Data Structure & Algorithms

Trainer: Nilesh Ghule

Stack and Queue

- Stack & Queue are utility data structures.
- Can be implemented using array or linked lists.
- Usually time complexity of stack & queue operations is O(1).
- Stack is Last-In-First-Out structure.
- Stack operations
  - push()
  - pop()
  - peek()
  - isEmpty()
  - isFull()*
- Simple queue is First-In-First-Out structure.
- Queue operations
  - push()
  - pop()
  - peek()
  - isEmpty()
  - isFull()*
- Queue types
  - Linear queue
  - Circular queue
  - Deque
  - Priority queue

Linear Queue

rear = push new element
front = pop an element

~~11~~	~~22~~	~~33~~	44	55	66
		f			r

arr[6]
front -
rear -

init:
f = -1;
r = -1;

Push:
r++;
arr[r] = val;

POP:
f++;
~~arr[f] = 0;~~

peek:
return arr[f+1];

q empty:
f == r

q full:
r == MAX-1

-1	0	1	2	3	4	5

fr

-1	0	1	2	3	4	5
	~~10~~	~~20~~	~~30~~	~~40~~		

fr

-1	0	1	2	3	4	5
	10	20	30	40	50	60

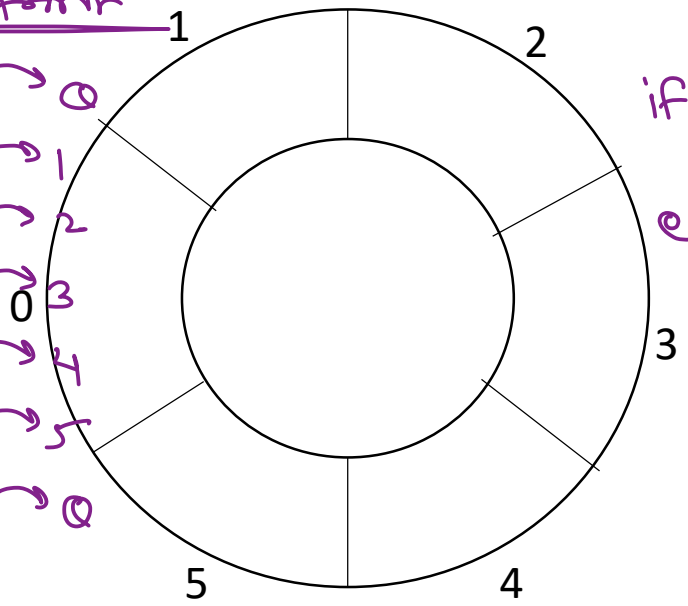
f r

Circular Queue

- In linear queue (using array) when *rear* reaches last index, further elements cannot be added, even If space is available due to deletion of elements from *front*. Thus space utilization is poor.
- Circular queue allows adding elements at the start of array if *rear* reaches last index and space is free at the start of the array.
- Thus *rear* and *front* can be incremented in circular fashion i.e. 0, 1, 2, 3, …, n-1. So they are said to be circular queue. 0, 1, 2, ...
- However queue full and empty conditions become tricky.

circular increment of rear & front

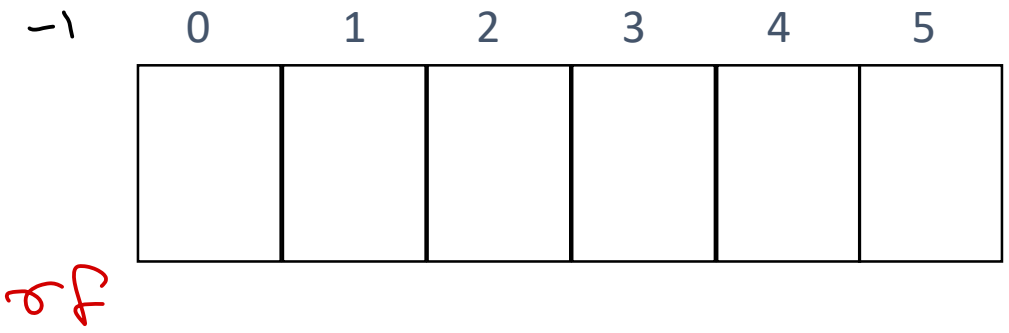
(-1+1)%6 → 0
(0+1)%6 → 1
(1+1)%6 → 2
(2+1)%6 → 3
(3+1)%6 → 4
(4+1)%6 → 5
(5+1)%6 → 0

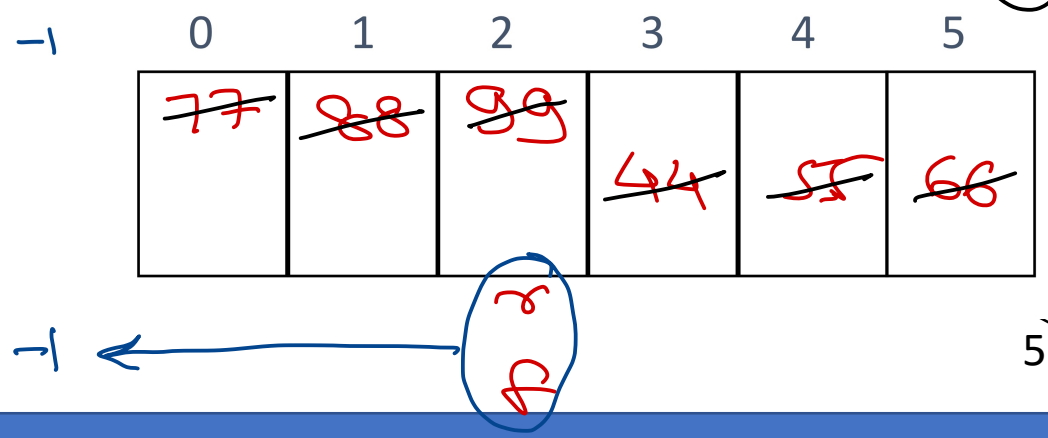


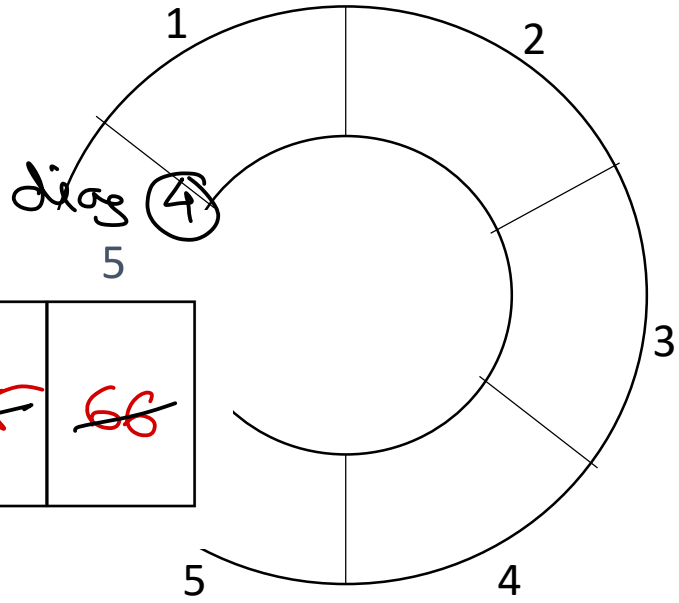

```
if(r==max-1)
    r=0;
else
    r++;
```

Circular Queue

```
init:
    r=-1;
    f=-1

push:
    r=(r+1)%.size;
    arr[r]=val;

pop:
    f=(f+1)%.size;
    if(f==r){
        f=-1;
        r=-1;
    }

peek:
    i=(f+1)%.size;
    return arr[i];

isempty:
    (r==f)&&(r==-1)
```







Circular Queue

diag (2)

-1	0	1	2	3	4	5
	~~11~~	~~22~~	~~33~~	44	55	66
	f					r

diag (1)

-1	0	1	2	3	4	5
	11	22	33	44	55	66
f						r

queue full.

(f == -1) && (r == size-1)

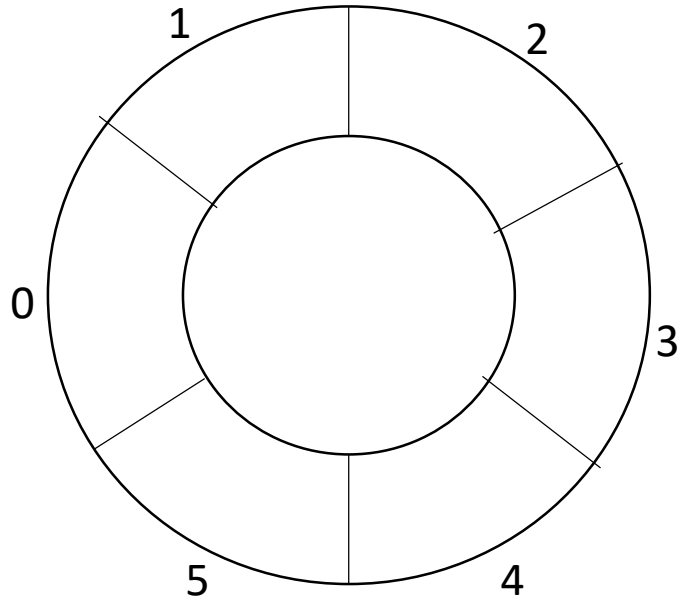
diag (3)

-1	0	1	2	3	4	5
	77 ~~11~~	88 ~~22~~	99 ~~33~~	44	55	66
	f r					

queue full.

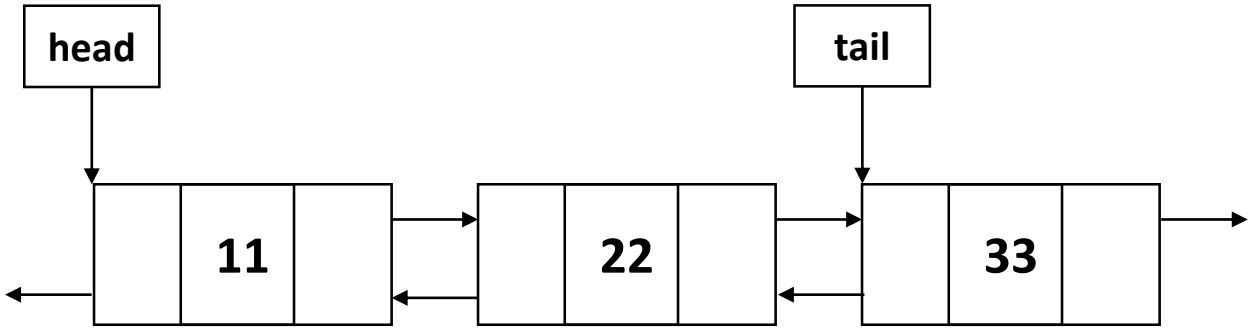
(r == f) && (r != -1)

OR



DeQueue

- In double ended queue, values can be added or deleted from front end or rear end.



Priority queue

- In priority queue, element with highest priority is removed first.

Doesn't follow FIFO.

Typically push/pop time complexity is not O(1).

Efficient way of implementing priority queue is heap data structures - time complexity is O(log n).

Stack

```
init:
    top = -1;

push:
    top++;
    arr[top] = val;

pop:
    top--;

peek:
    return arr[top];
```

```
isfull:
    top == size-1

is empty:
    top == -1
```

5	~~60~~
4	~~50~~
3	~~40~~
2	~~30~~
1	~~20~~
0	~~10~~

top -1

Stack / Queue in Java collections

- class java.util.Stack<E>
  - E push(E);
  - E pop();
  - E peek();
  - boolean isEmpty();

- interface java.util.Queue<E>
  - boolean offer(E e);
  - E poll();
  - E peek();
  - boolean isEmpty();

Expression Notations

A + B → infix
+ A B → prefix
A B + → postfix

Operator Priorities

( ) ← Highest
$
* /
+ − ← lowest

7 6 8 3 1 2 4 9 5

A + B * C − (D + E * F / G − H) + I $ J

A + B * C − (D + EF* / G − H) + I $ J

A + B * C − (D + EF*G/ − H) + I $ J

A + B * C − (DEF*G/+ − H) + I $ J

A + B * C − DFF*G/+H− + I $ J

A + B * C − DFF*G/+H− + IJ$

A + BC* − DFF*G/+H− + IJ$

ABC*+ − DFF*G/+H− + IJ$

ABC*+DFF*G/+H−− + IJ$

ABC*+DFF*G/+H−−IJ$+

+−+A*BC−+D/*EFGH$IJ

To solve any expr programmatically.

step 1: Convert given infix into postfix or prefix.

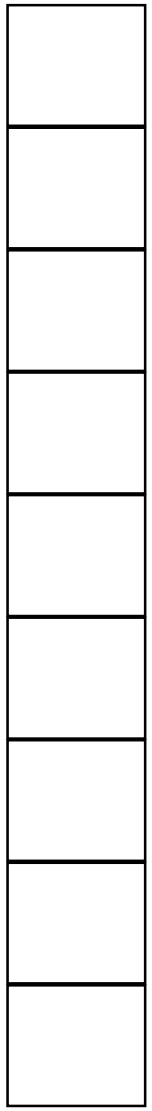
step 2: solve that postfix or prefix.

Infix to Postfix

- 5 + 9 – 4 * (8 – 6 / 2) + 1 $ (7 – 3)

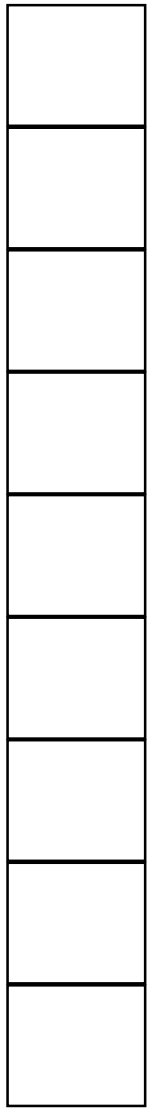
59+4862/-*-173-$+

1. traverse infix expr left to right.
2. if symbol is operand, then append to postfix.
3. if symbol is operator, push it on stack.
   - if priority of topmost operator >= priority of cur operator pop it from stack & append to postfix.
4. when all syms from infix are completed, pop all operators from stack and append to postfix.
5. if cur sym is (, push on stack.
6. if cur sym is), pop all operators from stack and append to postfix until (is found.
   . Also pop and discard (



Infix to Prefix

- 5 + 9 – 4 * (8 – 6 / 2) + 1 $ (7 – 3)



Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>