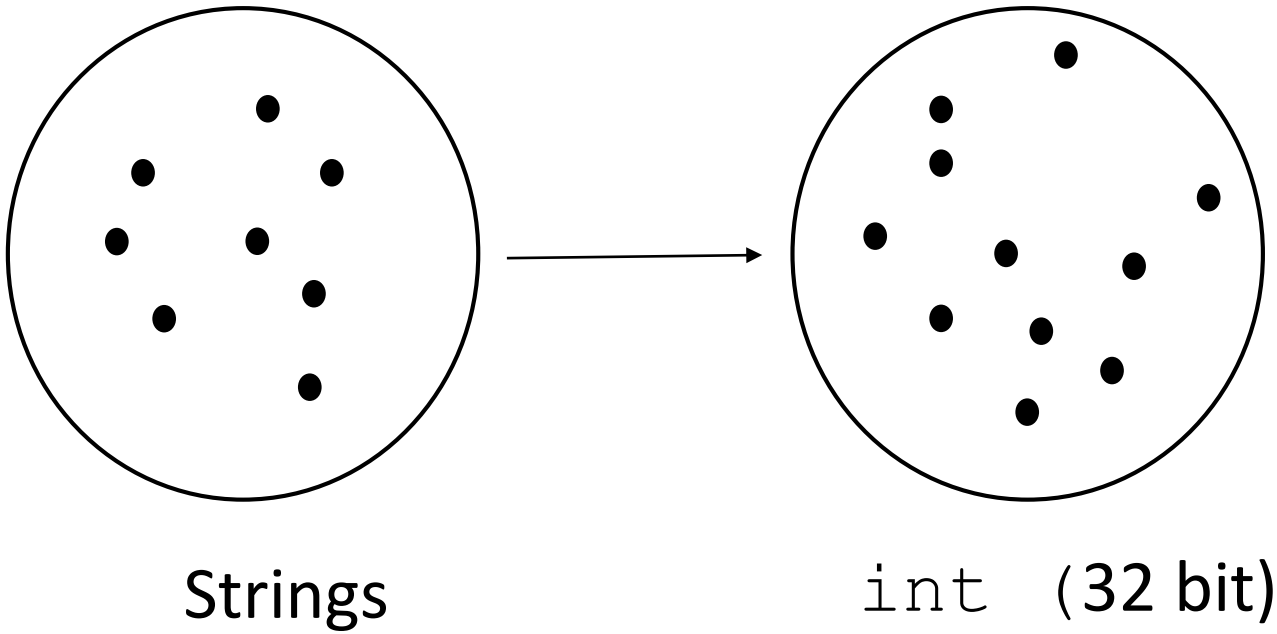
$$\xrightarrow[\text{(not many-to-1)}]{\text{1-to-1}} \{\,0, 1, 2, \ldots, 2^{24} - 1\}$$

objects in a Java program (runtime)

object's *base address* in JVM memory (24 bits)

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

String.hashcode() in Java



For each String, define an integer.

Example hash code for Strings

(not used in Java)

$$h(s) \equiv \sum_{i=0}^{s.length-1} s[i]$$

e.g.

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

ASCII and UNICODE values of ‘a’, ‘e’, ‘t’ are 97, 101, 116.

String.hashcode() in Java

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

where x = 31 and using `int` arithmetic.

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

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

String.hashcode() in Java

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

If s1.hashCode() == s2.hashCode() then ... ?

If s1.hashCode() != s2.hashCode() then ... ?

ASIDE: Use Horner's rule for efficient polynomial evaluation

s[0] * x^3 + s[1] * x^2 + s[2]* x + s[3]

There is no need to compute each x^i separately.

ASIDE: Use Horner's rule for efficient polynomial evaluation

s[0] * 31^3 + s[1] * 31^2 + s[2]* 31 + s[3]

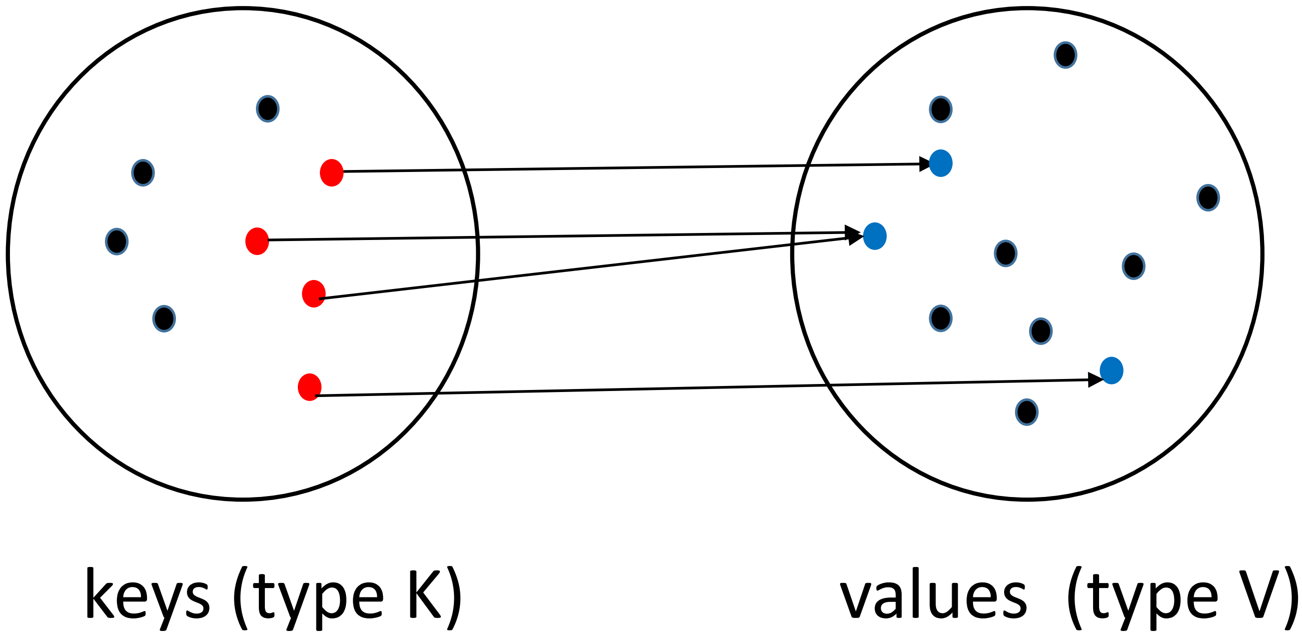
= (s[0] * 31^2 + s[1] * 31^1 + s[2]) * 31 + s[3]

= ((s[0] * 31^1 + s[1]) $*$ 31 + s[2]) * 31 + s[3]

```
h = 0
for (i = 0; i < s.length; i++)
    h = h*31 + s[i]
```

For a degree n polynomial, Horner's rule uses O(n) multiplications, not O(n^2).

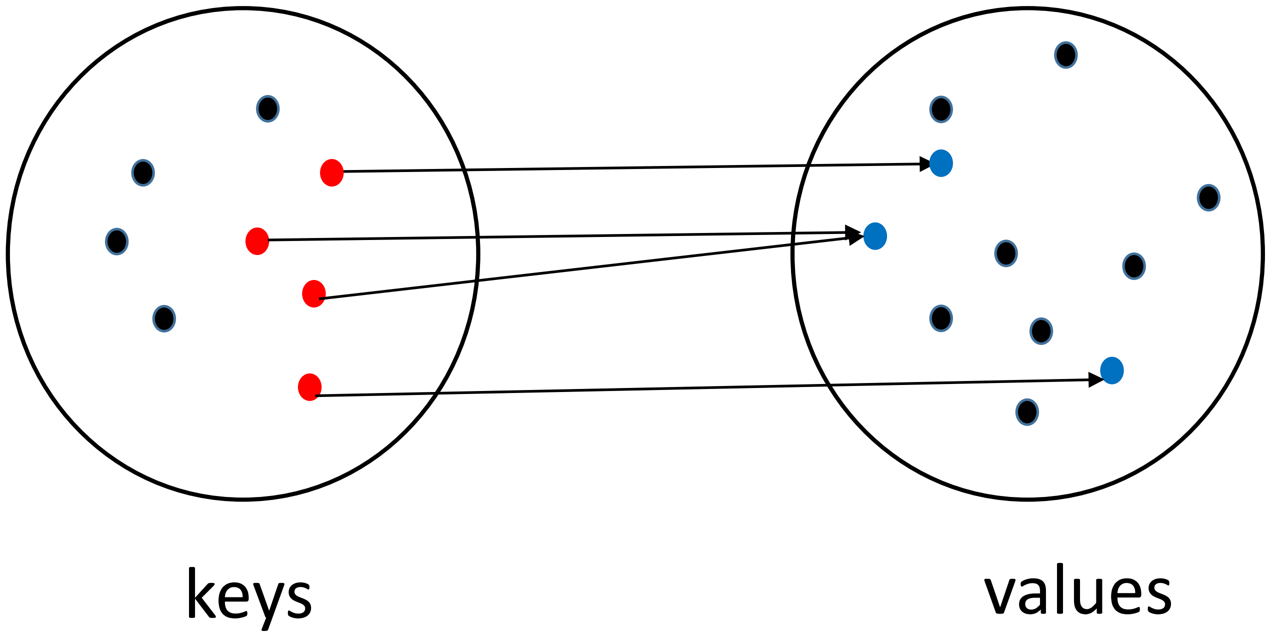
Map (in COMP 250)



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

Map	Keys	Values
Address book	Name	Address, email..
Caller ID	Phone #	Name
Student file	ID or Name	Student record
Index at back of book	keyword	List of book pages

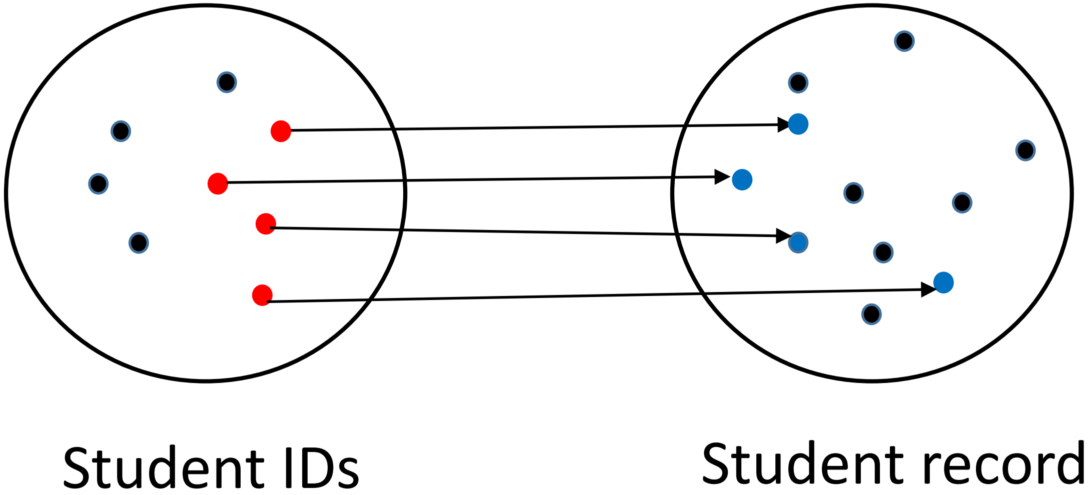
Map Entries



Each (key, value) pair is called an *entry*.

In this example, there are four entries.

Example



In COMP 250 this semester, the above mapping has over 400 entries.

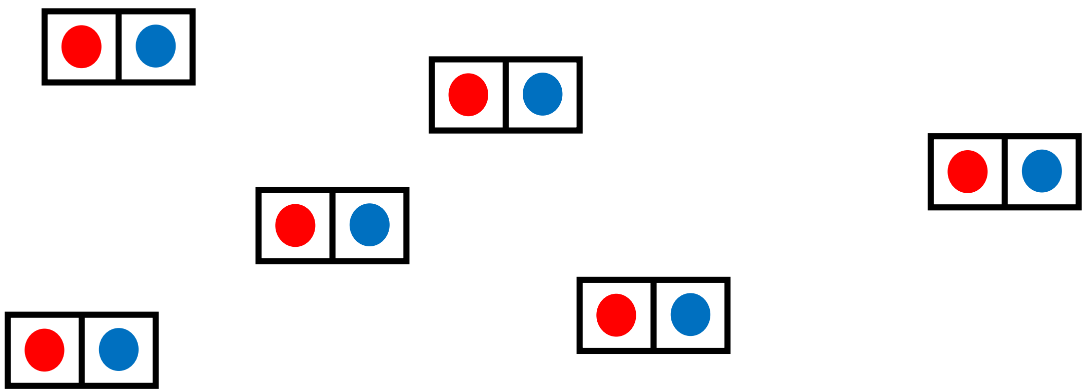
Most McGill students are not taking 250 this semester.

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

Map ADT

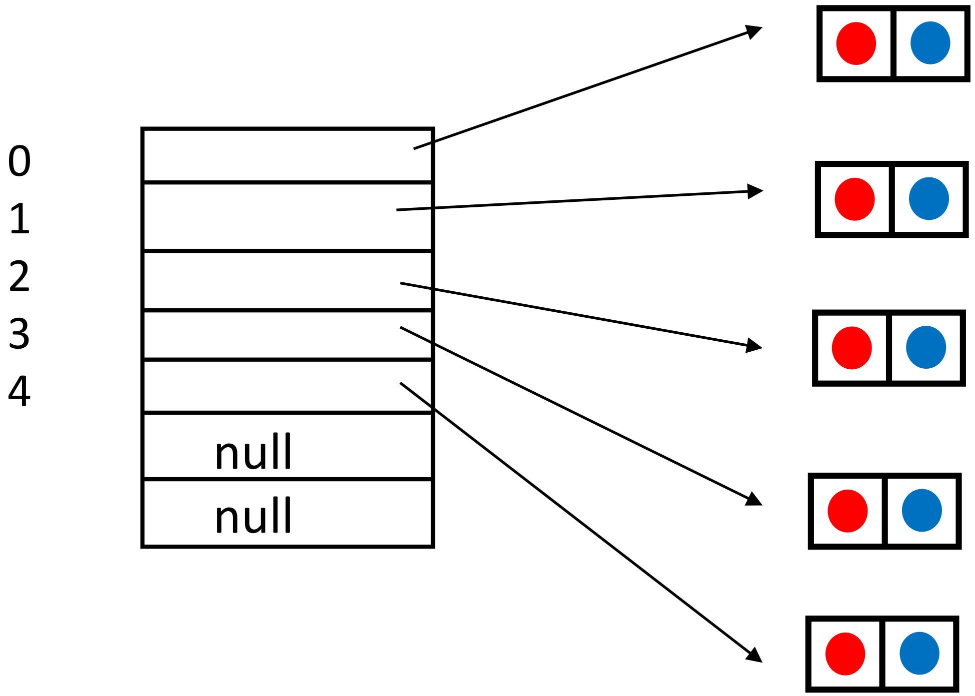
- put(key, value)   // add
- get(key)   //   why not get(key, value) ?
- remove(key)
- …

Data Structures for Maps

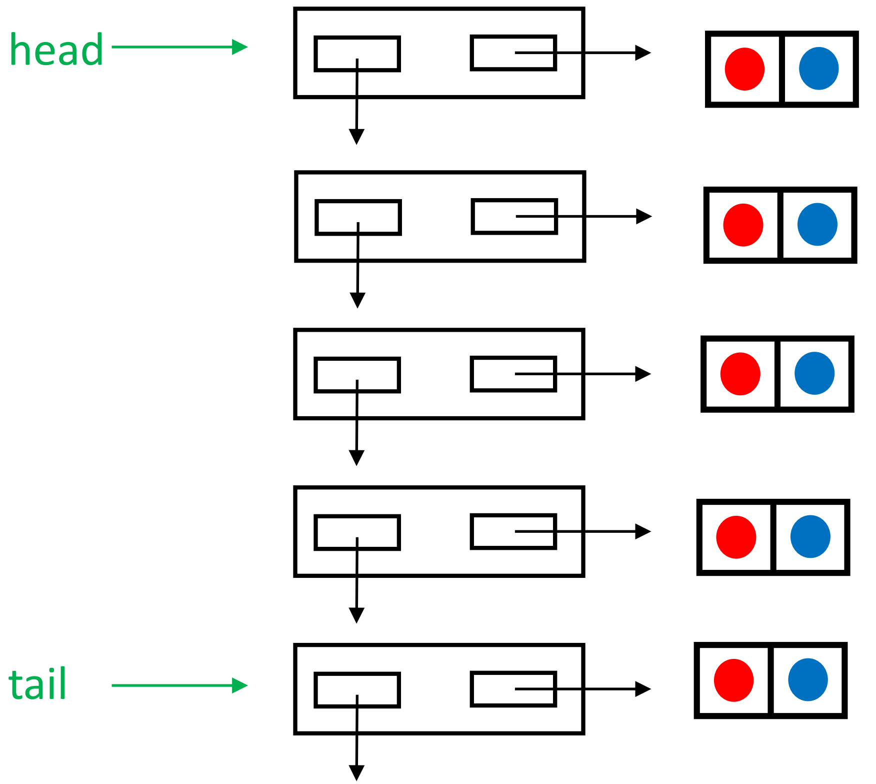


How to organize a set of (key, value) pairs, i.e. entries ?

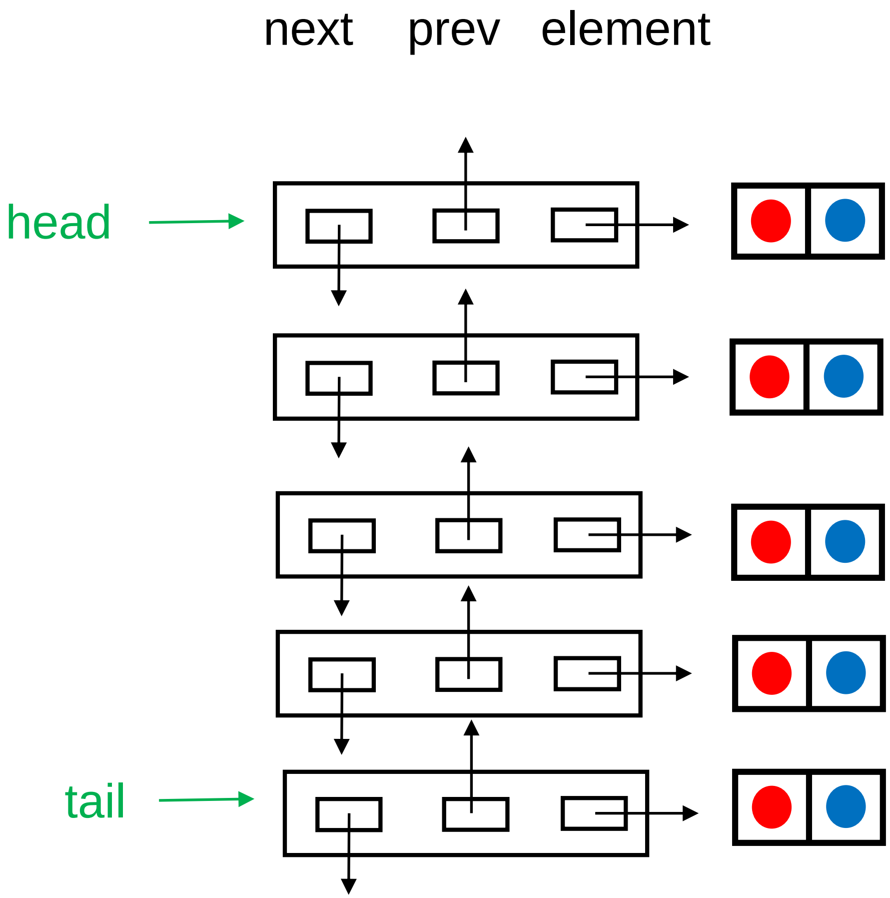
Array list



Singly linked list



Doubly linked list



Assumptions about keys

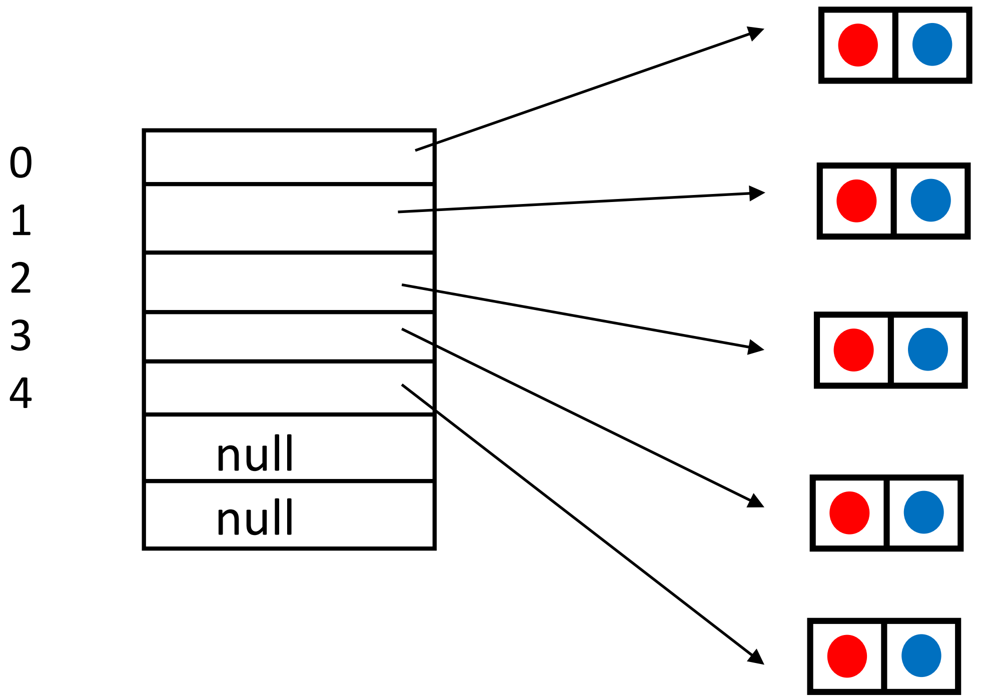
Can two keys have the same value ? *Yes.*

Can one key have two values ? *No.*

Special case #1: what if keys are comparable ?

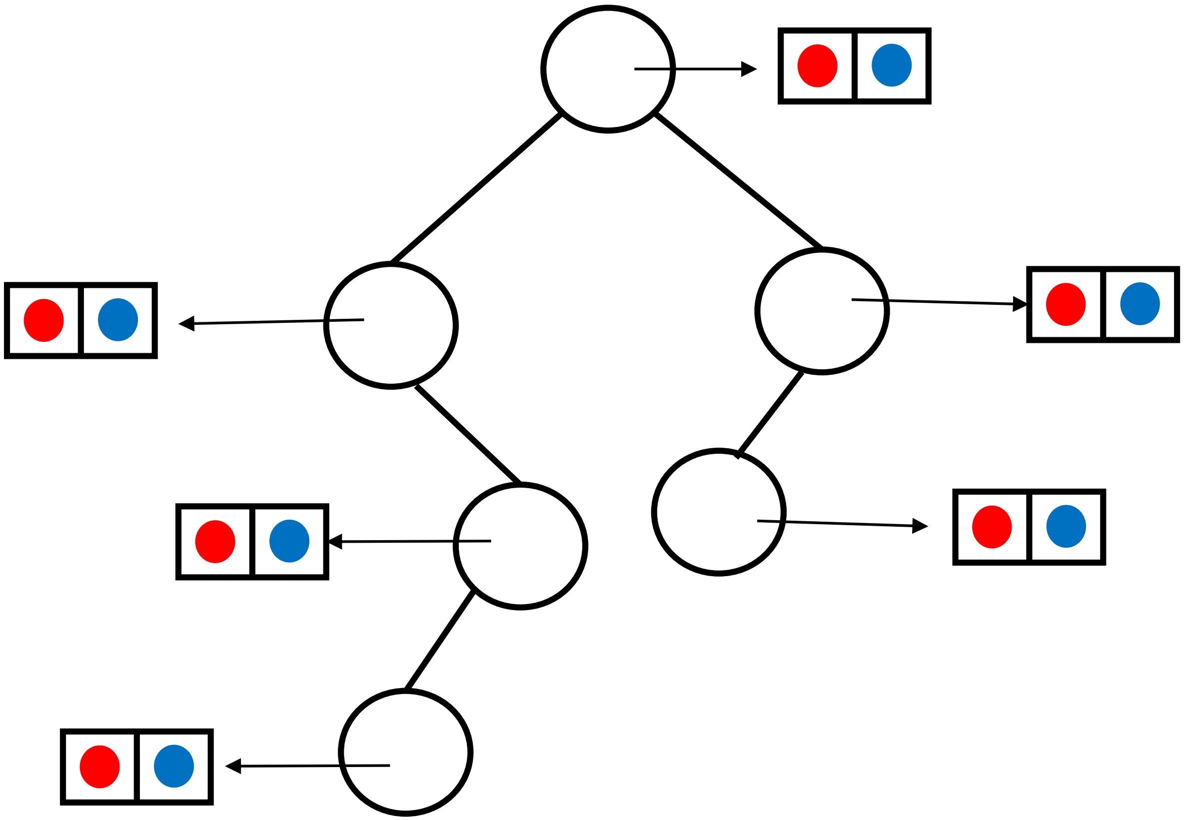
Next few slides consider this case.

Array list (sorted by key)



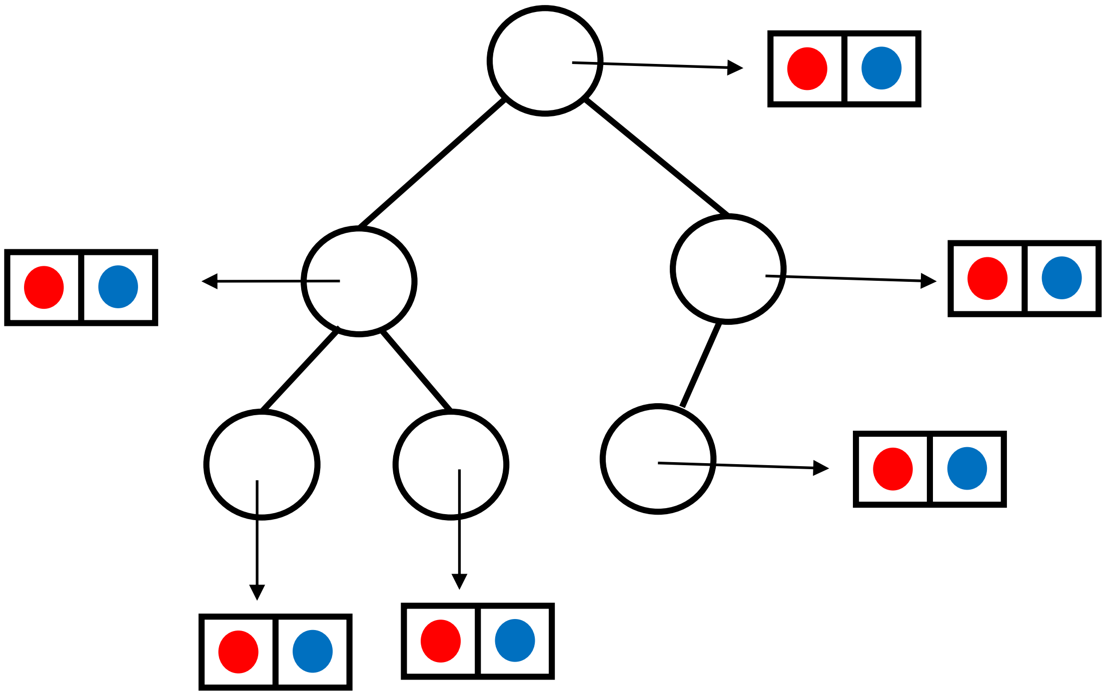
What is the O() bound for get(key), put(key, value), remove(key) ?

Binary Search Tree (sorted by key)



What is the O() bound for get(key), put(key, value), remove(key) ?

Heap (priority defined by key)



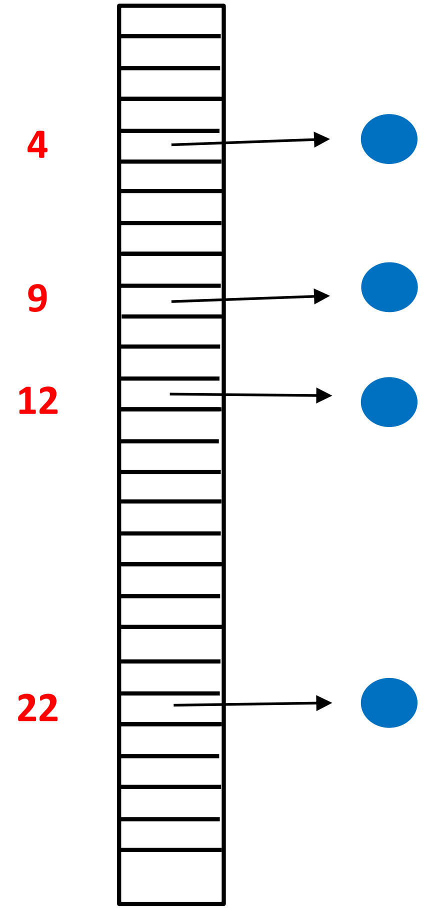
What is O() bound for put() and removeMin() ?

What is O() bound for get(key) and remove(key)?

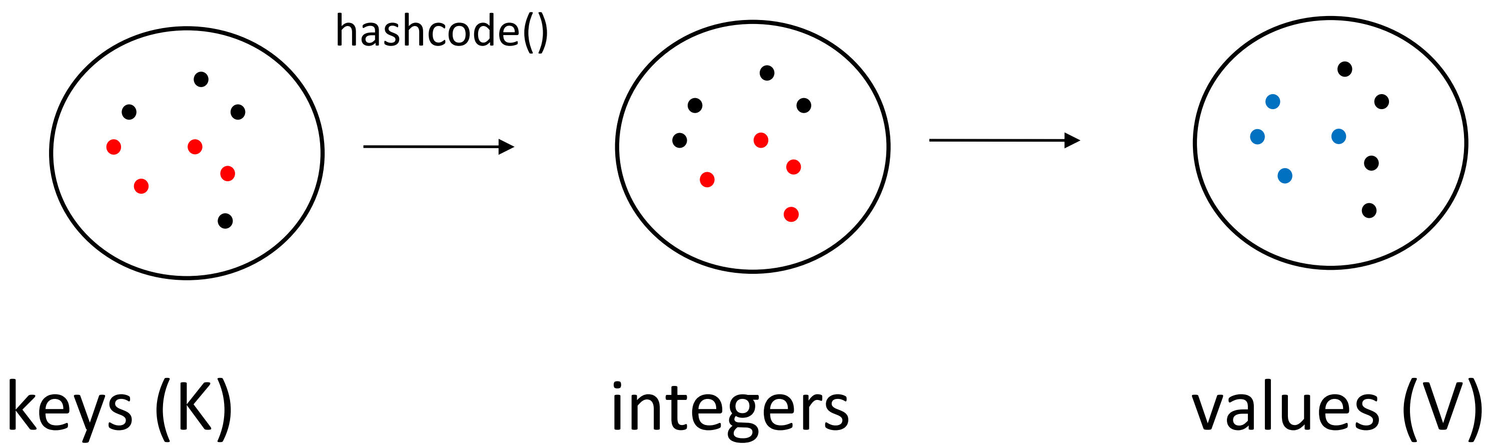
Special case #2: what if keys were unique positive integers in small range ?

Then, we could use an array of type V (value) and have O(1) access.

This would not work well for 9 digit student IDs



Next lecture: hash maps



Somehow we want to map the keys to a small range of positive integers.

How to make this work?