Data Structures II

Hakuna Matata Group

- ❑ Linked List
- ❑ Linked List Operations
- ❑ Types of Linked List
- ❑ Hash Table
- ❑ Heap Data Structure
- ❑ Fibonacci Heap
- ❑ Decrease Key and Delete node from Fibonacci Heap

INTR






Renz

Hakuna Matata Group

Topic: Link List

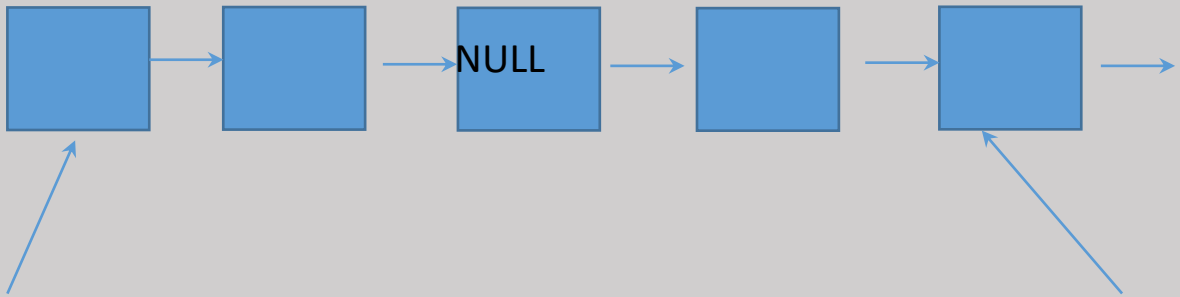
REN

Linked List

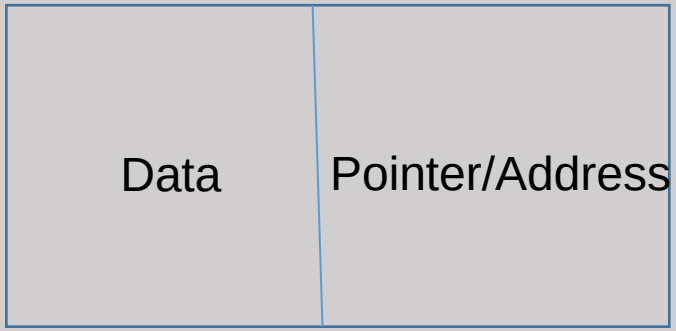
- A linked list is a linear data structure that includes a series of connected nodes. Here, each node stores the data and the address of the next node.

Collection of nodes/objects

FOR EXAMPLE :

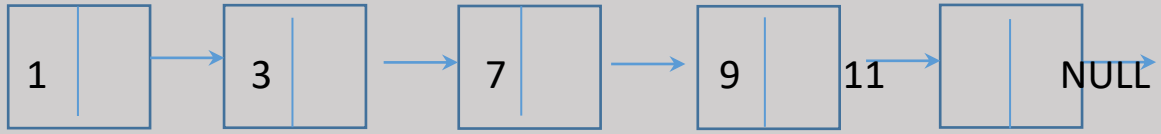


2 parts of nodes

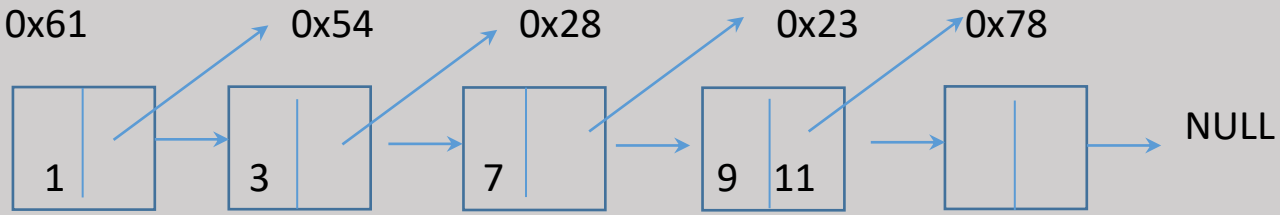


Linked List

DATA A = {1, 3 ,7 ,9, 11};



ADDRESS0



REN

Linked List

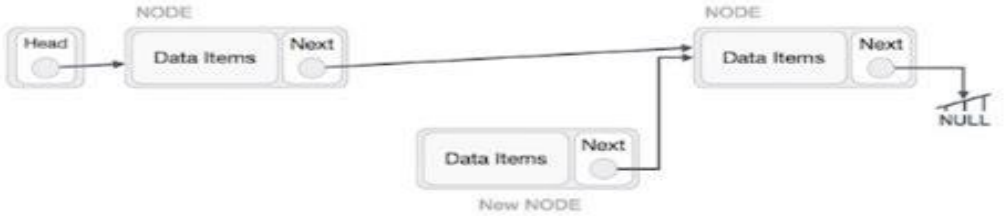
Insertion Operation

Adding a new node in linked list is a more than one step activity. We shall learn this with diagrams here. First, create a node using the same structure and find the location where it has to be inserted.

Imagine that we are inserting a node **B** (NewNode), between **A** (LeftNode) and **C** (RightNode). Then point B.next to C −

```
NewNode.next -> RightNode;
```

It should look like this −

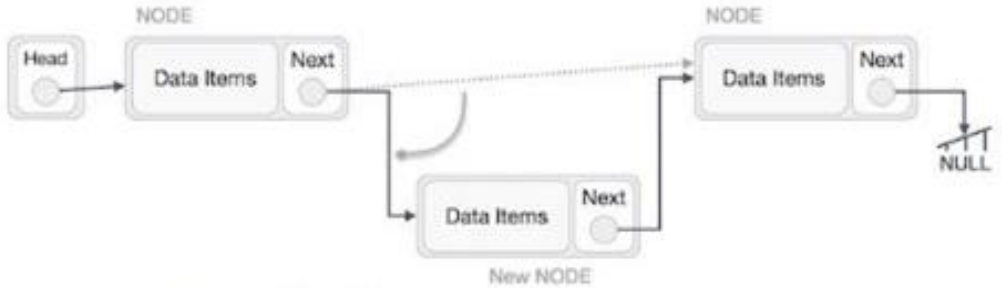


Now, the next node at the left should point to the new node.

REN

Linked List

```
LeftNode.next -> NewNode;
```



This will put the new node in the middle of the two. The new list should look like this –

Similar steps should be taken if the node is being inserted at the beginning of the list. While inserting it at the end, the second last node of the list should point to the new node and the new node will point to NULL.

REN

Deletion Operation

Deletion is also a more than one step process. We shall learn with pictorial representation. First, locate the target node to be removed, by using searching algorithms.

The left (previous) node of the target node now should point to the next node of the target node −

```
LeftNode.next -> TargetNode.next;
```

This will remove the link that was pointing to the target node. Now, using the following code, we will remove what the target node is pointing at.

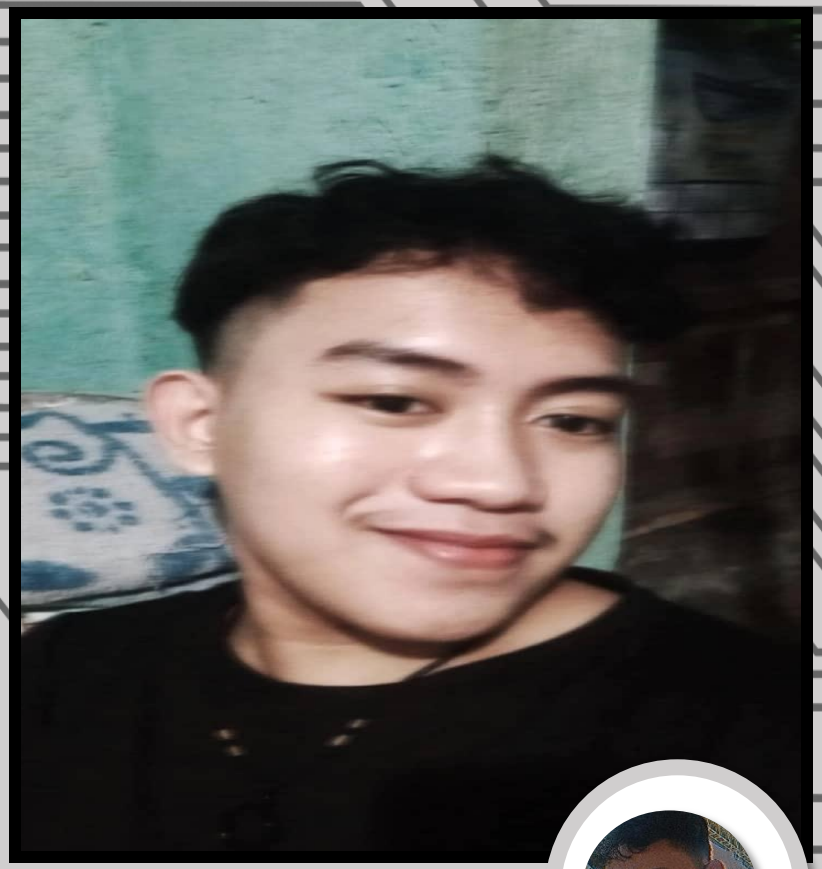
```
TargetNode.next -> NULL;
```

We need to use the deleted node. We can keep that in memory otherwise we can simply deallocate memory and wipe off the target node completely.

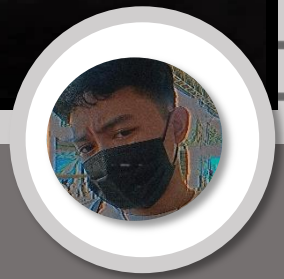
REN

Cestina, Justine

Hakuna Matata Group



Topic: Link List Operations

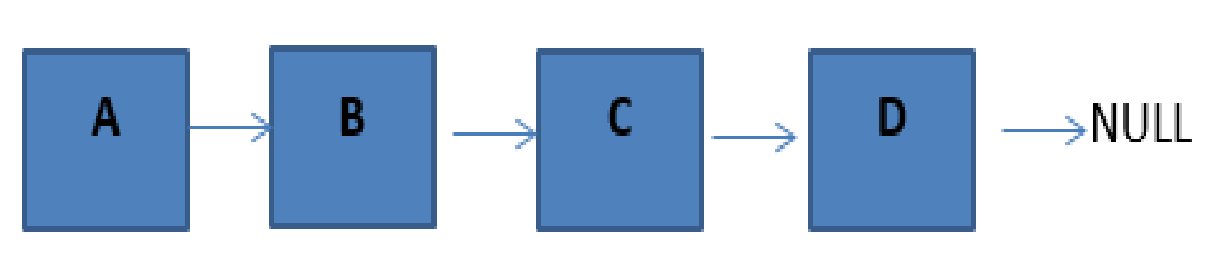


JUSTIN

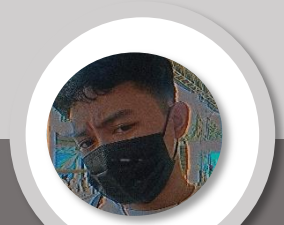
List Operations

- Search: Searches an element using the given key.
- Traversal: To traverse all the nodes one after another.

Traverse Operation example:




```
Void traverse (node * head)
{
    node * p=head
    while (p! = Null)
    {
      printf (“%C”,P⍰  data),
      P = P ⍰   next,
    }
}
```



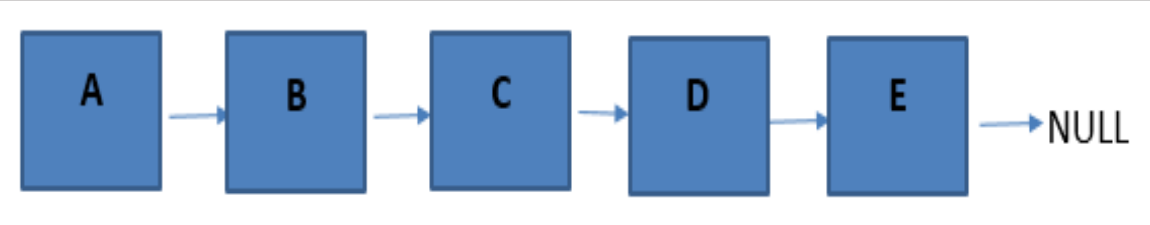
JUSTIN



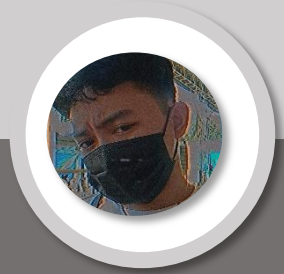


List Operations

Search Operation example:




```
void search (char s,node * head);
  {
                              node *p=head;
    while (p != NULL)
    {
    if(P⍰   data == s)
    {
print f("Element found" );
       break;
   }
P = P.next;
 }
  }
```



JUSTIN





Blangquisco, Reymar

Hakuna Matata Group

❖ **Topic: Types of Linked List**

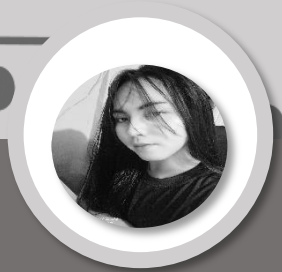
✔ *Singly Linked List*

✔ *Double Linked List*

✔ *Circular Linked List*

❖ **Topic: Hash Table**

✔ Good Hash Function

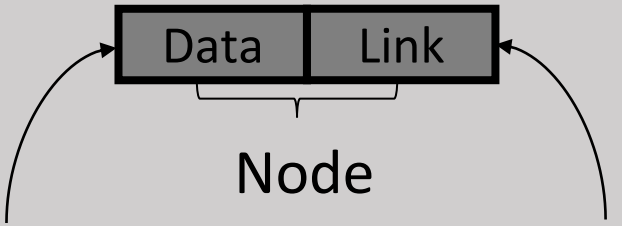


REYM

Single Linked List

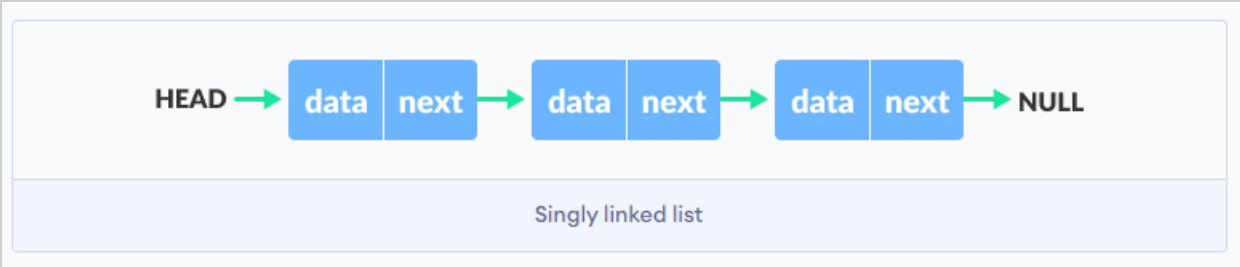
- Navigation or the flow is forward only(Head to Tail)

- a single linked list is a list made up of nodes that consists of two parts.
  - Data
  - Link/Pointer



Contains the Actual data

Contains the address of the next node of the list



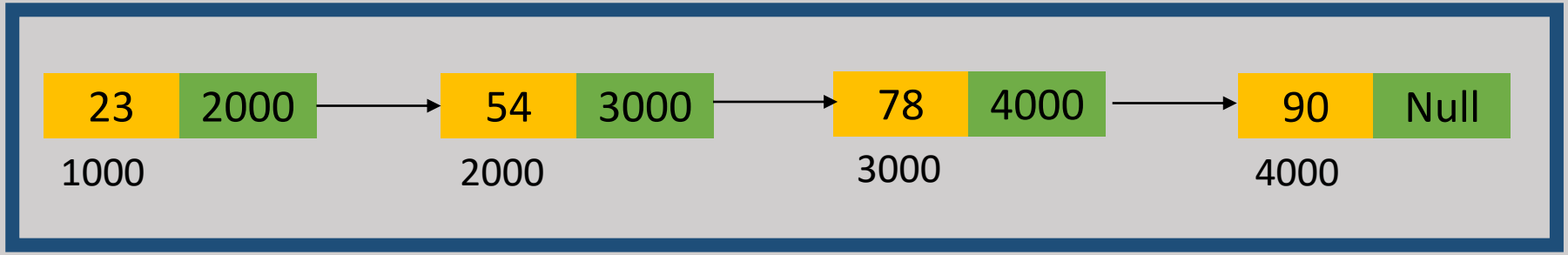




REYM


Single Linked List

Ex. List of numbers: 23, 54, 78, 90

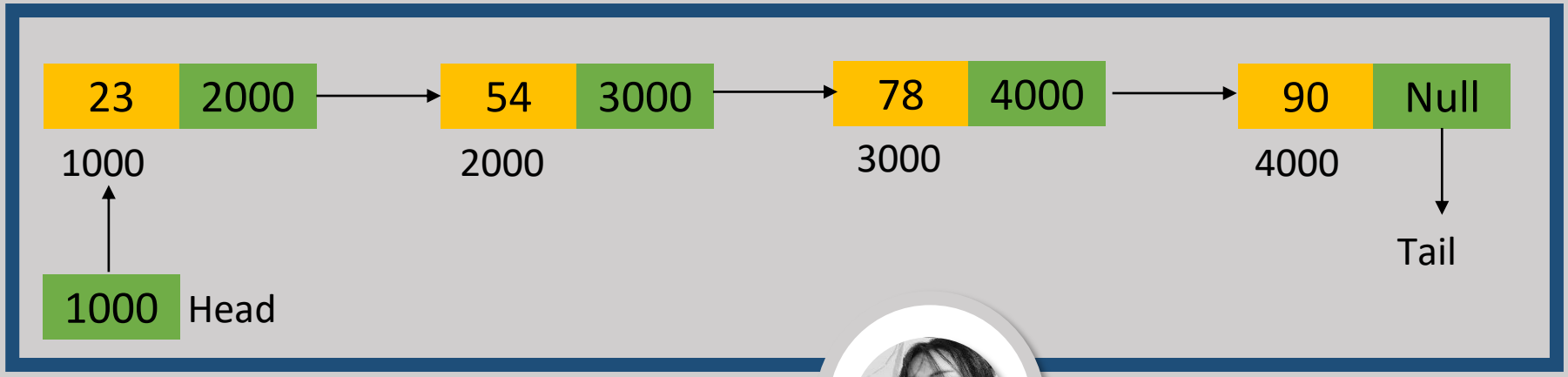


Memory

Data

Link/Pointer

How to access the first node of the linked list?







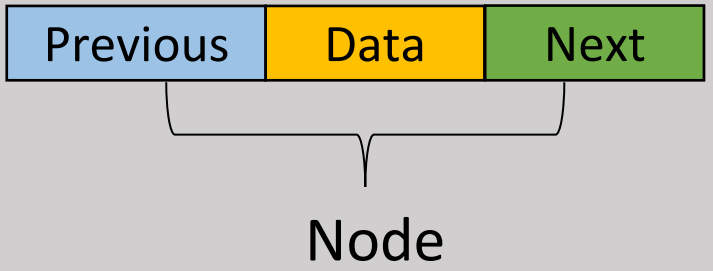
REYM


Double Linked List

- I.tems can be navigated forward and backward.

- a double linked list is a list made up of nodes that consists of three parts

  - Data
  - Link/Pointer
  - Prev/ Previous Pointer



Contains a link/next to the previous link

Contains the Actual data

Contains the address of the next node of the list

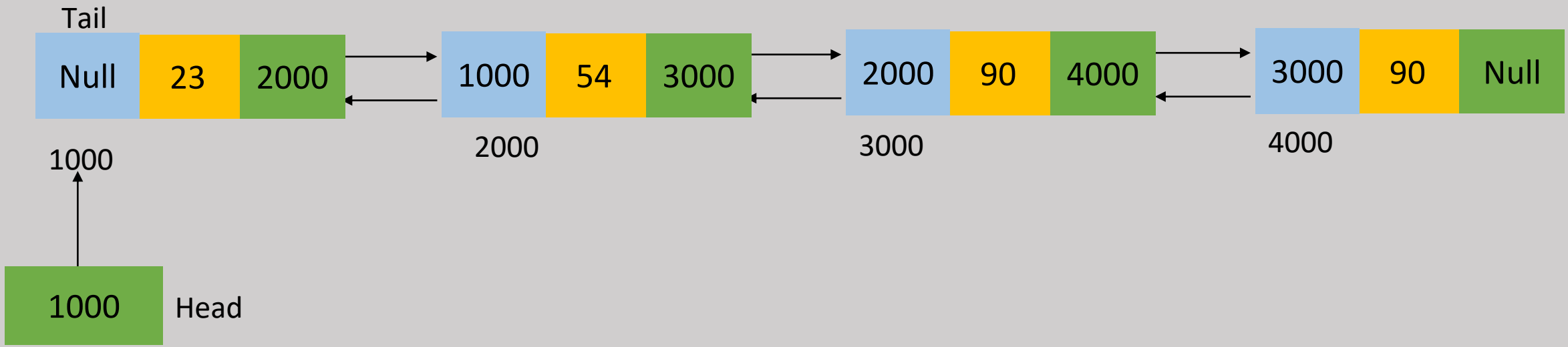




REYM


Double Linked List

Ex. List of numbers: 23, 54, 78, 90







REYM


Circular Linked List

- Circular Linked List is a variation of Linked list in which the first element points to the last element and the last element points to the first element. Both Singly Linked List and Doubly Linked List can be made into a circular linked list.

There are generally two types of circular linked lists:

- *Circular singly linked list*
- *Circular doubly linked list*

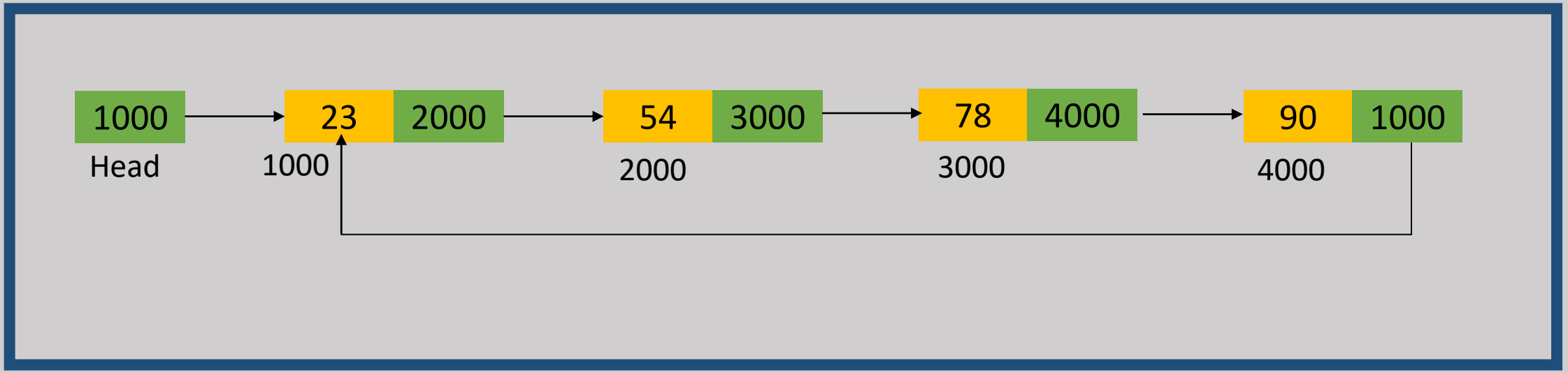




REYM


Circular Singly Linked List

Ex. List of numbers: 23, 54, 78, 90



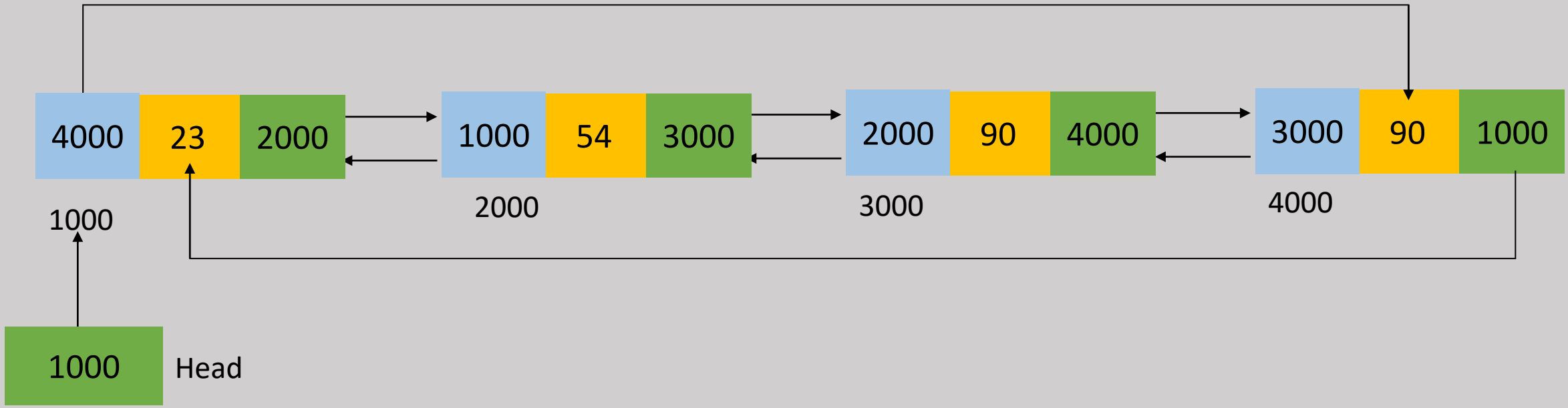




REYM


Circular Double Linked List

Ex. List of numbers: 23, 54, 78, 90







REYM


Viray, Roi Cedric

Hakuna Matata Group

Topic: Hash Table
Heap Data Structure

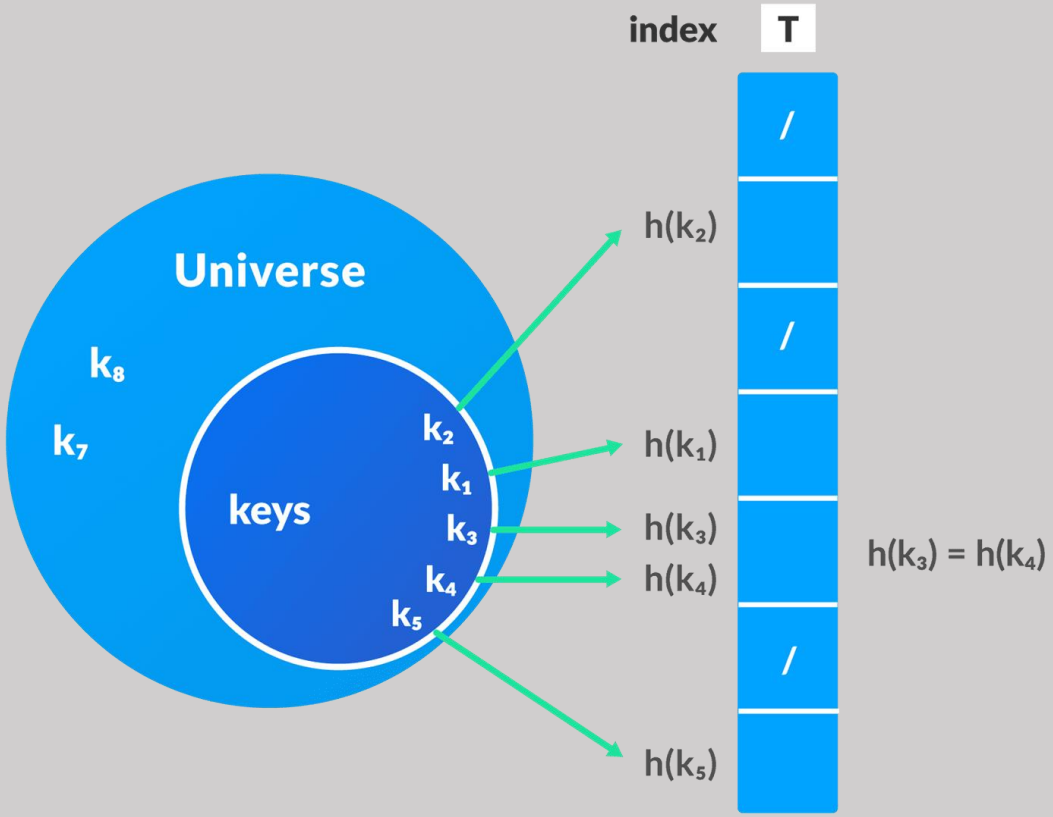
ROI

Hash Table

Hash Table is **a data structure which stores data in an associative manner**. In a hash table, data is stored in an array format, where each data value has its own unique index value.

- Key - unique integer that is used for indexing the values
- Value – data that are associated with keys

Hash Function - a function that takes a set of inputs of any arbitrary key and fits them into a table or other data structure that contains fixed-size elements.








ROI

Hashing Algorithm

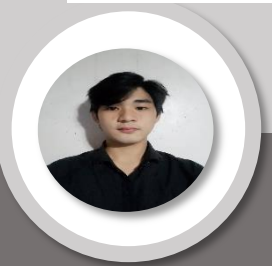
Hashing Algorithm
Calculation applied to a key to transform it into an address or fixed-size elements.

Numeric Key
h(k) = k % m

Alphanumeric key

h(k) = sum of ascii code % m
R = 82, o = 79, i = 105
= 266
h(k)= 266 % m

Character	Alt +	Character	Alt +	Character	Alt +
A	65	a	97	0	48
B	66	b	98	1	49
C	67	c	99	2	50
D	68	d	100	3	51
E	69	e	101	4	52
F	70	f	102	5	53
G	71	g	103	6	54
H	72	h	104	7	55
I	73	i	105	8	56
J	74	j	106	9	57
K	75	k	107	©	64
L	76	l	108	±	0177
M	77	m	109	µ	0181
N	78	n	110	™	0153
O	79	o	111	€	0163
P	80	P	112		
Q	81	Q	113		
R	82	R	114		
S	83	S	115		
T	84	T	116		
U	85	U	117		
V	86	V	118		
W	87	W	119		
X	88	X	120		
Y	89	Y	121		
Z	90	z	122		



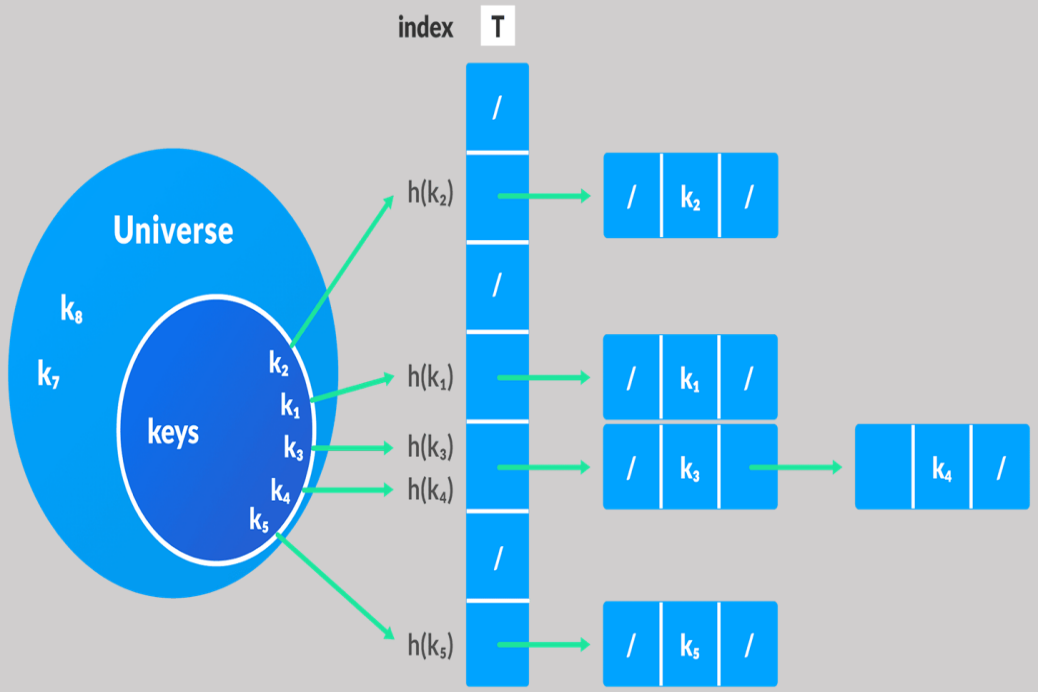
ROI

Hash Collisions

Hash Collision
When more than one value to be hashed by a particular hash function, hash to the same slot in the table or data structure (hash table) being generated by the hash function.

1. Collision resolution by chaining
In chaining, if a hash function produces the same index for multiple elements, these elements are stored in the same index by using a doubly-linked list.

2.Open addressing
Unlike chaining, open addressing doesn't store multiple elements into the same slot. Here, each slot is either filled with a single key or left NIL.



ROI

Different techniques used in open addressing are:

i. Linear Probing

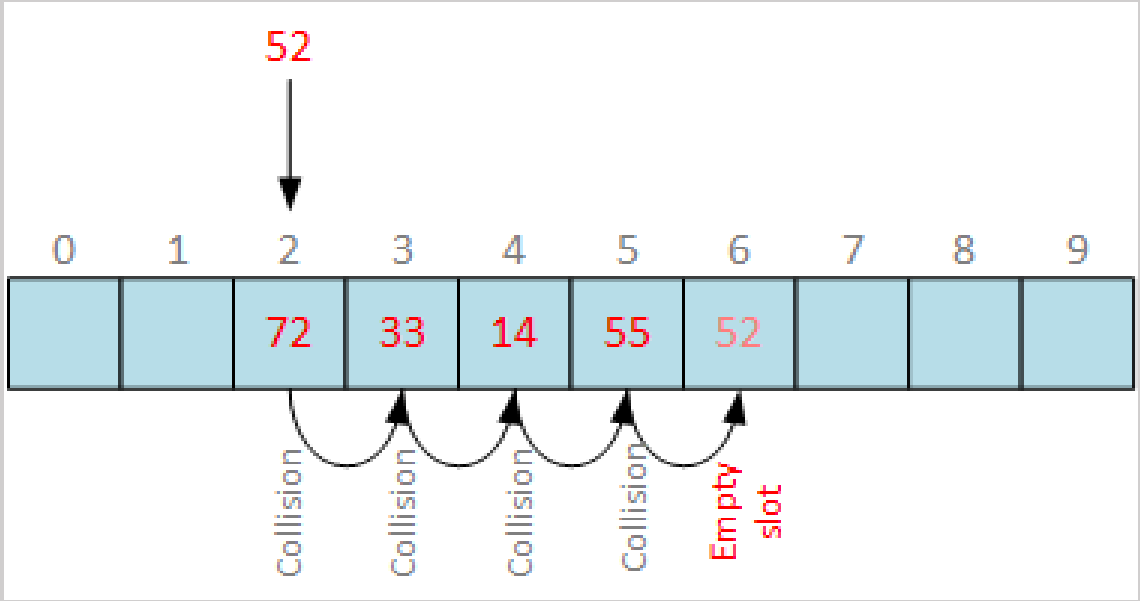
In linear probing, collision is resolved by checking the next slot.

$h(k, i) = (h'(k) + i) \bmod m$

where

$i = \{0, 1, ...\}$

$h'(k)$ is a new hash function



ii. Quadratic Probing

It works similar to linear probing but the spacing between the slots is increased (greater than one) by using the following relation.

$h(k, i) = (h'(k) + i^2) \bmod m$






ROI

Different techniques used in open addressing are:

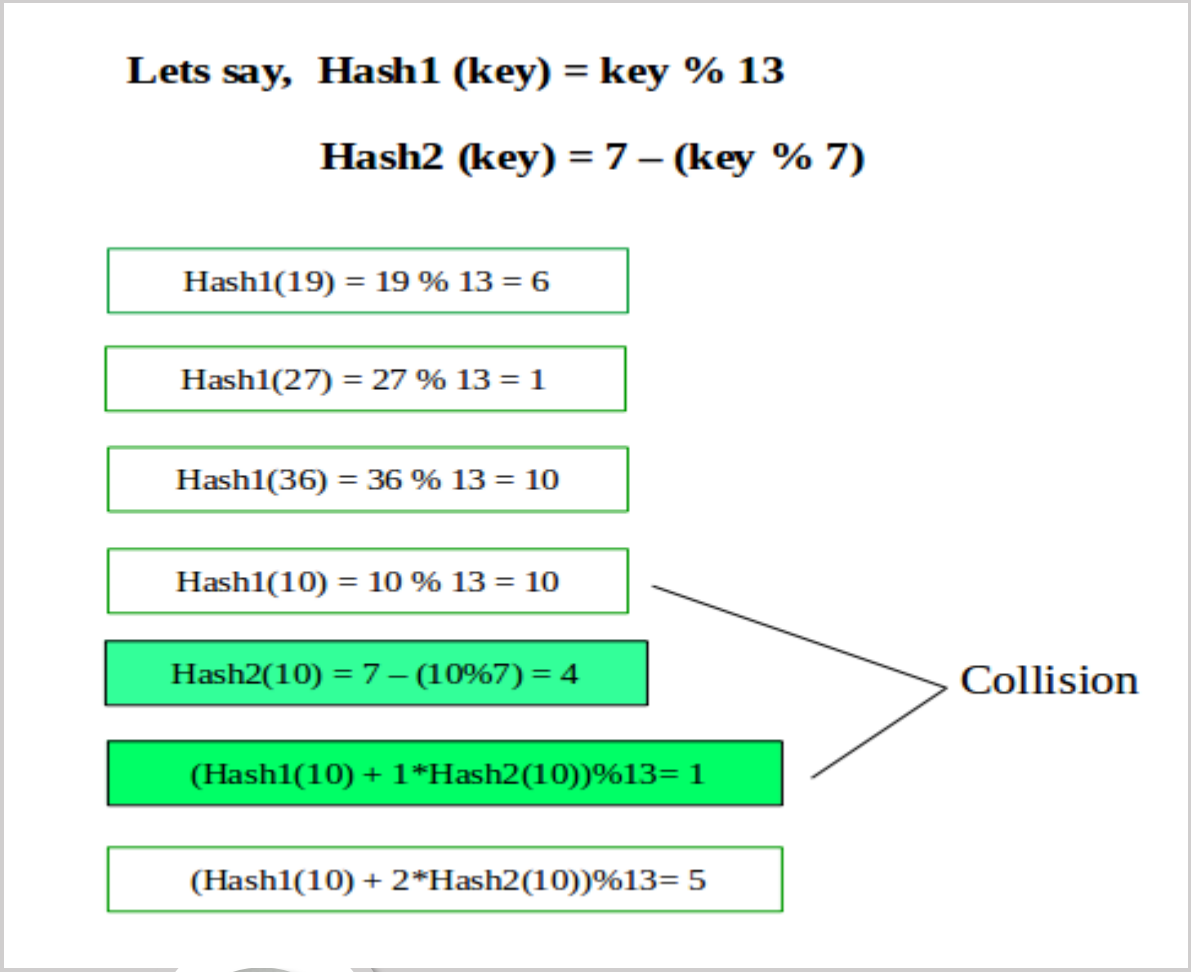
iii. Double hashing

If a collision occurs after applying a hash function h(k), then another hash function is calculated for finding the next slot.

hash2(key) = PRIME – (key % PRIME)

(hash1(k) + i * hash2(k)) % m

where : prime = smaller value of array size



ROI

Good Hash Functions

A good hash function may not prevent the collisions completely however it can reduce the number of collisions.

1. **Division Method**
2. **Multiplication Method**
3. **Universal Hashing**

DIVISION METHOD

h(k) = k mod m

Ex. m= 10 k = 123456
h(k) = 123456 mod 10
= 6

MULTIPLICATION METHOD

h(k) = ⌊m(kA mod 1)⌋

Ex. m= 10 k = 123456
A= 0.61803398

h(k)= ⌊10 (123456*0.61803398 mod 1)⌋
= ⌊10 (76299.882048 mod 1)⌋
= ⌊10 * 0.882048⌋
= 8

UNIVERSAL METHOD

h
x ≠ y
h (x) = h (y)

(full info) P(can create collision) = 1
(no info) P(can create collision) = 1/ m

F
1/1F1
1/1F1|{h: h(x) = h(y)}|

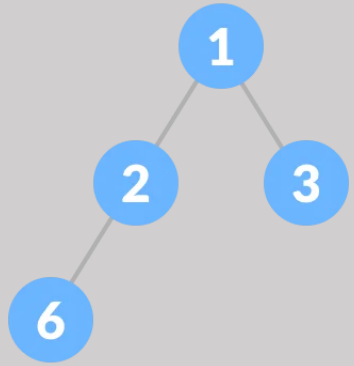




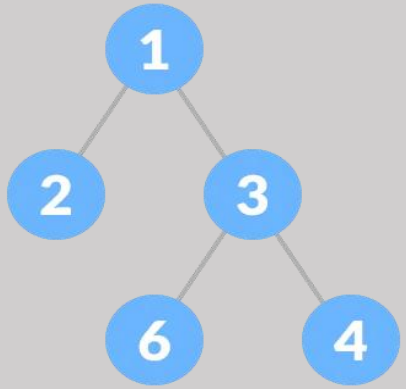

ROI

Heap Data Structures

Heap data structure is a complete binary tree that satisfies the heap property where any given root node is greater or smaller than the child node/s. This property is called

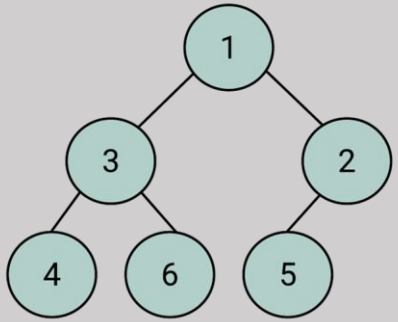


✗ Full Binary Tree
✓ Complete Binary Tree

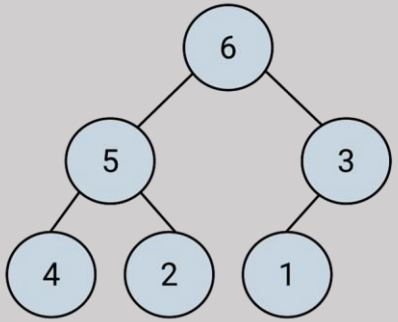


✓ Full Binary Tree
✗ Complete Binary Tree

Max heap and Min Heap



Min heap



Max Heap

ROI

Heap Data Structures

•HEAPIFY Heapify is the process of creating a heap data structure from a binary tree. It is used to create a Min-Heap or a Max-Heap.

•Insert Element into Heap
Step 1: Insert the node in the first available level order position.

Step 2: Compare the newly inserted node with its parent. If the newly inserted node is larger/smaller, swap it with its parent.

Step 3: Continue step 2 until the heap order property is restored.

•Delete Element from Heap
Step 1: Select the element to be deleted.
Step 2: Swap it with the last element.
Step 3: Remove the last element.
Step 4: Heapify the tree.

•Extract-Max/Min
Step 1: Swap the root and the last element.
Step 2: Remove the last element.
Step 3: Heapify the tree.






ROI

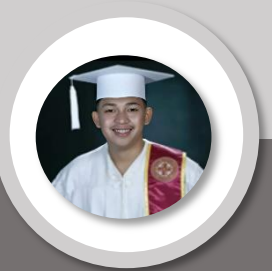
Bilbao, Gerome

Hakuna Matata Group

Topic: Fibonacci Heap



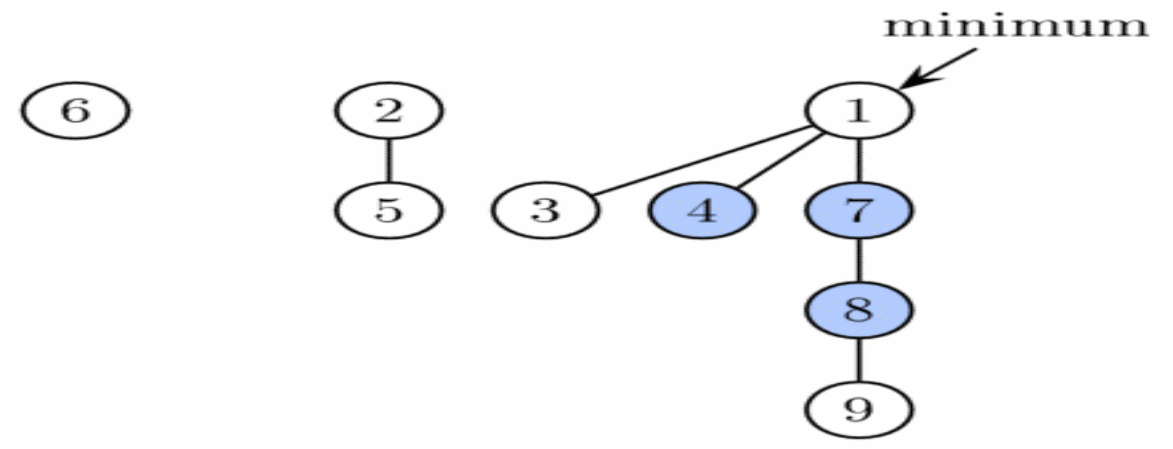


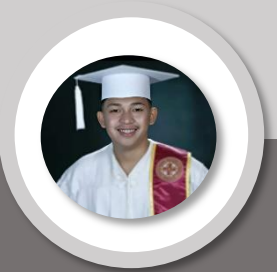


GERO

Fibonacci Heap

- A ***Fibonacci heap*** is a data structure for priority queue operations, consisting of a collection of heap-ordered trees. It has a better amortized running time than many other priority queue data structures including the binary heap and binomial heap.
- For Example:





GERO

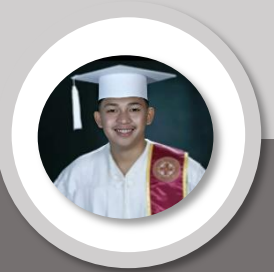
Fibonacci Heap

	Binomial Heaps (lazy)	Fibonacci Heaps
MAKE-HEAP	$O(1)$	$O(1)$
FIND-MIN	$O(1)$	$O(1)$
INSERT	$O(1)$	$O(1)$
DELETE-MIN	$O(\log n)$	$O(\log n)$
MELD	$O(1)$	$O(1)$
DELETE	$O(\log n)$	$O(\log n)$
DECREASE-KEY	$\boldsymbol{O(\log n)}$	$\boldsymbol{O(1)}$

Table 1: Amortized complexity of operations in Binomial and Fibonacci Heaps.







GERO


Sumayop, Edzel

Hakuna Matata Group

Topic: ***Decrease Key and Delete Node Operations on a Fibonacci Heap***






EDZE

Fibonacci Heap

What is Fibonacci heap?

- In computer science, a Fibonacci heap is a data structure for priority queue operations, consisting of a collection of heap-ordered trees. It has a better amortized running time than many other priority queue data structures including the binary heap and binomial heap. Michael L. Fredman and Robert E. Tarjan developed Fibonacci heaps in 1984 and published them in a scientific journal in 1987. Fibonacci heaps are named after the Fibonacci numbers, which are used in their running time analysis.







EDZE

Fibonacci Heap

How does Fibonacci heap work?

- A fibonacci heap is a data structure that consists of a collection of trees which follow min heap or max heap property. We have already discussed min heap and max heap property in the Heap Data Structure article. These two properties are the characteristics of the trees present on a fibonacci heap.
- In a fibonacci heap, a node can have more than two children or no children at all. Also, it has more efficient heap operations than that supported by the binomial and binary heaps.



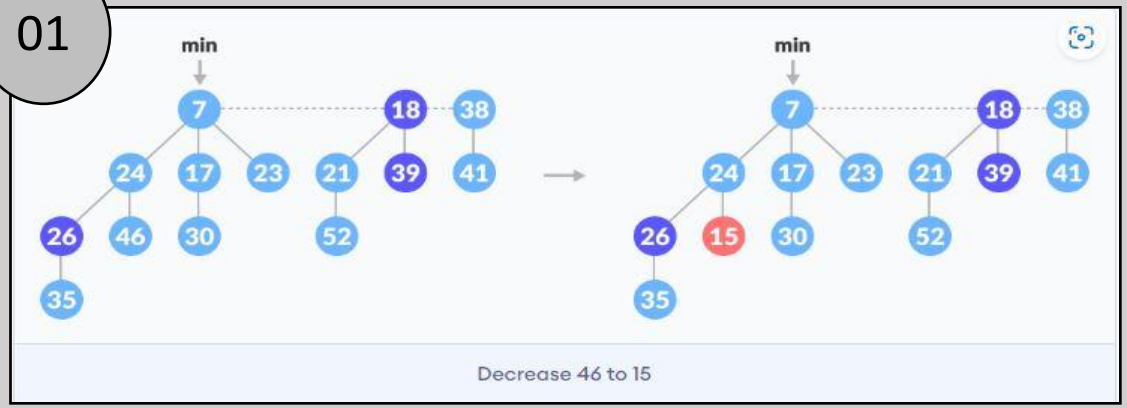




EDZE

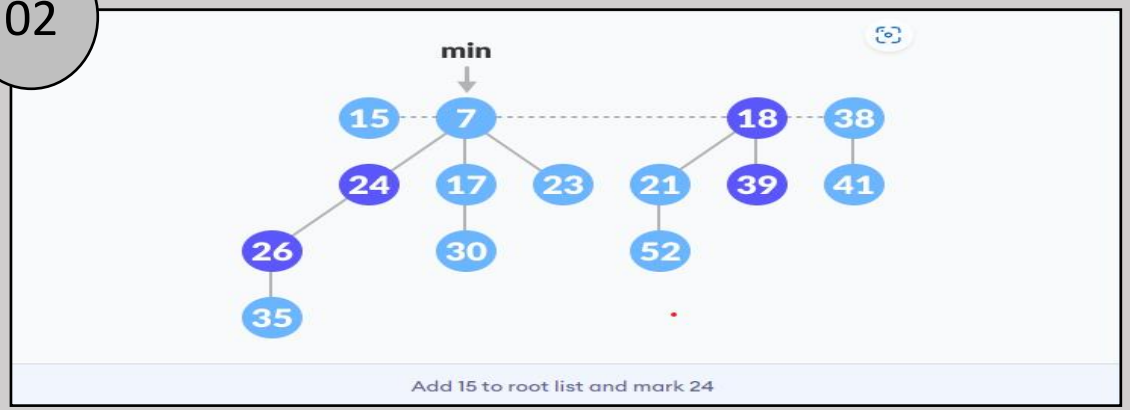
Fibonacci Heap

01



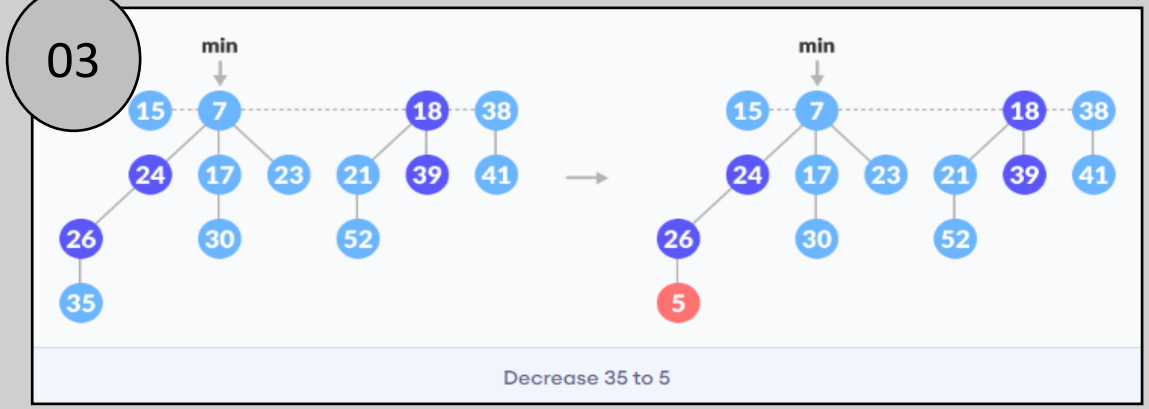
Decrease 46 to 15

02



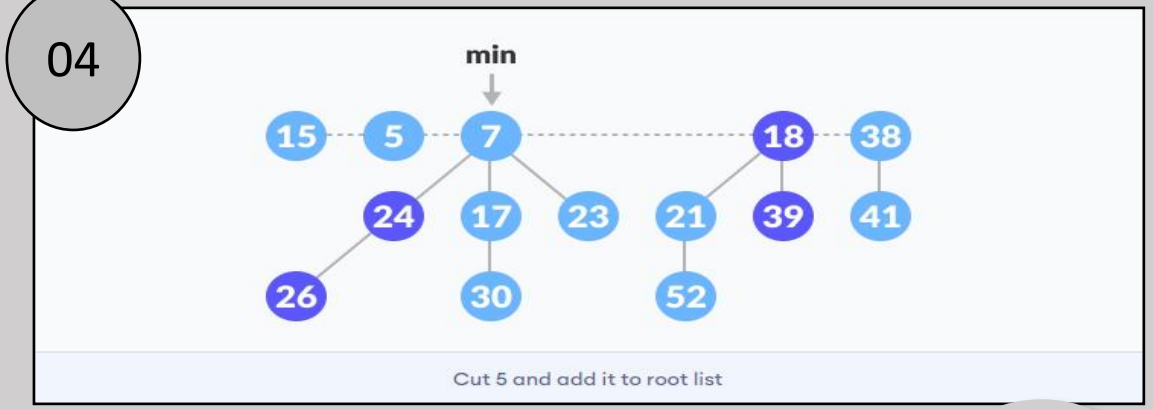
Add 15 to root list and mark 24

03

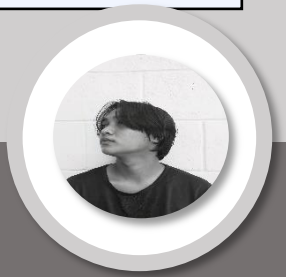


Decrease 35 to 5

04



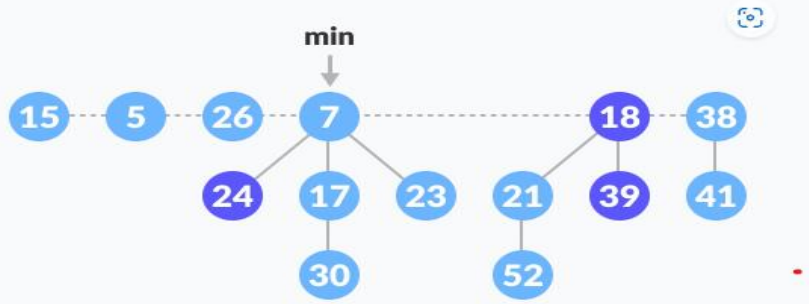
Cut 5 and add it to root list



EDZE

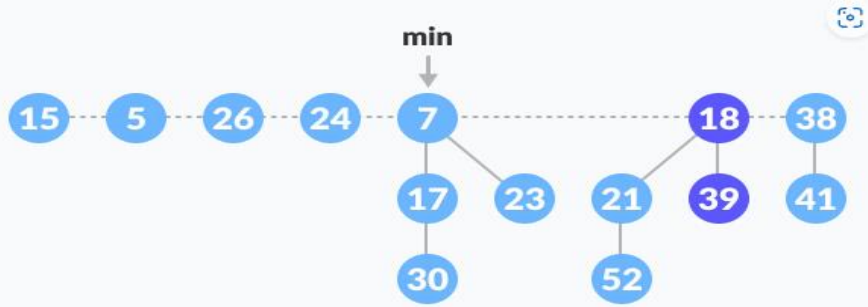
Fibonacci Heap

05



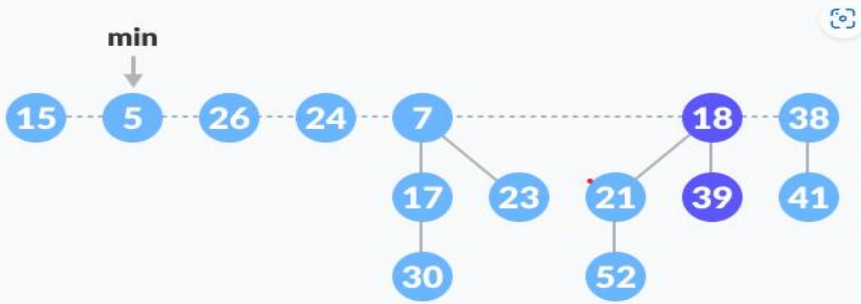
Cut 26 and add it to root list

06



Cut 24 and add it to root list

07



Mark 5 as min

DEFINITION

The Fibonacci numbers are the terms of a sequence of integers in which each term is the sum of the two previous terms with

$$F_1 = F_2 = 1, \quad F_n = F_{n-1} + F_{n-2}. \square$$

The first few Fibonacci numbers are

$$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, \ldots$$

EDZE





Thank You for Listening!