DS & A REVIEW

(1) WHAT & WHY

- STACKS
- QUEUE
- TREES
- LISTS

(2) RECURSION ★ & GENERAL PRACTICE

★ TAKE HOME TOOLKIT

HASH TABLE

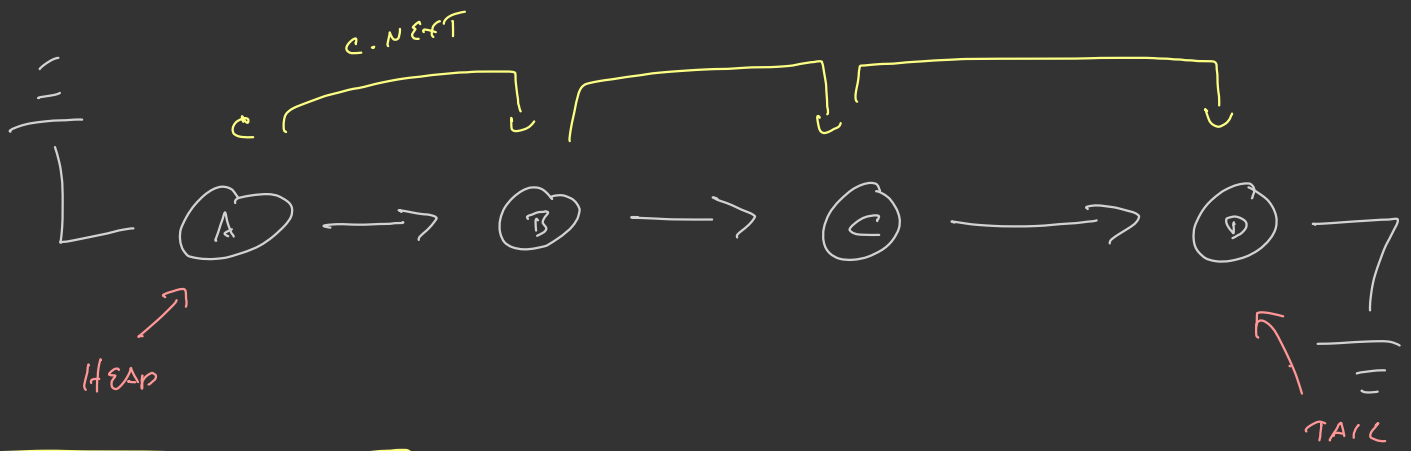
MAP

DICTIONARY

MAP

SET

LINKED LIST



TRAVERSE:

CURRENT = HEAD

WHILE LOOP

CURRENT → CURRENT.NEXT

RECURSIVELY

TRAVEL TO NEXT...

○ = NODES

- VALUE
- NEXT
- PREVIOUS

ARR.GET('1', NW, NW, (REQ, RES))

WHY?

LISTS

ITERATION

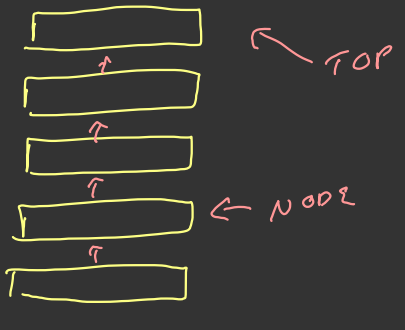
RE-USE

IN ORDER

STACK / QUEUE

STACK

FILO or LIFO



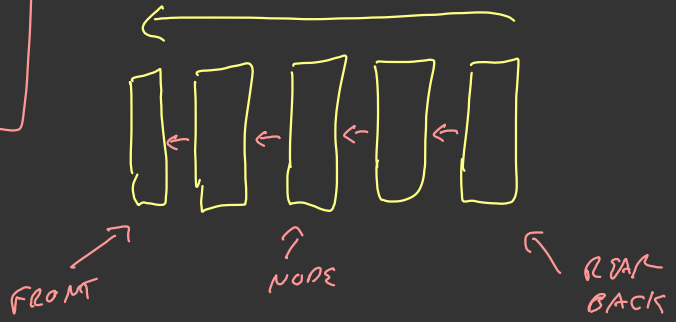
- .PUSH() ADD NODE TO TOP
- .POP() REMOVE & RETURN TOP
- .PEEK() RETURN TOP OR NULL
- .ISEMPTY() RETURN BOOL

WHY?

- PROCESSING ORDER MATTERS
- VOLATILE
- DESTRUCTIVE PROCESS

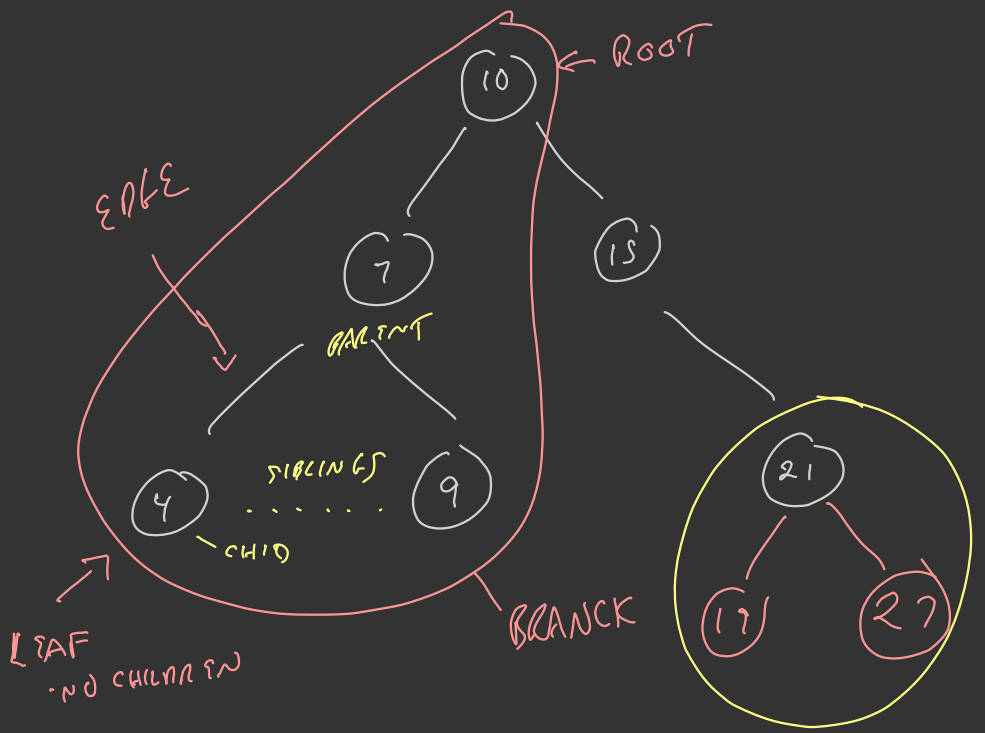
QUEUE

FIFO



- .ENQUEUE() ADD NODE TO BACK
- .DEQUEUE() REMOVE & RETURN FRONT
- .PEEK()
- .ISEMPTY() > SAME AS STACK

TREES



HEIGHT

2

BALANCED?

LEFT HEIGHT
vs
RIGHT HEIGHT
~ 1 ~

BINARY TREE

MAX: 2 CHILDREN PER NODE

LEFT

RIGHT

BINARY SEARCH TREE

LEFT = SMALLER

RIGHT = GREATER

K-ARY

- ALL THE CHILDREN

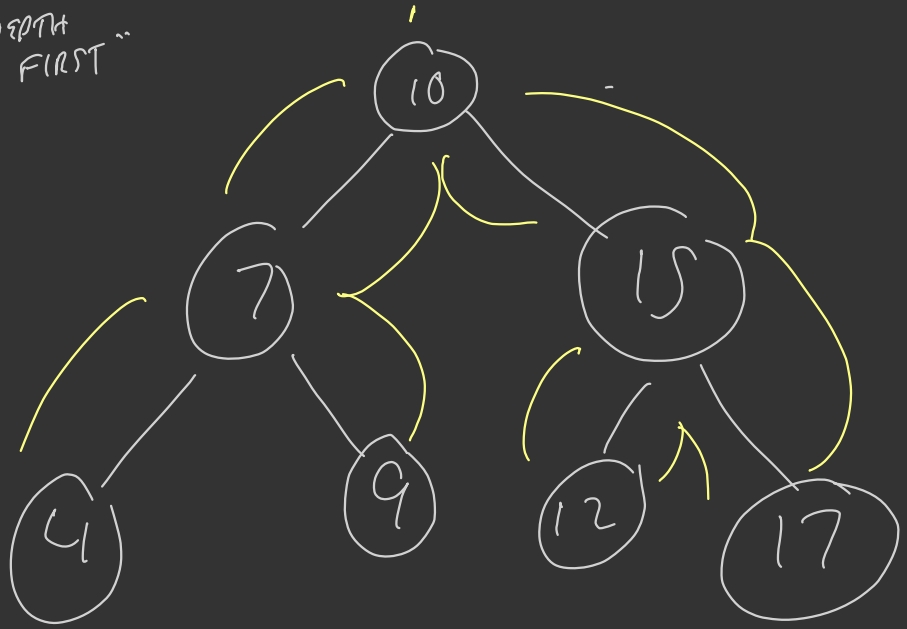
HEIRARCHY / ORDER / DECISION TREE
CYA

RECURSION "DEPTH FIRST"

PRE-ORDER.

IN-ORDER.

POST-ORDER.



10 7 4 9

S: O(H)

ITERATIVE

"BREADTH-FIRST"

LEVEL ORDER

WHILE LOOP

QUEUE

S: O(w)

ENQUE

WHILE QUEUE

~~10~~ ~~7~~ ~~15~~ 4 9 12 17

WIDTH

HASH TABLE

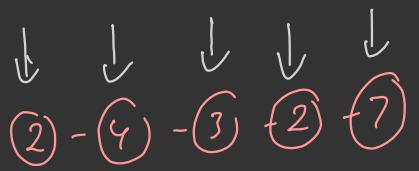
```
LET PEOPLE =
  NEW HASHTABLE()

People.ADD("ALLIE")
      .SET("ZACH")

People.HAS("JOHN")
  //TRUE
      .HAS("CATHY")
  //FALSE

People.GET("ALLIE")
  // 16
```

JESS	: 31
GINA	: 32
RACHEL	: 33
JAYA	: 34
DAVID	: 35
JOHN	: 53
ALLIE	: 16
ZACH	: 22



HT

- HAS(2) ? F
- ADD(2)
- HAS(4) ? F
- ADD(4)
- HAS(3) ? F
- ADD(3)
- HAS(2) ? T
- HAS(7) ?
- ADD(7)

OUT

[2

2	T
4	T
3	T
7	T

ADD TO OUT

KEYS

LIST : ITERATE

STACK / QUEUE : PROCESS

TREE : SEARCH, DECISION, BRANCH

HASH TABLE : LOOKUP, FIND, DICTIONARY

HASH TABLE

LET HT = NEW HASHTABLE (10)

HT

STORAGE (MAP)

ARRAY(10)

.ADD ("ALLIE", 16)

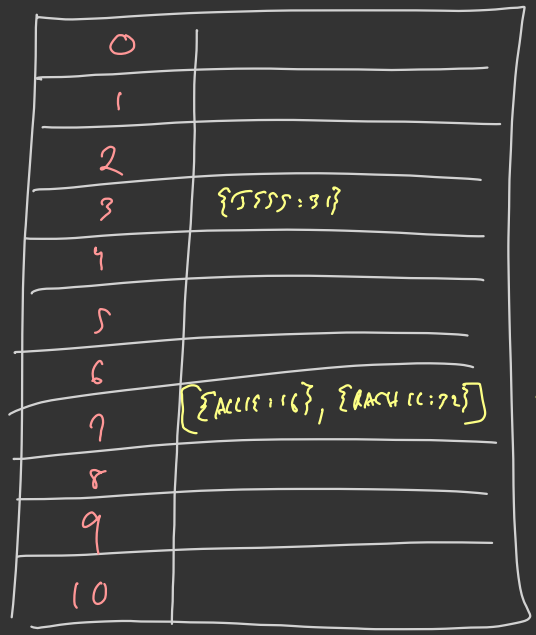
↓

.HASH (KEY)

BETWEEN 0 & LENGTH OF ARRAY

.ADD ('JESS') : 3

.ADD ('RACHEL') : 7



← BUCKETS

LOOKUP

O(1)

O(1)

.ADD(KEY, VALUE)

NUM = HASH(KEY); // — # ie. 7

TABLE[NUM].ADD({KEY: VAL});

.HAS(KEY)

NUM = HASH(KEY)

TABLE[NUM] ... ITERATE & FIND