

Data Structures

Maarten Dhondt

Realdolmen

June 23, 2017



Who am I?

- ▶ Master of Engineering: Computer Science (KUL)
 - ▶ Computational informatics
- ▶ Realdolmen: acADDemICT in 09/2015
- ▶ Current project: Planning infrastructure @ Infrabel



Outline

1 Introductory Data Structures

- Array
- Linked List
- Hash Table
- Tree
 - Heap
 - Binary Search Tree
 - Red-Black Tree

2 Java Collection API & Map API

- Java Collection API
- Java Map API

3 Advanced Data Structures

- Skip list
- Bloom Filter
- van Emde Boas Tree



Outline

1 Introductory Data Structures

- Array
- Linked List
- Hash Table
- Tree

2 Java Collection API & Map API

- Java Collection API
- Java Map API

3 Advanced Data Structures

- Skip list
- Bloom Filter
- van Emde Boas Tree



What are Data Structures?

Data Structure¹

A way in which data are stored for efficient search and retrieval.
Different data structures are suited for different problems.

- ▶ Data type \neq data structure
- ▶ `java.util.HashSet` vs. hash table
- ▶ array vs. array

¹Encyclopædia Britannica

Outline

1 Introductory Data Structures

Array
Linked List
Hash Table
Tree

2 Java Collection API & Map API

Java Collection API
Java Map API

3 Advanced Data Structures

Skip list
Bloom Filter
van Emde Boas Tree



Array

Definition

- ▶ An indexed set of related elements.²
 - ▶ An assemblage of items that are randomly accessible by integers, the index.³
-
- ▶ Example: linear array



² Oxford Dictionary

³ National Institute of Standards & Technology

Array

Operations

- ▶ `get`
- ▶ `set`
- ▶ `indexOf`



Array

Operations

- ▶ `get`
- ▶ `set`
- ▶ `indexOf`



`get(1)`

Array

Operations

- ▶ `get`
- ▶ `set`
- ▶ `indexOf`



`get(1)`

Array

Operations

- ▶ `get` $O(1)$
- ▶ `set`
- ▶ `indexOf`



`get(1)`

Array

Operations

- ▶ `get` $O(1)$
- ▶ `set`
- ▶ `indexOf`



`set(2)`

Array

Operations

- ▶ `get` $O(1)$
- ▶ `set`
- ▶ `indexOf`



`set(2)`

Array

Operations

- ▶ `get` $O(1)$
- ▶ `set` $O(1)$
- ▶ `indexOf`



`set(2)`

Array

Operations

- ▶ `get` $O(1)$
- ▶ `set` $O(1)$
- ▶ `indexOf`



`indexOf(object)`

Array

Operations

- ▶ `get` $O(1)$
- ▶ `set` $O(1)$
- ▶ `indexOf`



`indexOf(object)`

Array

Operations

- ▶ `get` $O(1)$
- ▶ `set` $O(1)$
- ▶ `indexOf`



`indexOf(object)`

Array

Operations

- ▶ `get` $O(1)$
- ▶ `set` $O(1)$
- ▶ `indexOf`



`indexOf(object)`

Array

Operations

- ▶ `get` $O(1)$
- ▶ `set` $O(1)$
- ▶ `indexOf`



`indexOf(object)`

Array

Operations

- ▶ `get` $O(1)$
- ▶ `set` $O(1)$
- ▶ `indexOf`



`indexOf(object)`

Array

Operations

- ▶ `get` $O(1)$
- ▶ `set` $O(1)$
- ▶ `indexOf` $O(n)$



Outline

1 Introductory Data Structures

Array
Linked List
Hash Table
Tree

2 Java Collection API & Map API

Java Collection API
Java Map API

3 Advanced Data Structures

Skip list
Bloom Filter
van Emde Boas Tree



Linked List

Definition

A linked list is a data structure in which the objects are arranged in a linear order. Unlike arrays in which the linear order is determined by indices, the order is determined by a pointer in each object.⁴

- ▶ Different types: singly, doubly, multiply, circular, ...
- ▶ Example: doubly linked list



⁴ Introduction to Algorithms By Cormen, Leieron, Rivest & Stein

Linked List

Operations

- ▶ add/remove first/last
- ▶ get/insertAt
- ▶ indexOf



Linked List

Operations

- ▶ add/remove first/last
- ▶ get/insertAt
- ▶ indexOf

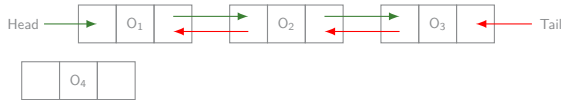


`addFirst(O_4)`

Linked List

Operations

- ▶ add/remove first/last
- ▶ get/insertAt
- ▶ indexOf

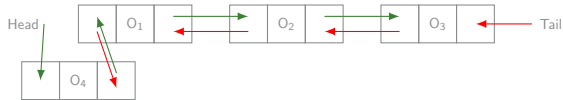


`addFirst(O4)`

Linked List

Operations

- ▶ add/remove first/last
- ▶ get/insertAt
- ▶ indexOf

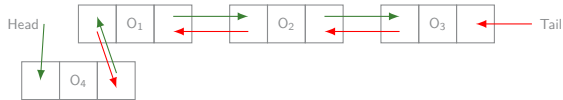


`addFirst(O4)`

Linked List

Operations

- ▶ add/remove first/last $O(1)$
- ▶ get/insertAt
- ▶ indexOf



`addFirst(O4)`

Linked List

Operations

- ▶ add/remove first/last $O(1)$
- ▶ get/insertAt
- ▶ indexOf

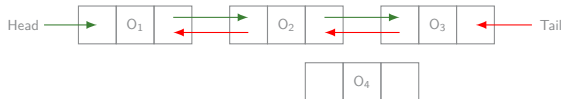


insertAt(2)

Linked List

Operations

- ▶ add/remove first/last $O(1)$
- ▶ get/insertAt
- ▶ indexOf

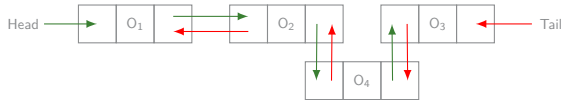


insertAt(2)

Linked List

Operations

- ▶ add/remove first/last $O(1)$
- ▶ get/insertAt
- ▶ indexOf

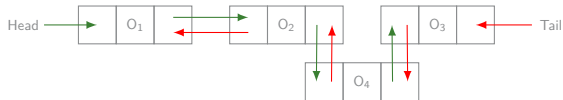


insertAt(2)

Linked List

Operations

- ▶ add/remove first/last $O(1)$
- ▶ get/insertAt $O(n)$
- ▶ indexOf



insertAt(2)

Linked List

Operations

- ▶ add/remove first/last $O(1)$
- ▶ get/insertAt $O(n)$
- ▶ indexOf



indexOf(O₂)



Linked List

Operations

- ▶ add/remove first/last $O(1)$
- ▶ get/insertAt $O(n)$
- ▶ indexOf

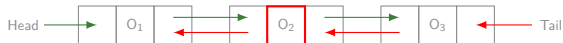


`indexOf(O_2)`

Linked List

Operations

- ▶ add/remove first/last $O(1)$
- ▶ get/insertAt $O(n)$
- ▶ indexOf



`indexOf(O2)`

Linked List

Operations

- ▶ add/remove first/last $O(1)$
- ▶ get/insertAt $O(n)$
- ▶ indexOf $O(n)$



indexOf(O₂)



Linked List

Operations

- ▶ add/remove first/last $O(1)$
- ▶ get/insertAt $O(n)$
- ▶ indexOf $O(n)$



Outline

1 Introductory Data Structures

Array
Linked List
Hash Table
Tree

2 Java Collection API & Map API

Java Collection API
Java Map API

3 Advanced Data Structures

Skip list
Bloom Filter
van Emde Boas Tree

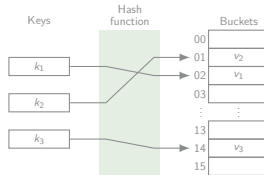


Hash Table

Definition

A dictionary in which keys are mapped to array positions by hash functions.⁵

- ▶ Hash functions: determinism, uniformity, defined range, data normalisation, non-invertible, perfect, . . .
- ▶ Collisions resolution: chaining, open addressing, . . .
- ▶ Example:

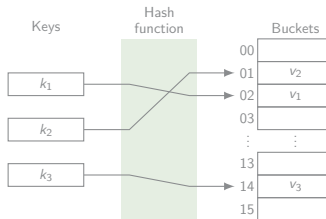


⁵ National Institute of Standards & Technology

Hash Table

Operations

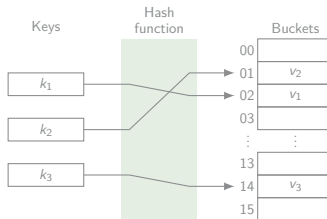
- put
- remove
- get



Hash Table

Operations

- put
- remove
- get

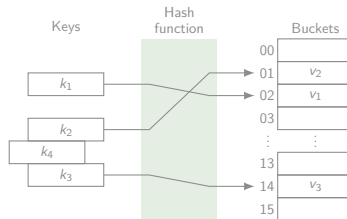


put(0₄)

Hash Table

Operations

- put
- remove
- get

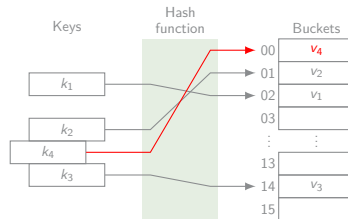


put(04)

Hash Table

Operations

- put
- remove
- get

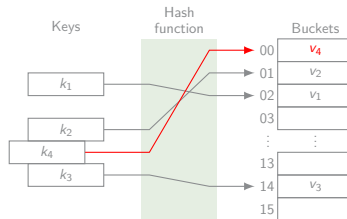


put(04)

Hash Table

Operations

- put $O(1) / O(n)$
- remove
- get

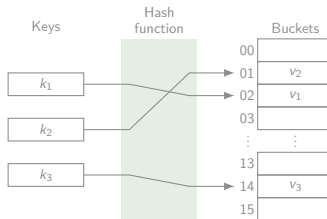


put(0_4)

Hash Table

Operations

- put $O(1) / O(n)$
- remove
- get

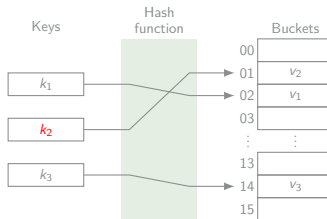


`remove(02)`

Hash Table

Operations

- ▶ put $O(1) / O(n)$
- ▶ remove
- ▶ get

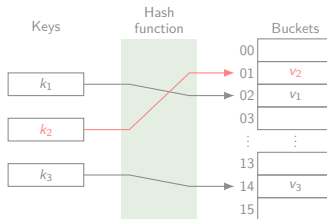


`remove(02)`

Hash Table

Operations

- put $O(1) / O(n)$
- remove
- get

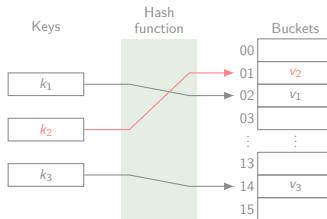


`remove(02)`

Hash Table

Operations

- put $O(1) / O(n)$
- remove $O(1) / O(n)$
- get

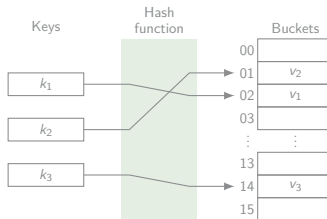


`remove(02)`

Hash Table

Operations

- put $O(1) / O(n)$
- remove $O(1) / O(n)$
- get

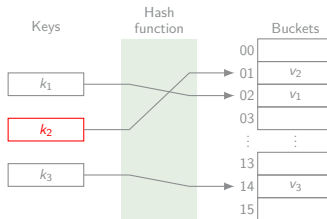


get(0_2)

Hash Table

Operations

- put $O(1) / O(n)$
- remove $O(1) / O(n)$
- get

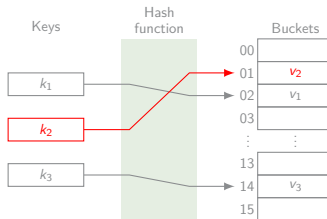


get(0₂)

Hash Table

Operations

- put $O(1) / O(n)$
- remove $O(1) / O(n)$
- get

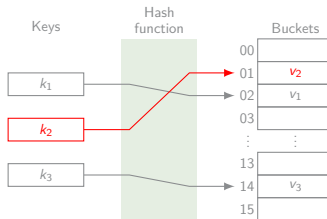


get(0_2)

Hash Table

Operations

- put $O(1) / O(n)$
- remove $O(1) / O(n)$
- get $O(1) / O(n)$

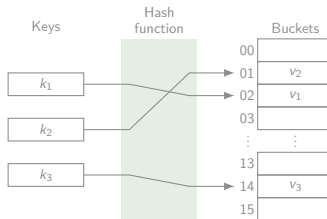


get(0_2)

Hash Table

Operations

- put $O(1) / O(n)$
- remove $O(1) / O(n)$
- get $O(1) / O(n)$



Outline

1 Introductory Data Structures

- Array
- Linked List
- Hash Table
- Tree
 - Heap
 - Binary Search Tree
 - Red-Black Tree

2 Java Collection API & Map API

- Java Collection API
- Java Map API

3 Advanced Data Structures

- Skip list
- Bloom Filter
- van Emde Boas Tree



Tree

Definition

A data structure made up of nodes or vertices and edges without having any cycle. A tree that is not empty consists of a root node and potentially many levels of additional nodes that form a hierarchy.

- ▶ Depth, binary, (nearly) complete, ...
- ▶ Example:

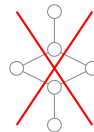
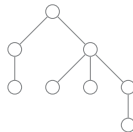


Tree

Definition

A data structure made up of nodes or vertices and edges without having any cycle. A tree that is not empty consists of a root node and potentially many levels of additional nodes that form a hierarchy.

- ▶ Depth, binary, (nearly) complete, ...
- ▶ Example:



Binary Heap

Definition (Heap)

A complete tree where every node has a key more extreme (greater or less) than or equal to the key of its parent.⁶

Definition (Binary Heap)

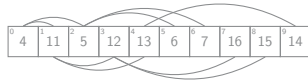
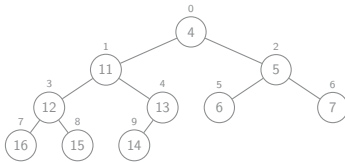
A binary heap data structure is an array object that we can view as a nearly complete binary tree that satisfies the min-heap or max-heap property.⁷

⁶ National Institute of Standards & Technology

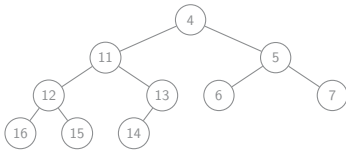
⁷ Introduction to Algorithms By Cormen, Leieron, Rivest & Stein

Binary Min-Heap

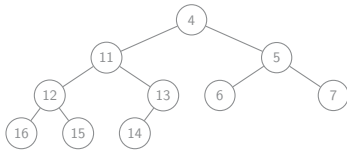
- ▶ $\text{Parent}(n) \quad \lfloor \frac{n-1}{2} \rfloor$
- ▶ $\text{Left}(n) \quad 2n + 1$
- ▶ $\text{Right}(n) \quad 2(n + 1)$



Binary Min-Heap



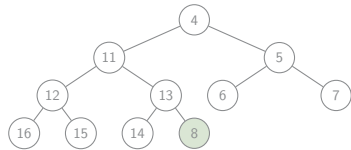
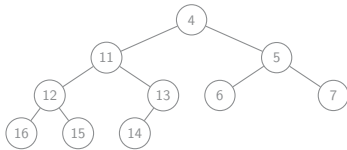
Binary Min-Heap



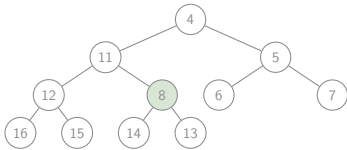
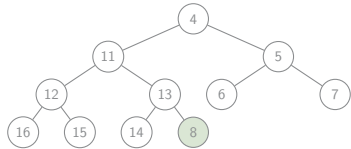
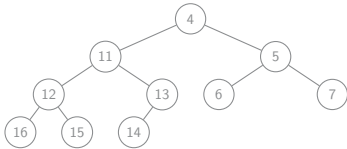
add 8



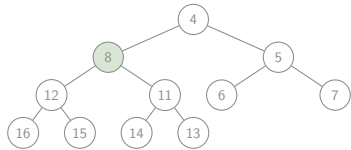
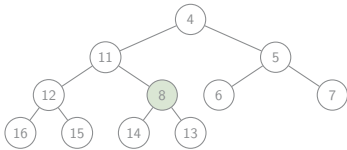
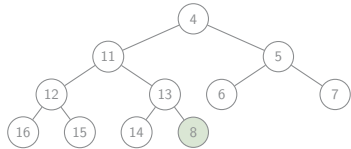
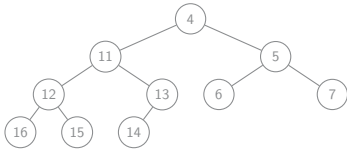
Binary Min-Heap



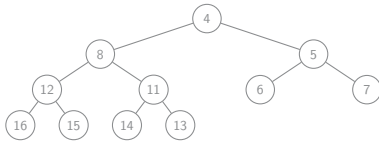
Binary Min-Heap



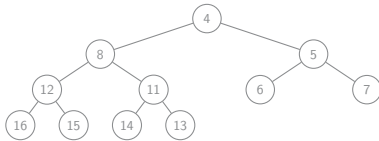
Binary Min-Heap



Binary Min-Heap



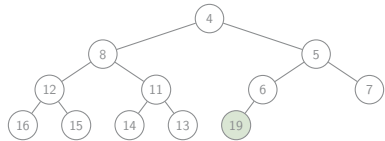
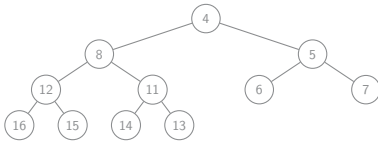
Binary Min-Heap



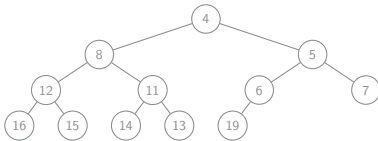
add 19



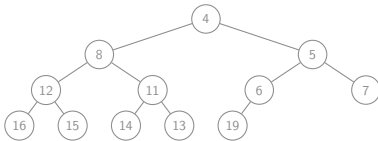
Binary Min-Heap



Binary Min-Heap



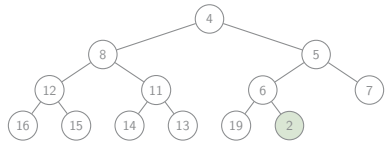
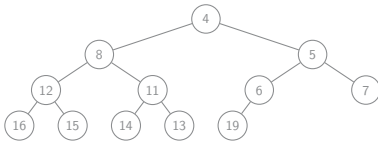
Binary Min-Heap



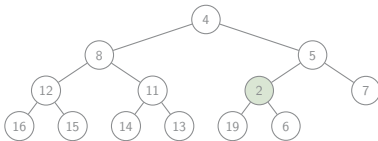
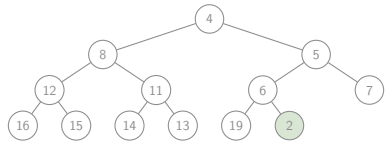
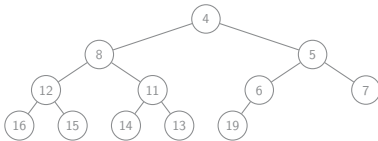
add 2



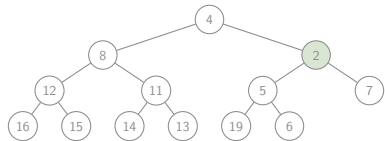
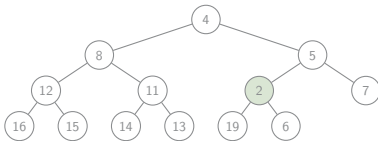
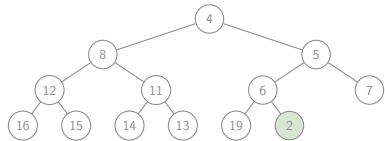
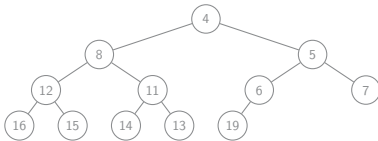
Binary Min-Heap



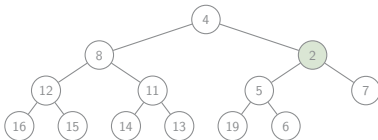
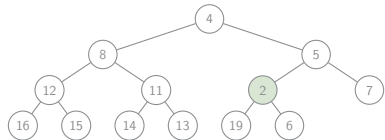
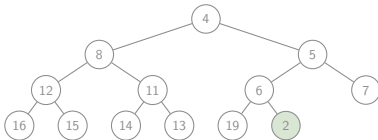
Binary Min-Heap



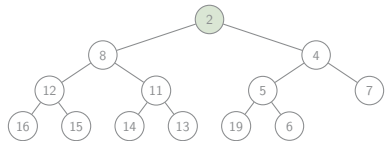
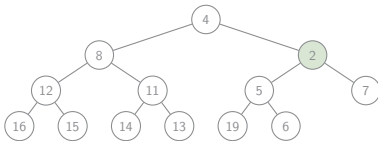
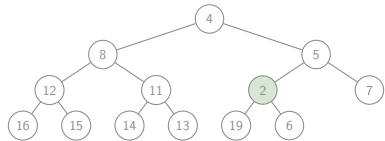
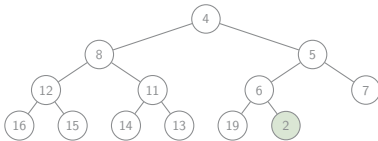
Binary Min-Heap



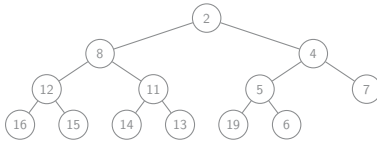
Binary Min-Heap



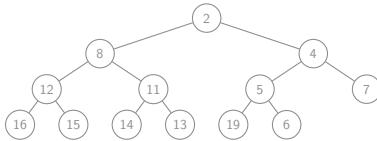
Binary Min-Heap



Binary Min-Heap



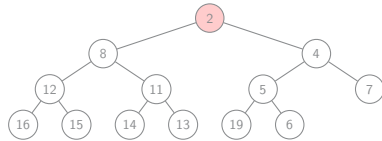
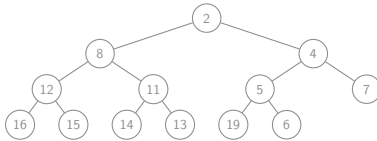
Binary Min-Heap



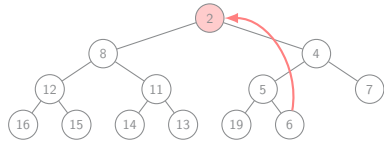
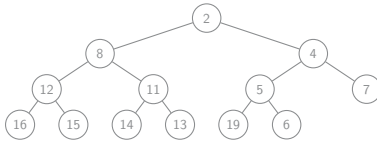
poll



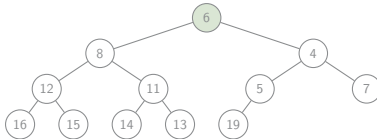
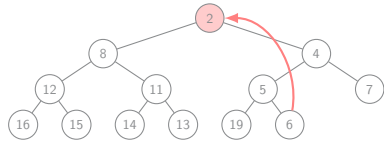
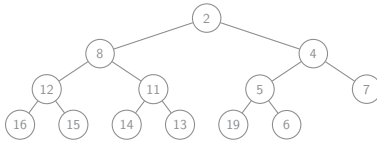
Binary Min-Heap



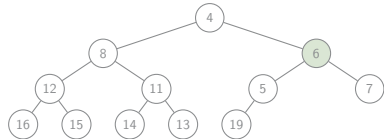
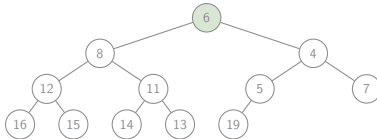
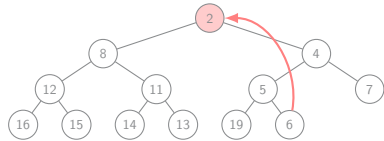
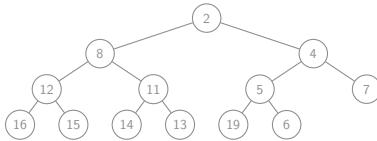
Binary Min-Heap



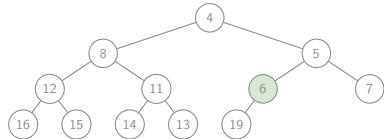
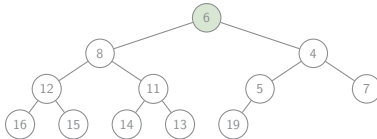
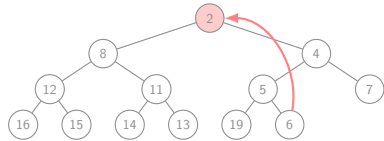
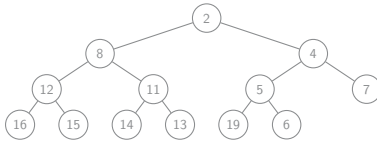
Binary Min-Heap



Binary Min-Heap



Binary Min-Heap



Binary Min-Heap

Operations

- ▶ `insert`
- ▶ `removeAt`
- ▶ `peek`
- ▶ `poll`



Binary Min-Heap

Operations

- ▶ insert $O(\log n)$
- ▶ removeAt $O(\log n)$
- ▶ peek $O(1)$
- ▶ poll $O(\log n)$



Binary Min-Heap

Operations

- ▶ insert $O(\log n)$
- ▶ removeAt $O(\log n)$
- ▶ peek $O(1)$
- ▶ poll $O(\log n)$

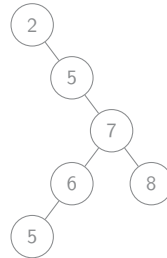
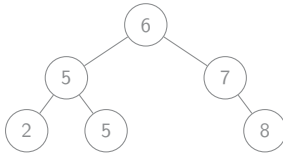
- ▶ Heapsort
- ▶ Frequently used in Priority Queues



Binary Search Tree

Definition

A binary tree in which the left child \leq the parent and the right child \geq the parent.



Binary Search Tree

Operations

- ▶ insert $O(\log n)$ / $O(n)$
- ▶ delete $O(\log n)$ / $O(n)$
- ▶ search $O(\log n)$ / $O(n)$



Red-Black Tree

- ▶ Binary search tree
- ▶ Approximately balanced
- ▶ NIL leaves
- ▶ Red-black properties



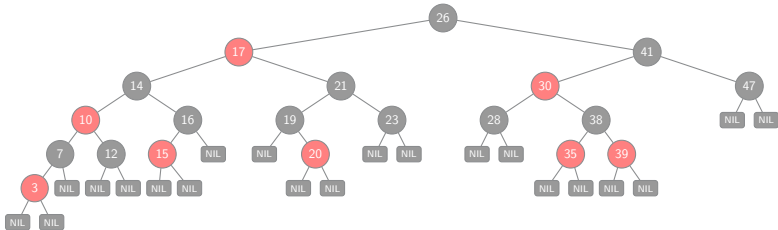
Red-Black Tree

- ▶ Binary search tree
- ▶ Approximately balanced
- ▶ NIL leaves
- ▶ Red-black properties
 - ▶ Every node is either red or black
 - ▶ Root is black
 - ▶ Every leaf is black
 - ▶ If a node is red, its children are black
 - ▶ For each node, all paths to its descendant leaves contain the same number of black nodes



Red-Black Tree

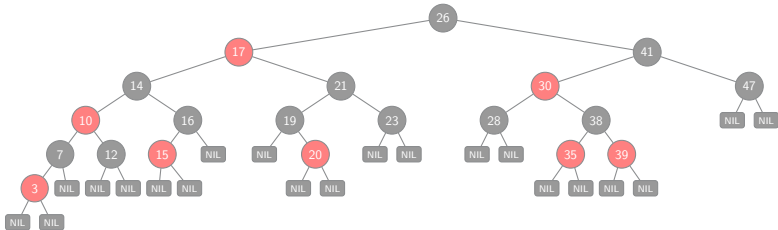
- ▶ Node is either red or black
- ▶ Root is black
- ▶ \forall node: all paths to its leaves have the same number of black nodes
- ▶ Every leaf is black
- ▶ If red, children are black



Red-Black Tree

Operations

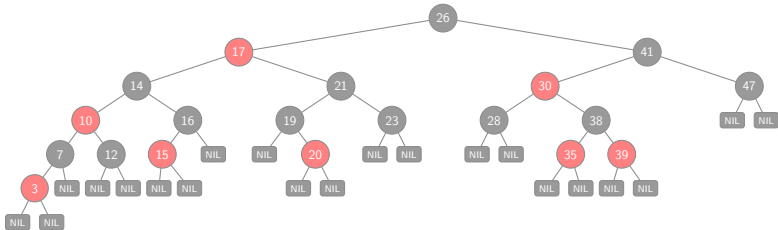
- ▶ insert
- ▶ delete
- ▶ search



Red-Black Tree

Operations

- ▶ insert $O(\log n)$
- ▶ delete $O(\log n)$
- ▶ search $O(\log n)$



Outline

1 Introductory Data Structures

- Array
- Linked List
- Hash Table
- Tree

2 Java Collection API & Map API

- Java Collection API
- Java Map API

3 Advanced Data Structures

- Skip list
- Bloom Filter
- van Emde Boas Tree



Outline

1 Introductory Data Structures

- Array
- Linked List
- Hash Table
- Tree

2 Java Collection API & Map API

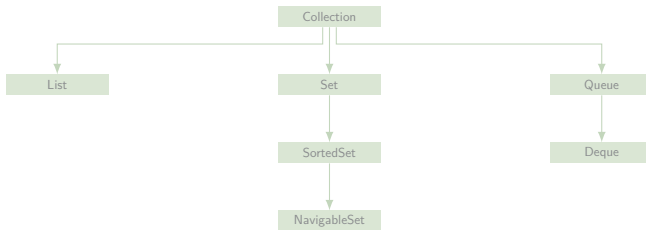
- Java Collection API
- Java Map API

3 Advanced Data Structures

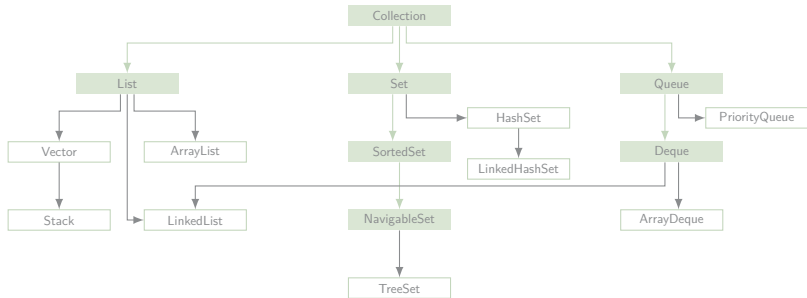
- Skip list
- Bloom Filter
- van Emde Boas Tree



Java Collection API



Java Collection API



List Interface

	Impl	add	remove	contains	get
LinkedList	linked list				
ArrayList	array				
Vector	array				
Stack	array				



List Interface

	Impl	add	remove	contains	get
LinkedList	linked list	$O(1)$			
ArrayList	array				
Vector	array				
Stack	array				



List Interface

	Impl	add	remove	contains	get
LinkedList	linked list	$O(1)$	$O(1)$		
ArrayList	array				
Vector	array				
Stack	array				



List Interface

	Impl	add	remove	contains	get
LinkedList	linked list	$O(1)$	$O(1)$	$O(n)$	
ArrayList	array				
Vector	array				
Stack	array				



List Interface

	Impl	add	remove	contains	get
LinkedList	linked list	$O(1)$	$O(1)$	$O(n)$	$O(n)$
ArrayList	array				
Vector	array				
Stack	array				



List Interface

	Impl	add	remove	contains	get
LinkedList	linked list	$O(1)$	$O(1)$	$O(n)$	$O(n)$
ArrayList	array	$O(1)$			
Vector	array				
Stack	array				



List Interface

	Impl	add	remove	contains	get
LinkedList	linked list	$O(1)$	$O(1)$	$O(n)$	$O(n)$
ArrayList	array	$O(1)$	$O(n)$		
Vector	array				
Stack	array				



List Interface

	Impl	add	remove	contains	get
LinkedList	linked list	$O(1)$	$O(1)$	$O(n)$	$O(n)$
ArrayList	array	$O(1)$	$O(n)$	$O(n)$	
Vector	array				
Stack	array				



List Interface

	Impl	add	remove	contains	get
LinkedList	linked list	$O(1)$	$O(1)$	$O(n)$	$O(n)$
ArrayList	array	$O(1)$	$O(n)$	$O(n)$	$O(1)$
Vector	array				
Stack	array				



List Interface

	Impl	add	remove	contains	get
LinkedList	linked list	$O(1)$	$O(1)$	$O(n)$	$O(n)$
ArrayList	array	$O(1)$	$O(n)$	$O(n)$	$O(1)$
Vector	array	$O(1)$	$O(n)$	$O(n)$	$O(1)$
Stack	array	$O(1)$	$O(n)$	$O(n)$	$O(1)$



Set Interface

	Impl	add	contains	next
HashSet	hash table			
LinkedHashSet	hash table linked list			
TreeSet	red-black tree			



Set Interface

	Impl	add	contains	next
HashSet	hash table	$O(1)$		
LinkedHashSet	hash table linked list			
TreeSet	red-black tree			



Set Interface

	Impl	add	contains	next
HashSet	hash table	$O(1)$	$O(1)$	
LinkedHashSet	hash table linked list			
TreeSet	red-black tree			



Set Interface

	Impl	add	contains	next
HashSet	hash table	$O(1)$	$O(1)$	$O(h/n)$
LinkedHashSet	hash table linked list			
TreeSet	red-black tree			



Set Interface

	Impl	add	contains	next
HashSet	hash table	$O(1)$	$O(1)$	$O(h/n)$
LinkedHashSet	hash table linked list	$O(1)$		
TreeSet	red-black tree			



Set Interface

	Impl	add	contains	next
HashSet	hash table	$O(1)$	$O(1)$	$O(h/n)$
LinkedHashSet	hash table linked list	$O(1)$	$O(1)$	
TreeSet	red-black tree			



Set Interface

	Impl	add	contains	next
HashSet	hash table	$O(1)$	$O(1)$	$O(h/n)$
LinkedHashSet	hash table linked list	$O(1)$	$O(1)$	$O(1)$
TreeSet	red-black tree			



Set Interface

	Impl	add	contains	next
HashSet	hash table	$O(1)$	$O(1)$	$O(h/n)$
LinkedHashSet	hash table linked list	$O(1)$	$O(1)$	$O(1)$
TreeSet	red-black tree	$O(\log n)$		



Set Interface

	Impl	add	contains	next
HashSet	hash table	$O(1)$	$O(1)$	$O(h/n)$
LinkedHashSet	hash table linked list	$O(1)$	$O(1)$	$O(1)$
TreeSet	red-black tree	$O(\log n)$	$O(\log n)$	



Set Interface

	Impl	add	contains	next
HashSet	hash table	$O(1)$	$O(1)$	$O(h/n)$
LinkedHashSet	hash table linked list	$O(1)$	$O(1)$	$O(1)$
TreeSet	red-black tree	$O(\log n)$	$O(\log n)$	$O(\log n)$



Queue Interface

	Impl	offer	peek	poll
PriorityQueue	binary heap			
ArrayDeque	array			
LinkedList	linked list			



Queue Interface

	Impl	offer	peek	poll
PriorityQueue	binary heap	$O(\log n)$		
ArrayDeque	array			
LinkedList	linked list			



Queue Interface

	Impl	offer	peak	poll
PriorityQueue	binary heap	$O(\log n)$	$O(1)$	
ArrayDeque	array			
LinkedList	linked list			



Queue Interface

	Impl	offer	peak	poll
PriorityQueue	binary heap	$O(\log n)$	$O(1)$	$O(\log n)$
ArrayDeque	array			
LinkedList	linked list			



Queue Interface

	Impl	offer	peak	poll
PriorityQueue	binary heap	$O(\log n)$	$O(1)$	$O(\log n)$
ArrayDeque	array	$O(1)$		
LinkedList	linked list			



Queue Interface

	Impl	offer	peak	poll
PriorityQueue	binary heap	$O(\log n)$	$O(1)$	$O(\log n)$
ArrayDeque	array	$O(1)$	$O(1)$	
LinkedList	linked list			



Queue Interface

	Impl	offer	peak	poll
PriorityQueue	binary heap	$O(\log n)$	$O(1)$	$O(\log n)$
ArrayDeque	array	$O(1)$	$O(1)$	$O(1)$
LinkedList	linked list			



Queue Interface

	Impl	offer	peak	poll
PriorityQueue	binary heap	$O(\log n)$	$O(1)$	$O(\log n)$
ArrayDeque	array	$O(1)$	$O(1)$	$O(1)$
LinkedList	linked list	$O(1)$	$O(1)$	$O(1)$



Outline

1 Introductory Data Structures

- Array
- Linked List
- Hash Table
- Tree

2 Java Collection API & Map API

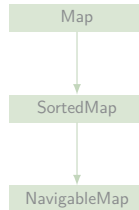
- Java Collection API
- Java Map API

3 Advanced Data Structures

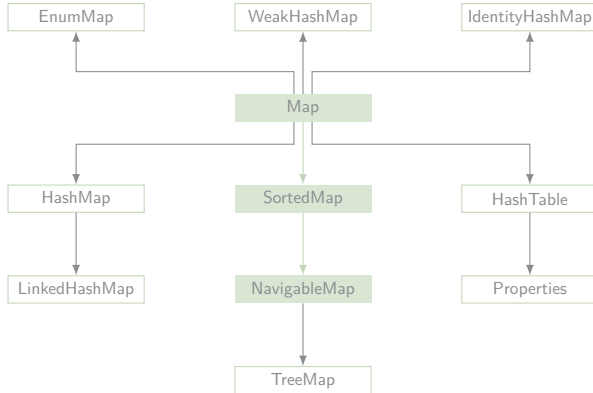
- Skip list
- Bloom Filter
- van Emde Boas Tree



Java Map API



Java Map API



Map Interface

	Impl	get	containsKey	next
HashTable	hash table			
Properties	hash table			
HashMap	hash table			
LinkedHashMap	hash table linked list			
TreeMap	red-black tree			
IdentityHashMap	array			
WeakHashMap	hash table			
EnumMap	array			



Map Interface

	Impl	get	containsKey	next
HashTable	hash table	$O(1)$		
Properties	hash table	$O(1)$		
HashMap	hash table	$O(1)$		
LinkedHashMap	hash table linked list			
TreeMap	red-black tree			
IdentityHashMap	array			
WeakHashMap	hash table			
EnumMap	array			



Map Interface

	Impl	get	containsKey	next
HashTable	hash table	$O(1)$	$O(1)$	
Properties	hash table	$O(1)$	$O(1)$	
HashMap	hash table	$O(1)$	$O(1)$	
LinkedHashMap	hash table linked list			
TreeMap	red-black tree			
IdentityHashMap	array			
WeakHashMap	hash table			
EnumMap	array			



Map Interface

	Impl	get	containsKey	next
HashTable	hash table	$O(1)$	$O(1)$	$O(h/n)$
Properties	hash table	$O(1)$	$O(1)$	$O(h/n)$
HashMap	hash table	$O(1)$	$O(1)$	$O(h/n)$
LinkedHashMap	hash table linked list			
TreeMap	red-black tree			
IdentityHashMap	array			
WeakHashMap	hash table			
EnumMap	array			



Map Interface

	Impl	get	containsKey	next
HashTable	hash table	$O(1)$	$O(1)$	$O(h/n)$
Properties	hash table	$O(1)$	$O(1)$	$O(h/n)$
HashMap	hash table	$O(1)$	$O(1)$	$O(h/n)$
LinkedHashMap	hash table linked list	$O(1)$		
TreeMap	red-black tree			
IdentityHashMap	array			
WeakHashMap	hash table			
EnumMap	array			



Map Interface

	Impl	get	containsKey	next
HashTable	hash table	$O(1)$	$O(1)$	$O(h/n)$
Properties	hash table	$O(1)$	$O(1)$	$O(h/n)$
HashMap	hash table	$O(1)$	$O(1)$	$O(h/n)$
LinkedHashMap	hash table linked list	$O(1)$	$O(1)$	
TreeMap	red-black tree			
IdentityHashMap	array			
WeakHashMap	hash table			
EnumMap	array			



Map Interface

	Impl	get	containsKey	next
HashTable	hash table	$O(1)$	$O(1)$	$O(h/n)$
Properties	hash table	$O(1)$	$O(1)$	$O(h/n)$
HashMap	hash table	$O(1)$	$O(1)$	$O(h/n)$
LinkedHashMap	hash table linked list	$O(1)$	$O(1)$	$O(1)$
TreeMap	red-black tree			
IdentityHashMap	array			
WeakHashMap	hash table			
EnumMap	array			



Map Interface

	Impl	get	containsKey	next
HashTable	hash table	$O(1)$	$O(1)$	$O(h/n)$
Properties	hash table	$O(1)$	$O(1)$	$O(h/n)$
HashMap	hash table	$O(1)$	$O(1)$	$O(h/n)$
LinkedHashMap	hash table linked list	$O(1)$	$O(1)$	$O(1)$
TreeMap	red-black tree	$O(\log n)$		
IdentityHashMap	array			
WeakHashMap	hash table			
EnumMap	array			



Map Interface

	Impl	get	containsKey	next
HashTable	hash table	$O(1)$	$O(1)$	$O(h/n)$
Properties	hash table	$O(1)$	$O(1)$	$O(h/n)$
HashMap	hash table	$O(1)$	$O(1)$	$O(h/n)$
LinkedHashMap	hash table linked list	$O(1)$	$O(1)$	$O(1)$
TreeMap	red-black tree	$O(\log n)$	$O(\log n)$	
IdentityHashMap	array			
WeakHashMap	hash table			
EnumMap	array			



Map Interface

	Impl	get	containsKey	next
HashTable	hash table	$O(1)$	$O(1)$	$O(h/n)$
Properties	hash table	$O(1)$	$O(1)$	$O(h/n)$
HashMap	hash table	$O(1)$	$O(1)$	$O(h/n)$
LinkedHashMap	hash table linked list	$O(1)$	$O(1)$	$O(1)$
TreeMap	red-black tree	$O(\log n)$	$O(\log n)$	$O(\log n)$
IdentityHashMap	array			
WeakHashMap	hash table			
EnumMap	array			



Map Interface

	Impl	get	containsKey	next
HashTable	hash table	$O(1)$	$O(1)$	$O(h/n)$
Properties	hash table	$O(1)$	$O(1)$	$O(h/n)$
HashMap	hash table	$O(1)$	$O(1)$	$O(h/n)$
LinkedHashMap	hash table linked list	$O(1)$	$O(1)$	$O(1)$
TreeMap	red-black tree	$O(\log n)$	$O(\log n)$	$O(\log n)$
IdentityHashMap	array	$O(1)$	$O(1)$	$O(h/n)$
WeakHashMap	hash table	$O(1)$	$O(1)$	$O(h/n)$
EnumMap	array	$O(1)$	$O(1)$	$O(1)$



Outline

1 Introductory Data Structures

- Array
- Linked List
- Hash Table
- Tree

2 Java Collection API & Map API

- Java Collection API
- Java Map API

3 Advanced Data Structures

- Skip list
- Bloom Filter
- van Emde Boas Tree



Outline

1 Introductory Data Structures

- Array
- Linked List
- Hash Table
- Tree

2 Java Collection API & Map API

- Java Collection API
- Java Map API

3 Advanced Data Structures

- Skip list
- Bloom Filter
- van Emde Boas Tree

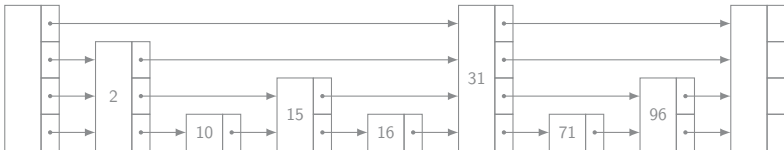


Skip list

- ▶ Balanced binary tree alternative
 - ▶ insert, delete & search in $O(\log n)$
- ▶ Probabilistic balancing rather than strictly enforced balancing
- ▶ Insertion and deletion \rightarrow simpler and faster
- ▶ Linked hierarchy of subsequences, with each successive subsequence skipping over fewer elements than the previous one
 - ▶ Hierarchy has $\approx \log n$ levels

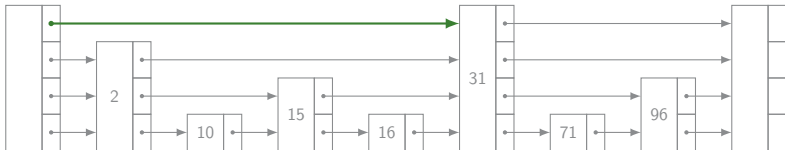


Skip list



[illegible]

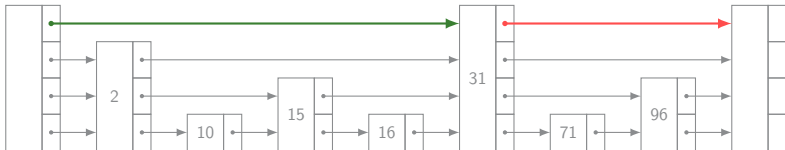
Skip list



find 71



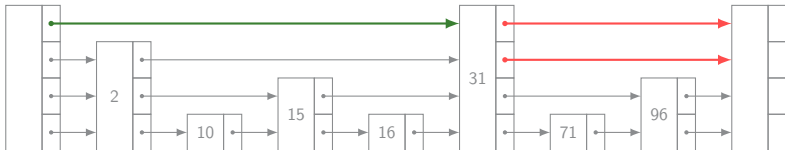
Skip list



find 71



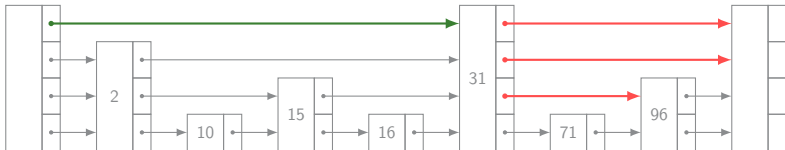
Skip list



find 71



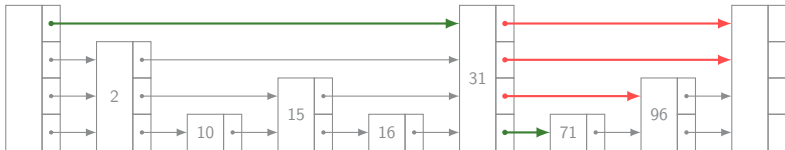
Skip list



find 71



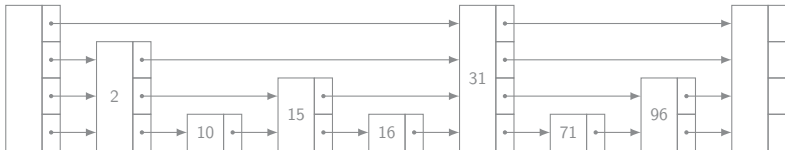
Skip list



find 71



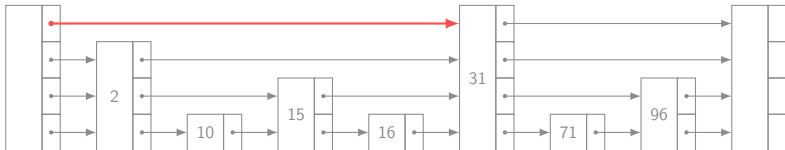
Skip list



find 12



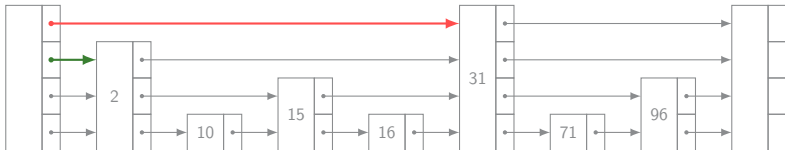
Skip list



find 12



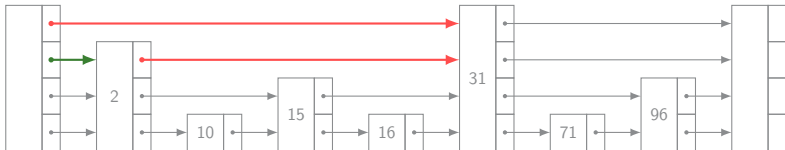
Skip list



find 12



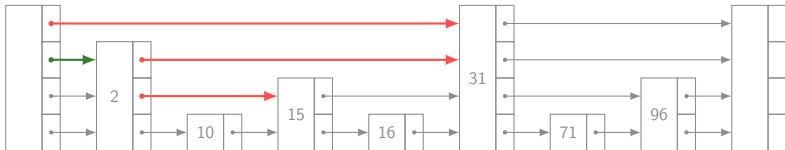
Skip list



find 12



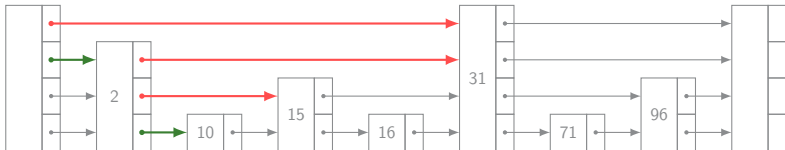
Skip list



find 12



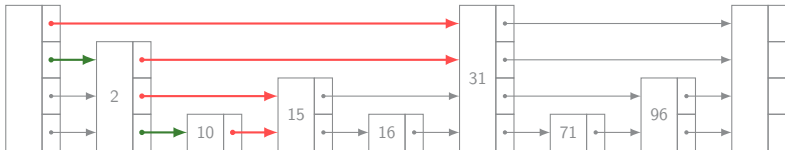
Skip list



find 12



Skip list



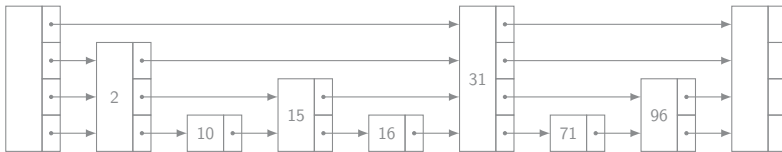
find 12



Skip list

Insertion

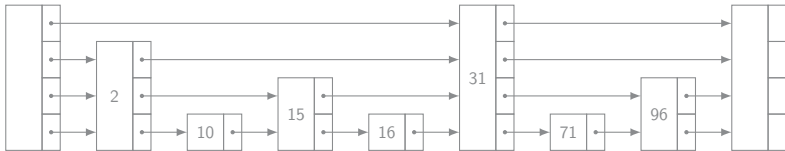
- ▶ insert at level=1
- ▶ while (coinflip() == HEADS)
 insert at ++level



Skip list

Insertion

- ▶ `insert at level=1`
- ▶ `while (coinflip() == HEADS)`
 `insert at ++level`

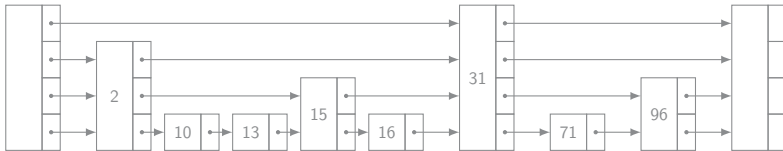


`insert 13`

Skip list

Insertion

- ▶ `insert at level=1`
- ▶ `while (coinflip() == HEADS)`
 `insert at ++level`

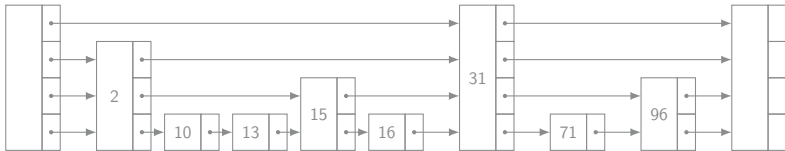


`insert 13`

Skip list

Insertion

- ▶ insert at level=1
- ▶ while (coinflip() == HEADS)
 insert at ++level

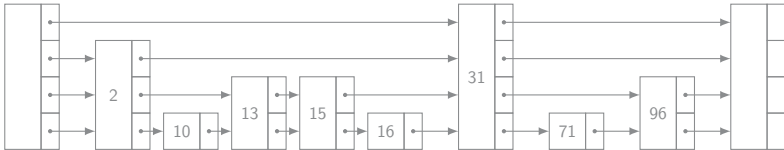


insert 13 — Coinflips: H

Skip list

Insertion

- ▶ insert at level=1
- ▶ while (coinflip() == HEADS)
 insert at ++level

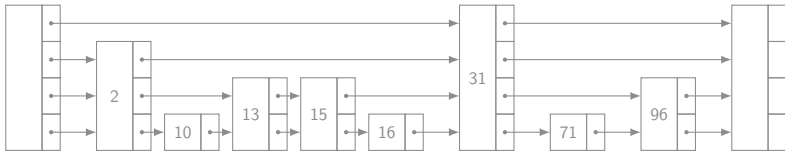


insert 13 — Coinflips: H

Skip list

Insertion

- ▶ insert at level=1
- ▶ while (coinflip() == HEADS)
 insert at ++level



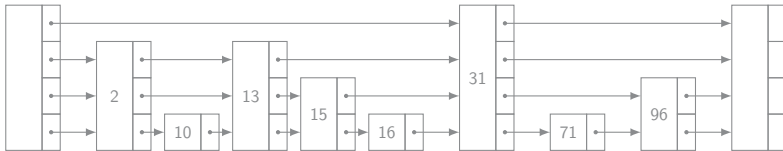
insert 13 — Coinflips: HH



Skip list

Insertion

- ▶ insert at level=1
- ▶ while (coinflip() == HEADS)
 insert at ++level



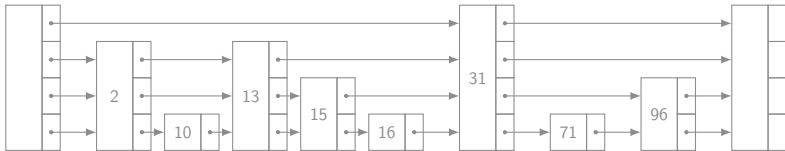
insert 13 — Coinflips: HH



Skip list

Insertion

- ▶ insert at level=1
- ▶ while (coinflip() == HEADS)
 insert at ++level



insert 13 — Coinflips: HHT



Skip list

Operations

- ▶ insert
- ▶ delete
- ▶ search



Skip list

Operations

- ▶ insert $O(\log n)$ / $O(n)$
- ▶ delete $O(\log n)$ / $O(n)$
- ▶ search $O(\log n)$ / $O(n)$



Outline

1 Introductory Data Structures

- Array
- Linked List
- Hash Table
- Tree

2 Java Collection API & Map API

- Java Collection API
- Java Map API

3 Advanced Data Structures

- Skip list
- Bloom Filter**
- van Emde Boas Tree



Bloom Filter

- ▶ Data structure to indicate if an element is contained in a set
- ▶ Bit array → space efficient
- ▶ Probabilistic
 - ▶ False positives possible
 - ▶ False negatives impossible
- ▶ Elements can be added to the set, but not removed



Bloom Filter

0
1
2
3
4
5
6
7
8
9
10
11
12
13



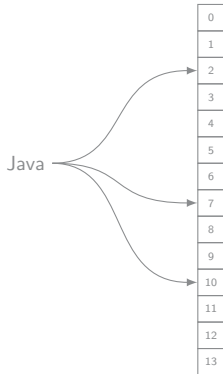
Bloom Filter

Java

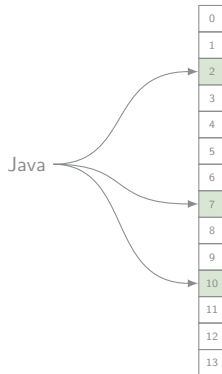
0
1
2
3
4
5
6
7
8
9
10
11
12
13



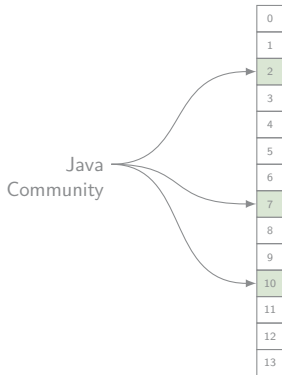
Bloom Filter



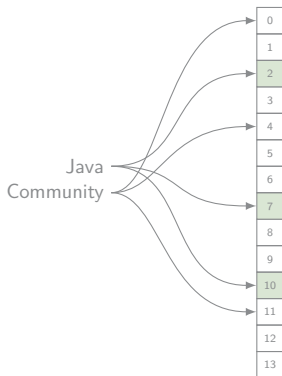
Bloom Filter



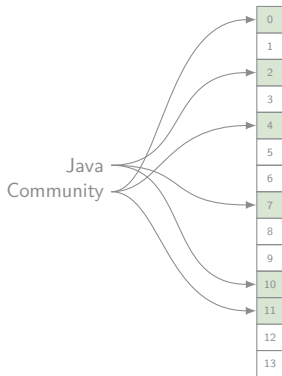
Bloom Filter



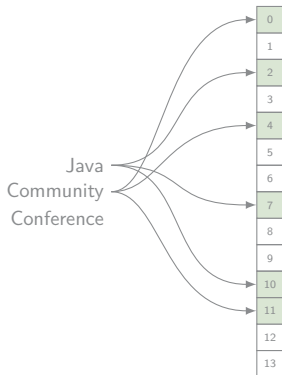
Bloom Filter



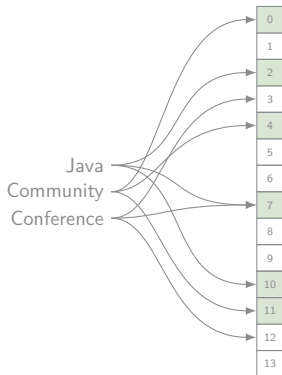
Bloom Filter



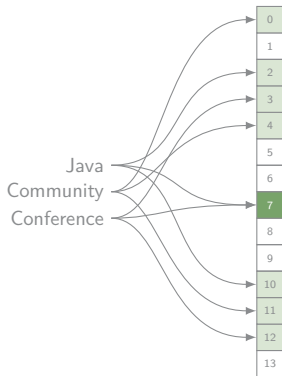
Bloom Filter



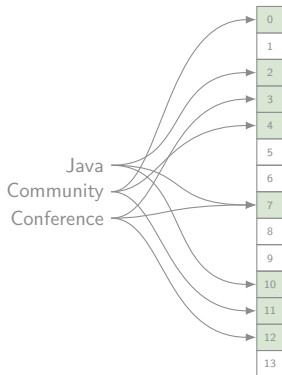
Bloom Filter



Bloom Filter



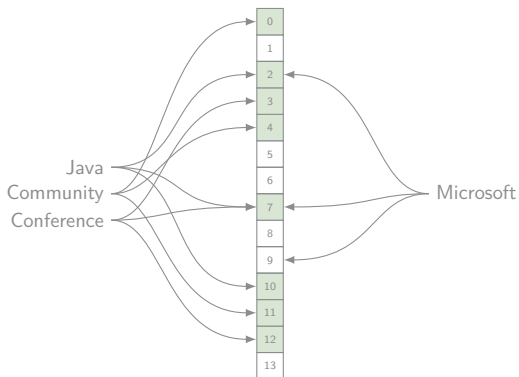
Bloom Filter



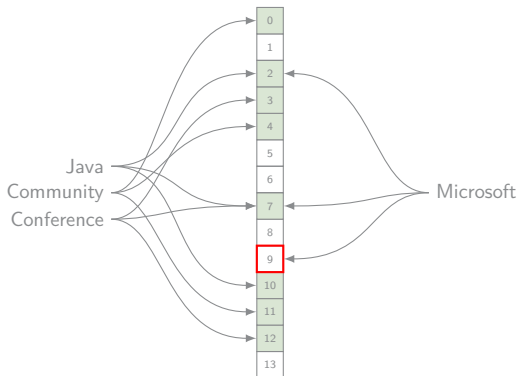
Microsoft



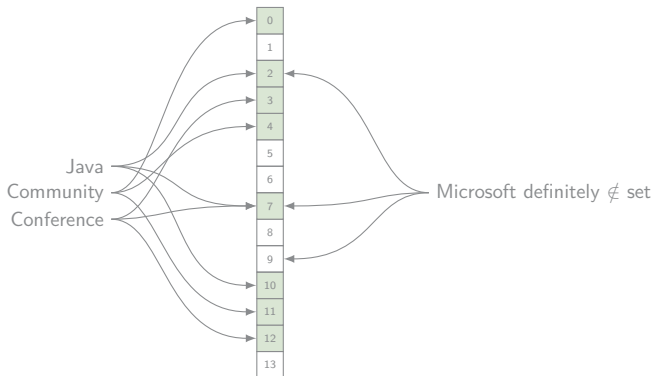
Bloom Filter



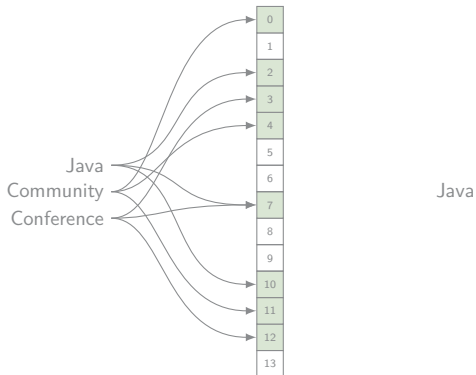
Bloom Filter



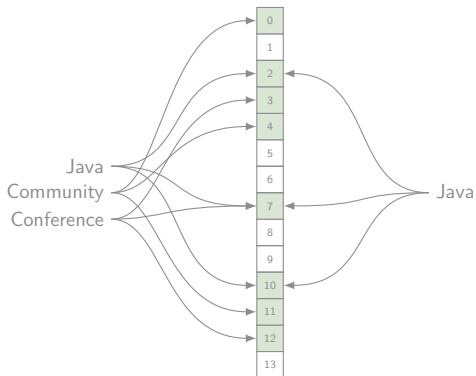
Bloom Filter



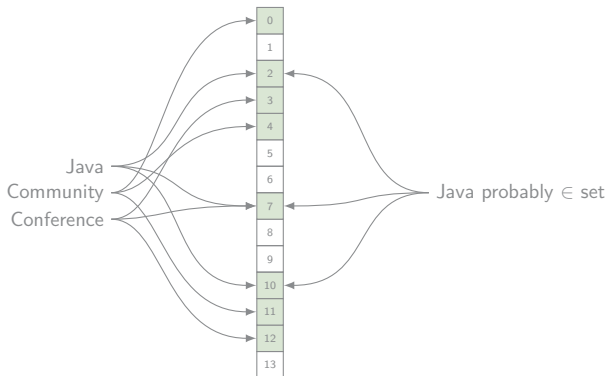
Bloom Filter



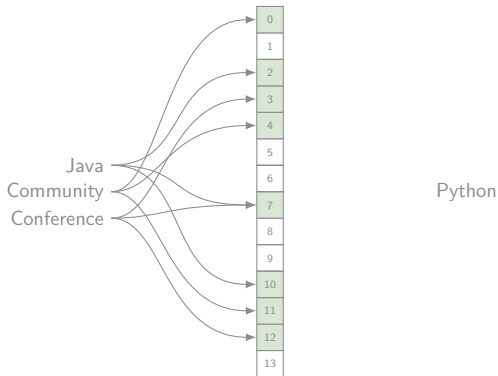
Bloom Filter



Bloom Filter



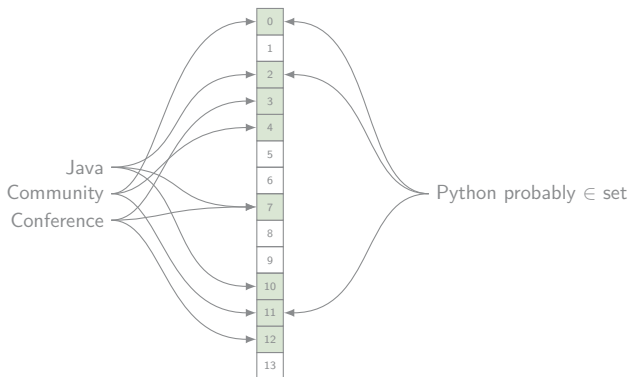
Bloom Filter



Bloom Filter



Bloom Filter



Bloom filter

Math

- ▶ m : array length
- ▶ k : # hash functions
- ▶ n : # of inserted elements
- ▶ p : false positive probability
- ▶ # bits needed: $m = \frac{-n \ln p}{(\ln 2)^2}$
- ▶ Optimal # of hash functions: $k = \frac{m}{n} \ln(2)$

Bloom filter

Math

- ▶ m : array length
- ▶ k : # hash functions
- ▶ n : # of inserted elements
- ▶ p : false positive probability
- ▶ # bits needed: $m = \frac{-n \ln p}{(\ln 2)^2}$
- ▶ Optimal # of hash functions: $k = \frac{m}{n} \ln(2)$
- ▶ Supports unions and intersections
- ▶ Fits arbitrarily # of elements (false positive rate increases)
- ▶ Common application: caching

Outline

1 Introductory Data Structures

- Array
- Linked List
- Hash Table
- Tree

2 Java Collection API & Map API

- Java Collection API
- Java Map API

3 Advanced Data Structures

- Skip list
- Bloom Filter
- van Emde Boas Tree

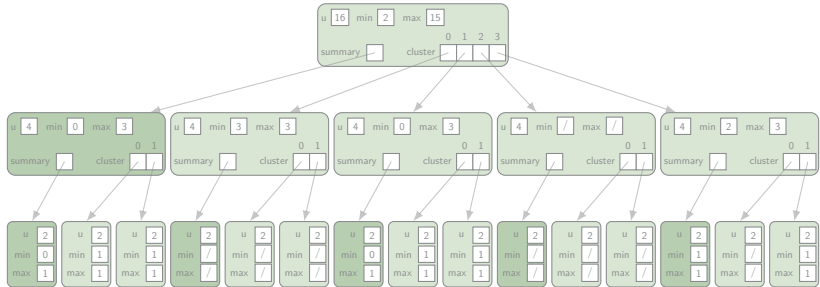


van Emde Boas Tree

- ▶ Dynamic ordered set/map
 - ▶ insert, delete, contains, minimum, maximum, successor, predecessor operations in $O(\log \log U)$
- ▶ Integer keys
 - ▶ No duplicates
- ▶ $O(U)$ space \rightarrow only for large collections

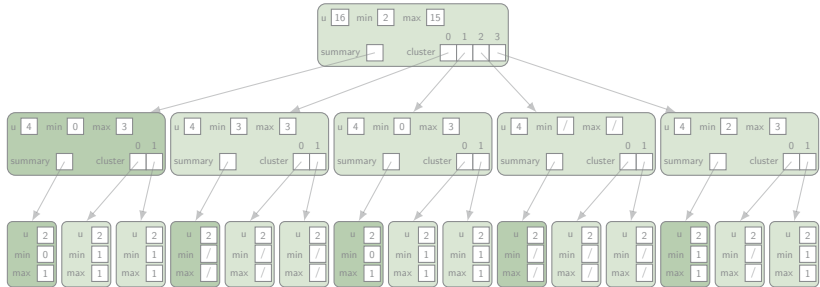


van Emde Boas Tree



{2,3,4,5,7,14,15}

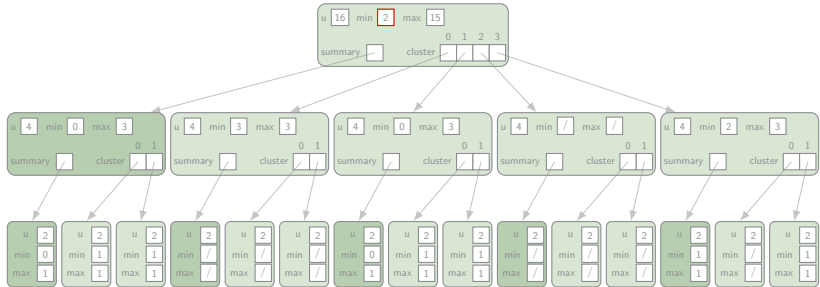
van Emde Boas Tree



`{2,3,4,5,7,14,15}`
minimum



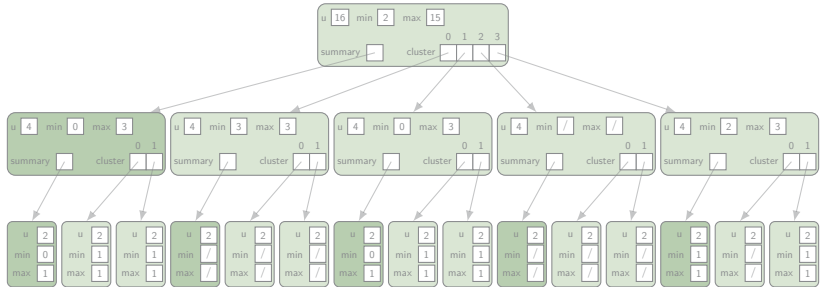
van Emde Boas Tree



{2,3,4,5,7,14,15}
minimum



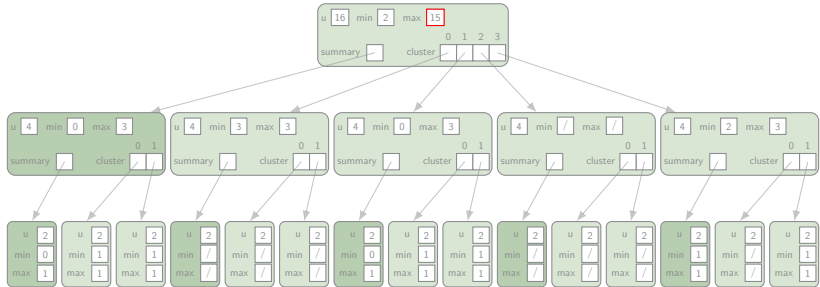
van Emde Boas Tree



{2,3,4,5,7,14,15}
maximum



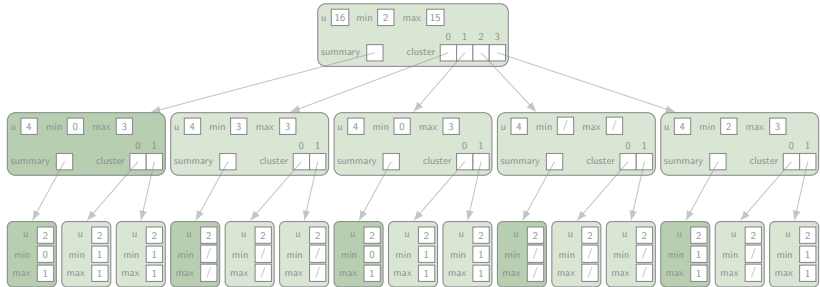
van Emde Boas Tree



$\{2, 3, 4, 5, 7, 14, 15\}$
maximum

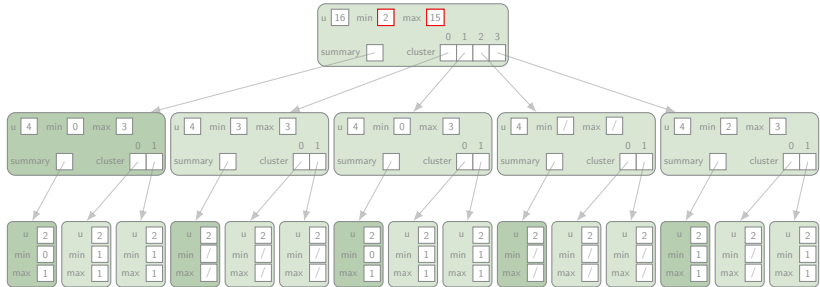


van Emde Boas Tree



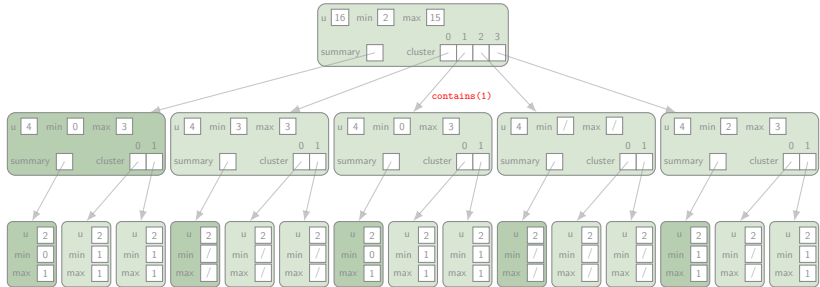
`{2,3,4,5,7,14,15}`
`contains(5)`

van Emde Boas Tree



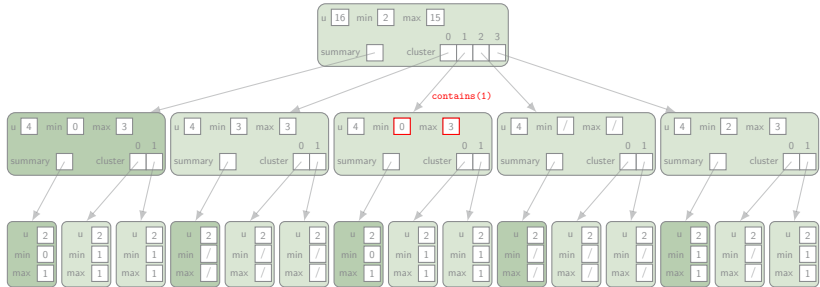
`{2,3,4,5,7,14,15}`
`contains(5)`

van Emde Boas Tree



$\{2,3,4,5,7,14,15\}$
contains(5)

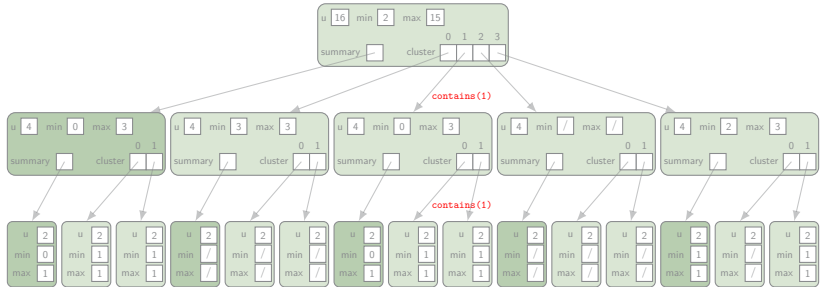
van Emde Boas Tree



`{2,3,4,5,7,14,15}`
`contains(5)`

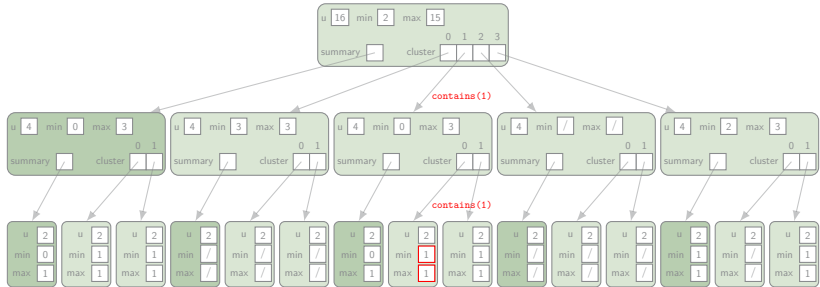


van Emde Boas Tree



`{2,3,4,5,7,14,15}`
`contains(5)`

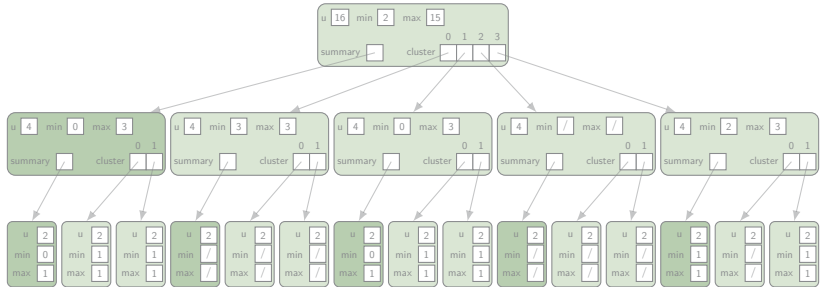
van Emde Boas Tree



`{2,3,4,5,7,14,15}`
`contains(5)`

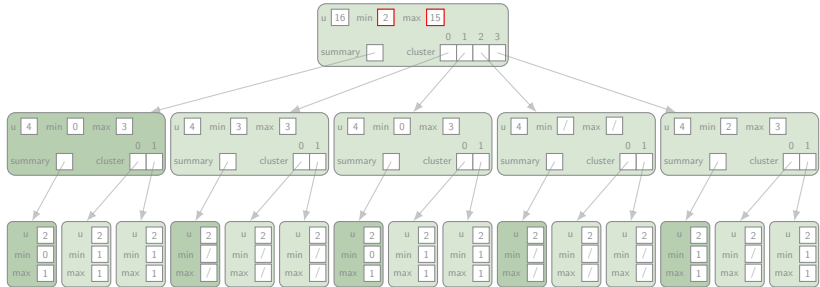


van Emde Boas Tree



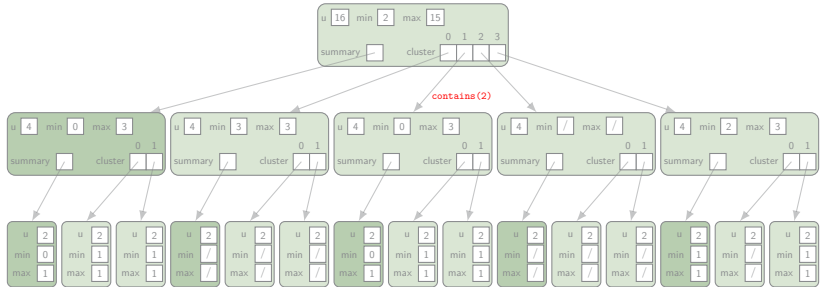
`{2,3,4,5,7,14,15}`
`contains(6)`

van Emde Boas Tree



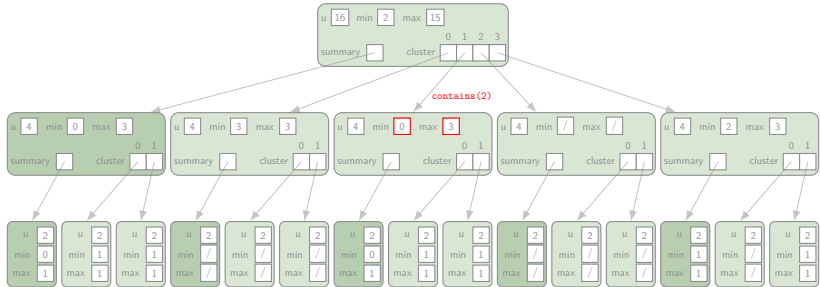
$\{2,3,4,5,7,14,15\}$
contains(6)

van Emde Boas Tree



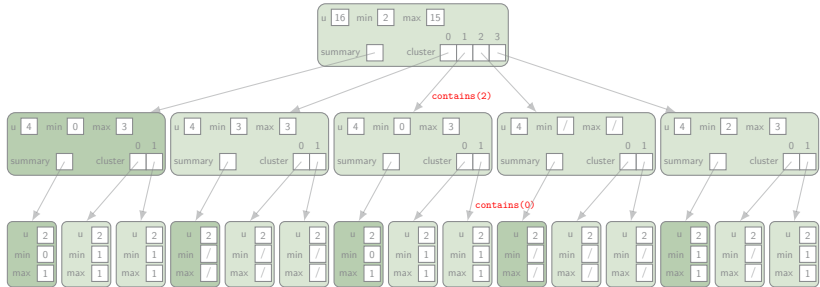
{2,3,4,5,7,14,15}
contains(6)

van Emde Boas Tree



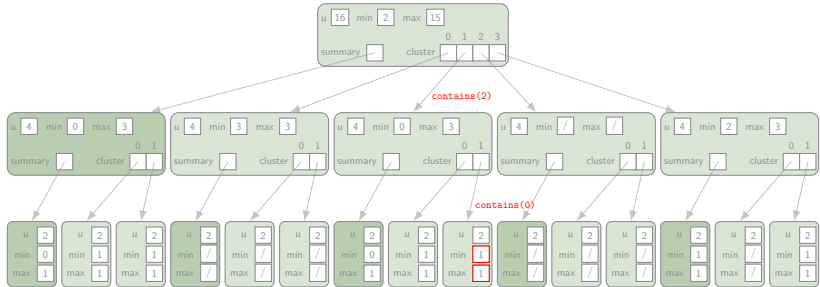
`{2,3,4,5,7,14,15}`
`contains(6)`

van Emde Boas Tree



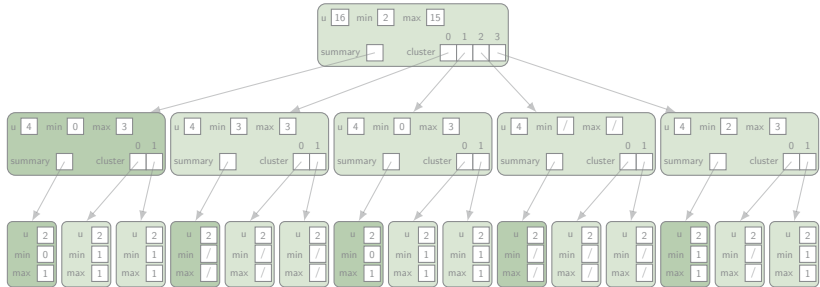
{2,3,4,5,7,14,15}
contains(6)

van Emde Boas Tree



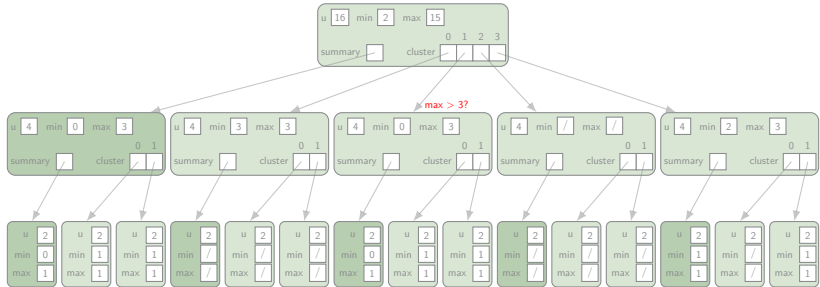
{2,3,4,5,7,14,15}
`contains(6)`

van Emde Boas Tree



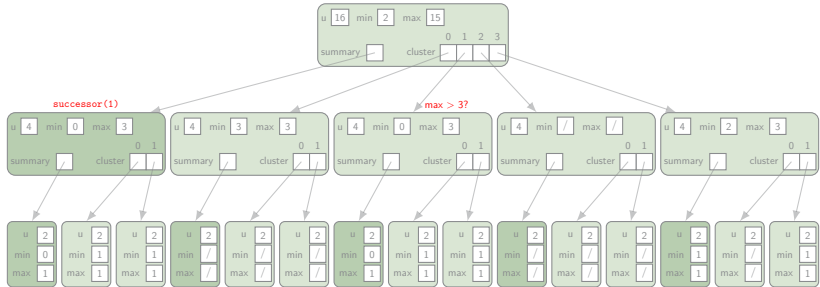
{2,3,4,5,7,14,15}
successor(7)

van Emde Boas Tree



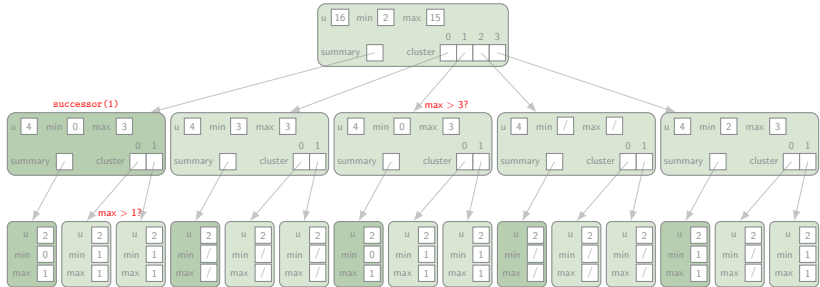
{2,3,4,5,7,14,15}
successor(7)

van Emde Boas Tree



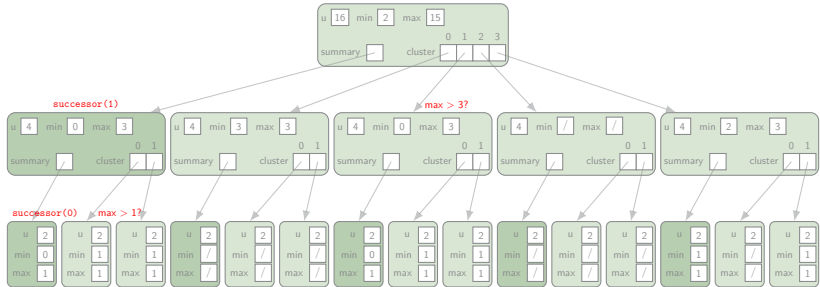
{2,3,4,5,7,14,15}
successor(7)

van Emde Boas Tree



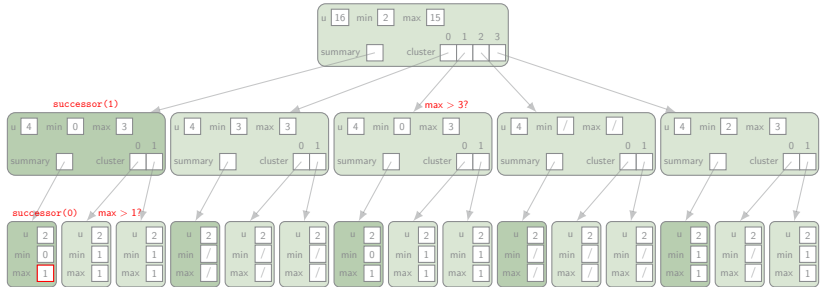
{2,3,4,5,7,14,15}
successor(7)

van Emde Boas Tree



{2,3,4,5,7,14,15}
successor(7)

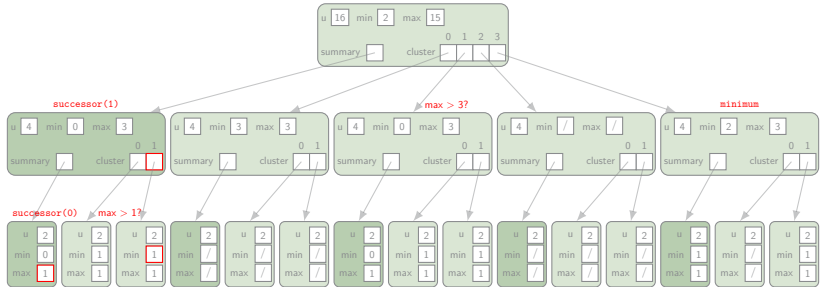
van Emde Boas Tree



{2,3,4,5,7,14,15}
successor(7)

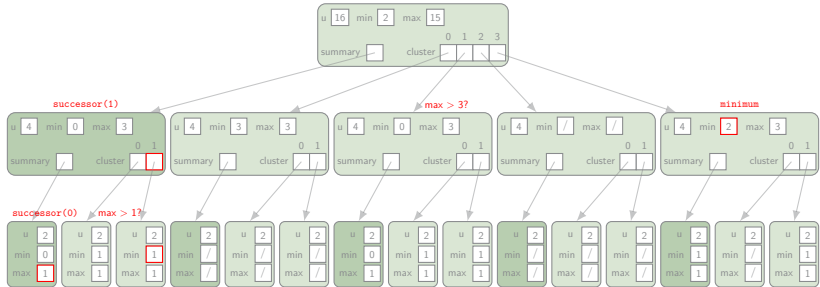
```
{2,3,4,5,7,14,15}
  successor(7)
```

van Emde Boas Tree



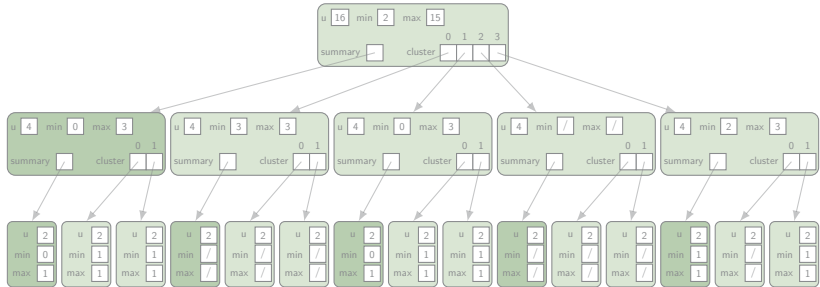
{2,3,4,5,7,14,15}
successor(7)

van Emde Boas Tree



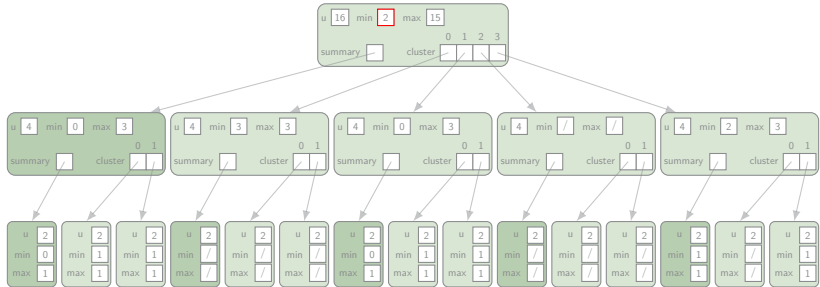
{2,3,4,5,7,14,15}
successor(7)

van Emde Boas Tree



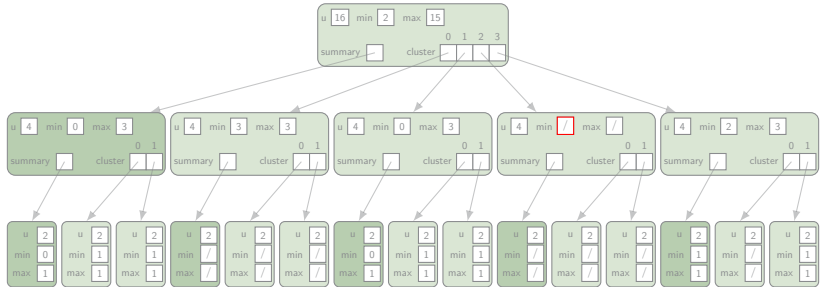
{2,3,4,5,7,14,15}
insert(11)

van Emde Boas Tree



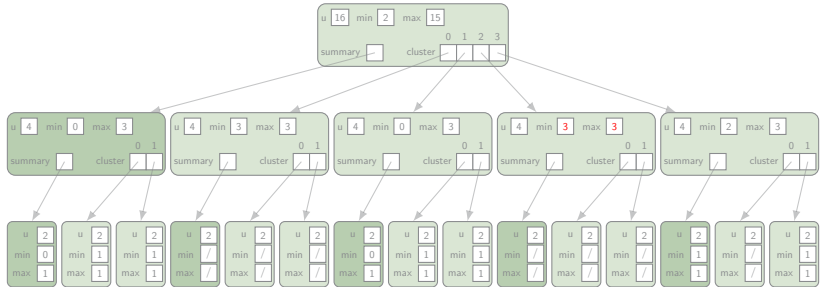
{2,3,4,5,7,14,15}
insert(11)

van Emde Boas Tree



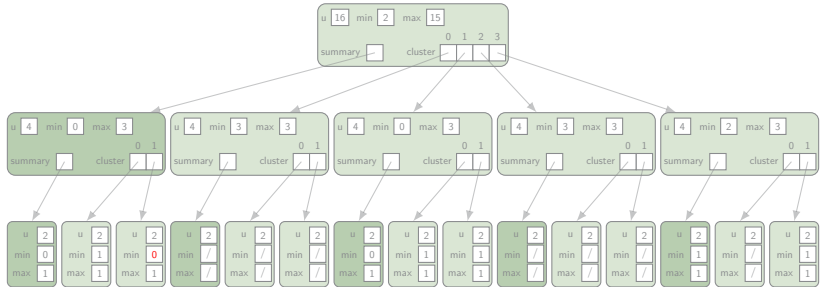
{2,3,4,5,7,14,15}
insert(11)

van Emde Boas Tree



{2,3,4,5,7,14,15}
insert(11)

van Emde Boas Tree



{2,3,4,5,7,14,15}
insert(11)



van Emde Boas Tree

Operations

- ▶ min, max, isEmpty $O(1)$
- ▶ insert, delete, contains, succ, pred $O(\log \log U)$
- ▶ $O(U)$ space
 - ▶ X-fast trie: $O(n \log U)$ space
 - ▶ Y-fast trie: $O(n)$ space
- ▶ $U = 2^k, k \in \mathbb{N}$



Thank you for your attention!

Slides:

<https://github.com/MDhondt/Data-Structures-presentation>

