29-07-2024

1. Linear search :-

-> 'Data' :- This represents the collection of elements where we want to search for a specific 'target' element.

-> 'target' :- The value we are looking for within the data

-> for each element ('item' in data, it checks if 'item' equals 'targe':

-> If a match is found, it returns true, else it returns false.

Input = {10, 4, 2, 6, 8, 3}

target = 8

output = ~~e~~ 8 found at index 4.

2. Binary search :-

-> Data :- Sorted away to search through.

-> target : Element we are searching for low=0, high=n-1

-> Mid = $\frac{low + high}{2}$

-> If target is greater, ignore left half

-> If target is smaller, ignore right half.

Input :- [2, 4, 6, 8, 10, 12, 14, 16]

target = 10

output :- element 10 index '4'

3. Stack Application :-

-> Initialize an empty stack & empty list for output.

-> For each character in infix expression:

-> If character is operand, add to output lst.

-> while stack is not empty.

i. pop from stack to ouput.

(ii) return output as postfix.

Input :-

(A+B)*C-(D-E)*(F+G)

output :- AB+C*DE-FG+**-

4. Evaluating postfix

-> Postfix ["2", "3", "4", "5", "*"]

-> Push '2' & '3' onto the stack.

-> Push '5' operand on to the stack.

-> Final result '25' is left on the stack after processing all tokens.

Input :- 23+5* -> Postfix.

output :

'25'.

5. Balanced Equation

-> Exam '(a+b)':

-> Push '(' on to stack.

-> encounter 'a', '+', 'b'.

-> Encounter ')': POP '('

from matching opening.

-> return 'true', indicating that '(a+b)'.

Input:- '(a+b)'.

output:- True.

6. Queue

-> Array Implementation

-> The queue is implemented using a fixed-size array.

-> front & rear are indices to keep track of the front and rear of the quee

-> Size keeps track of the current number.

Input :- (10, 20, 30, 40, 50, 60)

Front Element :- 10

Deque Element : 10

" " :- 20

Queue Element: 40 50

Queue size = 2

Queue Empty

Infix - postfix

1. A+B*C

input	stack	output
A+B*C	-	-
+B*C	-	A
B*C	+	A
*C	+	AB
C	+*	AB
	+*	ABC
	+	ABC*
		ABC+

2. (A+B) * (C-D)

Input	stack	output
(A+B)-(C-D)	-	-
A+B)*(C-D)	(	-
+B)*(C-D)	(*	A
B)*(C-D)	(+	A
)*(C-D)	(+	AB
*(C-D)		AB+
*(C-D)	*	AB+
(C-D)	*(	AB+
-D)	*(	AB+C
D)	*(-	AB+C
)	*	AB+CD-
		AB+CD*

Post

③ AB

step

1. (

2.

3.

4.

5.

6.

④

Ste

1.

2.

3.

4.

5.

6.

7

8

Postfix to Sinfix

(3) AB+C*

step	postfix	Sinfix.
1	AB+C*	[ ]
2.	B+C*	[A]
3.	+C*	[A . B]
4.	C*	[(A+B)]
5.	*	[(A+B) . C]
6.		[(A+B) *C]

(4) ABC*+D

step	postfix	stack
1.	ABC*+D	[ ]
2.	BC* +D -	A
3.	C*+D -	A, B, C
4.	*+D -	A, (B*C)
5.	+D -	A, (B*C)
6.	D -	[(A+B*C)]
7.	—	[(A+(B*C)), D]
8.		[(A+(B*C)) - D]

30/07/24

Balancing paranthasis

(5) (A+B)*(C-D)

step	Read	stack
1.	(	[(]
2	A	[(]
3.	+	[(]
4.	B	[(]
5.	)	[ ]
6.	*	[ ]
7.	(	[(]
8.	(	[(]
9.	—	[(]
10.	D	[(]
11	)	[ ]

stack is Empty

(6) {A+(B*C)-D})

step	Read	stack
1.	{	[{]
2.	A	[{]
3.	+	[{]
4.	(	[{, (]
5.	B	[{, (]
6.	)	[{]
7	D	[{]
8.	}	[ ]
9.	)	[)]

Stack not Empty