Stacks & Queues

– Karun Karthik

Contents

Stack → Linear data structure

- follows LIFO, last in first out.
- operations → push : insert into top of stack
  pop : delete from top of stack.

Applications →

- by compilers to check for parenthesis
- to evaluate postfix expression
- to convert infix to postfix / prefix form.
- to store values during recursion & context during function call.
- to implement DFS of graph

Queue → Linear data structure

- follows FIFO, first in first out.
- operations → enqueue : insert element at end of queue
  dequeue : delete element at start of queue

Applications →

- Schedule jobs by CPU.
- to carry out FIFO basis like printing jobs.
- to implement BFS of graph

Types →

- Queue
- Circular Queue
- Doubly ended Queue
- Priority Queue.

① Implement a Stack using Linkedlist →

code →

```cpp
#include <bits/stdc++.h>
using namespace std;

struct Node{
    int data;
    Node* next;
};

Node* top;

void push(int data){
    Node* temp = new Node();
    if (!temp){
        cout << "\nStack Overflow";
        exit(1);
    }
    // add at the top and change top as new node
    temp->data = data;
    temp->next = top;
    top = temp;
}

int isEmpty(){
    // if top is null then empty
    return top == NULL;
}

int peek(){
    // if stack is not empty then return top node's data
    if (!isEmpty())
        return top->data;
    else
        exit(1);
}

void pop(){
    Node* temp;
    if(top == NULL){
        cout << "\nStack Underflow" << endl;
        exit(1);
    } else {
        temp = top;
        top = top->next;
        free(temp);
    }
}

```

② Implement a Queue using Linkedlist →

code →

```
class Node {
    int data;
    Node* next;
    Node(int d){
        data = d;
        next = NULL;
    }
};

class Queue {
    Node *front, *rear;

    Queue(){
        front = rear = NULL;
    }

    void enQueue(int x)
    {
        Node* temp = new Node(x);
        // if empty then node is both front and rear
        if (rear == NULL) {
            front = rear = temp;
            return;
        }
        // else add at end
        rear->next = temp;
        rear = temp;
    }

    void deQueue()
    {
        // if empty then return NULL
        if (front == NULL)
            return;
        // store front node
        Node* temp = front;
        front = front->next;

        // if front is NULL => no Nodes, change rear to NULL
        if (front == NULL)
            rear = NULL;
        // free node
        delete (temp);
    }
};
```

③ Implement a Stack using Queue →

If push, push into queue from rear end & pop & push all elements

If pop, pop from queue from front end.

Code →

```
class Stack {
  queue <int> q;

  public:

    // push operation
    void Push(int x) {
        int n = q.size();
        q.push(x);
        for (int i = 0; i < n; i++)
        {
            int value = q.front();
            q.pop();
            q.push(value);
        }
    }

    // pop operation
    int Pop() {
        int value = q.front();
        q.pop();
        return value;
    }

    // accessing top value
    int Top() {
        return q.front();
    }

    // finding size of stack
    int Size() {
        return q.size();
    }
};
```

(4) Implement a Queue using Stack →

→ use 2 stacks.

→ while pop( ), shift all elements in 1 stack to another.
& return top value.

code →

```cpp
class Queue {
  public:
    stack <int> in;
    stack <int> out;

    // push operation
    void Push(int x) {
        in.push(x);
    }

    // pop operation
    int Pop() {
        // shift in to out
        if (out.empty()){
            while (in.size()){
                out.push(in.top());
                in.pop();
            }
        }
        int x = out.top();
        out.pop();
        return x;
    }

    // peek operation
    int Top() {
        if (out.empty()){
            while (in.size()){
                out.push(in.top());
                in.pop();
            }
        }
        return out.top();
    }

    int Size() {
        return in.size()+out.size();
    }
};
```

5 Valid parenthesis

s = "{ }" → T

s = "( ){ }" → T

s = "{[ ]}" → T

s = ")[" → F

Eg s = "{[ ]( )}( )[ ]([ ])" → True.

→ if match found then pop, else push.

stack : [ ]	s = "{[ ]( )}( )[ ]([ ])"
stack : [{]	s = "{[ ]( )}( )[ ]([ ])"
stack : [{[ ]	s = "{[ ]( )}( )[ ]([ ])"
stack : [{~~[~~]	s = "{[ ]( )}( )[ ]([ ])"
stack : [{( ]	s = "{[ ]( )}( )[ ]([ ])"
stack : [{~~(~~]	s = "{[ ]( )}( )[ ]([ ])"
stack : [~~{~~]	s = "{[ ]( )}( )[ ]([ ])"
stack : [( ]	s = "{[ ]( )}( )[ ]([ ])"
stack : [~~(~~]	s = "{[ ]( )}( )[ ]([ ])"
stack : [[ ]	s = "{[ ]( )}( )[ ]([ ])"
stack : [~~[~~]	s = "{[ ]( )}( )[ ]([ ])"
stack : [( ]	s = "{[ ]( )}( )[ ]([ ])"
stack : [([ ]	s = "{[ ]( )}( )[ ]([ ])"
stack : [(~~[~~]	s = "{[ ]( )}( )[ ]([ ])"
stack : [~~(~~]	s = "{[ ]( )}( )[ ]([ ])"

∴ As the stack is empty & string is completely traversed the string is valid ∴ return true.

Code →

```
class Solution {
public:
    bool isValid(string s) {
        stack<char> st;
        for(auto i : s)
        {
            if (st.empty() || i == '(' || i == '{' || i == '[')
            {
                st.push(i);
            }
            else
            {
                if ((i == ')' && st.top() != '(') ||
                    (i == ']' && st.top() != '[') ||
                    (i == '}' && st.top() != '{')){
                    return false;
                }
                st.pop();
            }
        }
        return st.empty();
    }
};
```

TC → O(n)

SC → O(n)

6 Asteroid Collision →

+ve sign => right direction
-ve sign => left direction

only consider magnitude

If x & y collide then min(x, y) will be removed
If x == y then both will be removed.

Eg [5, 10, -5]   5, 10 will not collide
10, -5 will collide & -5 will be removed

result = [5, 10]

Eg [10, 6, -8, -8, 8, 9]

stack		[10, 6, -8, -8, 8, 9]	
stack	10,	[10, 6, -8, -8, 8, 9]	
stack	10, 6	[10, 6, -8, -8, 8, 9]	as 6 is +ve push
stack	10, 6	[10, 6, -8, -8, 8, 9]	as 6 & 8 will collide (opp directions), 6 will be removed
stack	10	[10, 6, -8, -8, 8, 9]	as 10 & 8 will collide (opp directions), 8 will be removed
stack	10	[10, 6, -8, -8, 8, 9]	as 10 & 8 will collide (opp directions), 8 will be removed
stack	10, 8	[10, 6, -8, -8, 8, 9]	as 8 is +ve push
stack	10, 8, 9	[10, 6, -8, -8, 8, 9]	as 9 is +ve push

result = [10, 8, 9]

TC → O(2n) ≃ O(n)   SC → O(n)
worstcase

code →

```
class Solution {
public:
    vector<int> asteroidCollision(vector<int>& asteroids) {

        vector<int> res;

        for(int i=0; i< asteroids.size(); i++){

            if(res.empty() || asteroids[i]>0)
                res.push_back(asteroids[i]);
            else {

                while(!res.empty() && res.back()>0 && res.back()<abs(asteroids[i])) {
                    res.pop_back();
                }

                if(!res.empty() && res.back()+asteroids[i]==0)
                    res.pop_back();
                else if(res.empty() || res.back()<0)
                    res.push_back(asteroids[i]);
            }
        }
        return res;
    }
};
```

7 Next greater element

$\rightarrow$ [2, 4, 1, 3, 1, 6]

2 → 4   3 → 6
4 → 6   1 → 6
1 → 3   6 → -1

Eg [4, 5, 2, 25]

4 → 5   2 → 25
5 → 25   25 → -1

→ Iterate from last & compare its value with top of stack
→ If stack is greater then its the next greater element
→ else keep popping till the next greater element is found.

Eg [11, 13, 3, 10, 7, 21, 26]

←

Stack = [ ]   [11, 13, 3, 10, 7, 21, 26]
Stack = [26]   [11, 13, 3, 10, 7, 21, 26]   26 → -1
Stack = [26, 21]   [11, 13, 3, 10, 7, 21, 26]   21 → 26
Stack = [26, 21, 7]   [11, 13, 3, 10, 7, 21, 26]   7 → 21
Stack = [26, 21, ~~7~~, 10]   [11, 13, 3, 10, 7, 21, 26]   pop 7, push 10
10 → 21
Stack = [26, 21, 10]   [11, 13, 3, 10, 7, 21, 26]   3 → 10
Stack = [26, 21, ~~10~~, ~~3~~, 13]   [11, 13, 3, 10, 7, 21, 26]   pop 3,10 push 13
13 → 21
Stack = [26, 21, 13]   [11, 13, 3, 10, 7, 21, 26]   11 → 13

res = [13, 21, 10, 21, 21, 26, -1]

Code →

```
class Solution
{
    public:
    //Function to find the next greater element for each element of the array.
    vector<long long> nextLargerElement(vector<long long> arr, int n){

        stack<long long> st;
        vector<long long> res(n);

        for(int i=n-1; i>=0 ; i--){
            long long currVal = arr[i];

            while(!st.empty() && st.top()<=currVal)
                st.pop();

            res[i] = st.empty()?-1:st.top();
            st.push(currVal);
        }
        return res;
    }
};

```

TC → O(n)

SC → O(n)

8 Next Smaller element →

→ entire approach is similar to next greater element except for comparision.

Tc → O(n)

Sc → O(n)

code →

```
vector<int> nextSmallerElement(vector<int> &arr, int n)
{
    stack<int> st;
    vector<int> res(n);
    for(int i=n-1; i>=0 ; i--){

        long long currVal = arr[i];

        while(!st.empty() && st.top()>=currVal)
            st.pop();

        res[i] = st.empty()?-1:st.top();
        st.push(currVal);
    }
    return res;
}
```

(9) Stock Span Problem → given price quotes of stock for n days. we need to find span of stock on any particular day.

max no. of consecutive days for which price <= curr day's price

Eg [100, 80, 60, 70, 60, 75, 85]
(indexes: 0 1 2 3 4 5 6)

stack = [stores indexes]

span = [0 | 0 | 0 | 0 | 0 | 0 | 0]
(indexes: 0 1 2 3 4 5 6)

if currentElement > stack.top
  pop stack
else :
  span = currentIndex − stack.top

→ push index into stack after processing →

Array (0 1 2 3 4 5 6)	Steps	stack	span
[**100**, 80, 60, 70, 60, 75, 85]	span of 1st element = 1	[0]	[~~0~~ 1, 0, 0, 0, 0, 0, 0]
[100, **80**, 60, 70, 60, 75, 85]	80 > 100 ⇒ false ∴ span = 1 − 0 = 1	[0, 1]	[1, ~~0~~ 1, 0, 0, 0, 0, 0]
[100, 80, **60**, 70, 60, 75, 85]	60 > 100 ⇒ false ∴ span = 2 − 1 = 1	[0, 1, 2]	[1, 1, ~~0~~ 1, 0, 0, 0, 0]
[100, 80, 60, **70**, 60, 75, 85]	70 > 60 ⇒ true ∴ pop 70 > 80 ⇒ false ∴ span = 3 − 1 = 2	[0, 1, 3]	[1, 1, 1, ~~0~~ 2, 0, 0, 0]
[100, 80, 60, 70, **60**, 75, 85]	60 > 70 ⇒ false ∴ span = 4 − 3 = 1	[0, 1, 3, 4]	[1, 1, 1, 2, ~~0~~ 1, 0, 0]
[100, 80, 60, 70, 60, **75**, 85]	75 > 60 ⇒ true ∴ pop 75 > 70 ⇒ true ∴ pop 75 > 80 ⇒ false span = 5 − 1 = 4	[0, 1, 5]	[1, 1, 1, 2, 1, ~~0~~ 4, 0]
[100, 80, 60, 70, 60, 75, **85**]	85 > 75 ⇒ true ∴ pop 85 > 80 ⇒ true ∴ pop 85 > 100 ⇒ false span = 6 − 0 = 6	[0, 6]	[1, 1, 1, 2, 1, 4, ~~0~~ 6]

span = [1 | 1 | 1 | 2 | 1 | 4 | 6]
(indexes: 0 1 2 3 4 5 6)

Code →

Tc → O(n)
Sc → O(n)

```cpp
class Solution
{
    public:
    //Function to calculate the span of stocks price for all n days.
    vector <int> calculateSpan(int price[], int n)
    {
        vector<int> span(n);
        stack<int> st;

        st.push(0);
        span[0] = 1;

        for(int i=1; i<n; i++){

            int currPrice = price[i];

            while(!st.empty() && currPrice >= price[st.top()])
                st.pop();

            if(st.empty()){
                span[i] = i+1;
            } else {
                span[i] = i-st.top();
            }

            st.push(i);
        }
        return span;
    }
};
```

(10) Celebrity Problem →

A Celebrity is a person, who is known to everyone & knows none.

Given a square matrix M & if i^{th} person knows j^{th} person then $M[i][j] = 1$, else 0.

Eg →

$$M = \begin{array}{cc} & \begin{matrix} 0 & 1 & 2 \end{matrix} \\ \begin{matrix} 0 \\ 1 \\ 2 \end{matrix} & \begin{bmatrix} [0, & 1, & 0], \\ [0, & 0, & 0], \\ [0, & 1, & 0] \end{bmatrix} \end{array} \quad n = 3.$$

1. create stack & push values from 0 to n−1.
2. do the following till stack more than has 1 value.
   - pop 1st element & set it to A
   - pop again & set it to B
   - if A knows B then push B else A.

stack → [ ] ⇒ stack [0, 1, 2]

⇒ stack [0, ~~1~~, ~~2~~] A = 2, B = 1 & $M[2][1] == 1$ (true) ∴ push 1 ⇒ stack [0, 1]

stack [~~0~~, ~~1~~] A = 1, B = 0 & $M[1][0] == 1$ (false) ∴ push 1 ⇒ stack [1]

∴ as stack has only 1 element, stop.

⇒ Now pop the stack & consider it as celebrity & check for

- anyone doesn't know celeb (!M[i][celeb])
- if celeb knows anyone (M[celeb][i])

} return −1.

∴ from i = 0 to 2 & celeb = 1

i = 0 (!M[0][1] or M[1][0]) = 0   (with $M[0][1]$ marked 0 and $M[1][0]$ marked 0)

i = 1 skip as celeb is i

i = 2 (!M[2][1] or M[1][2]) = 0

all are failed i.e no violation of conditions.

∴ return celeb i.e 1

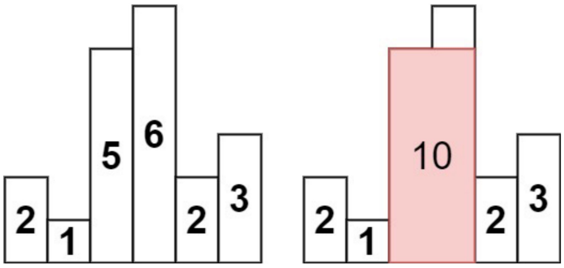
code →

TC = O(n)

SC = O(n)

```cpp
class Solution
{
    public:
    //Function to find if there is a celebrity in the party or not.
    int celebrity(vector<vector<int> >& M, int n) {

        stack<int> s;

        for(int i=0;i<n;i++)    s.push(i);

        // check and if is a celebrity then push into stack
        while(s.size()>1)
        {
            int a=s.top();
            s.pop();
            int b=s.top();
            s.pop();

            if(M[a][b]==1)
                s.push(b);
            else
               s.push(a);
        }

        int celeb = s.top();

        for (int i = 0; i < n; i++){
            // if i person doesn't know celeb or celeb knows anyone else
            // then return -1
            if ( (i!=celeb) && (!M[i][celeb]) || M[celeb][i] ))
                return -1;
        }

       return celeb;
    }
};
```

(11) Largest Rectangle in Histogram →



⇒ given an array of heights, return area of largest rectangle

Ans = 10.

0 1 2 3 4 5 Stack.

arr = [2, 1, 5, 6, 2, 3] [ ] area = 0 maxArea = 0

i = 0 [2, 1, 5, 6, 2, 3] [0] area = 0 maxArea = 0

→ i = 1 [2, 1, 5, 6, 2, 3] [0] area = 0 maxArea = 0

now arr[st.top()] > currElement ⇒ ht = arr[st.top()] & st.pop() [~~0~~]

& as stack is empty now, width = i & push(i)

∴ ht = 2 & width = 1 ∴ area = 2 & maxArea = ~~0~~ 2.

→ i = 2 [2, 1, 5, 6, 2, 3] [1] area = 0 maxArea = 2

now arr[st.top()] > currElement ⇒ false ∴ push(i)

→ i = 3 [2, 1, 5, 6, 2, 3] [1, 2] area = 0 maxArea = 2

now arr[st.top()] > currElement ⇒ false ∴ push(i)

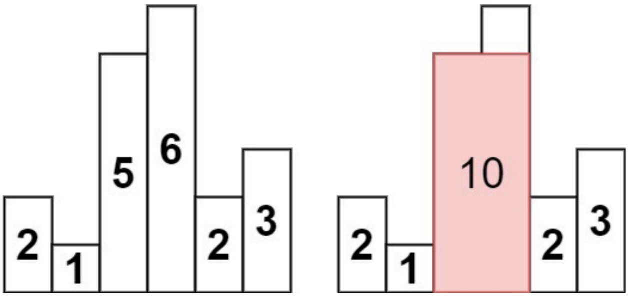
→ i = 4 [2, 1, 5, 6, 2, 3] [1, 2, 3] area = 0 maxArea = 2

now arr[st.top()] > currElement ⇒ ht = arr[st.top()] & st.pop() [1, 2, ~~3~~]

width = i − st.top() − 1 = 1 ∴ area = 6 * 1 = 6 maxArea = ~~2~~ 6.

& push(i)

https://www.linkedin.com/in/karun-karthik



0 1 2 3 4 5

→ [2, 1, 5, 6, 2, 3]   [1, 2]   area = 10   maxArea = 10

i = 4

now arr[st.top()] > currElement ⇒ ht = arr[st.top()] & st.pop()   [1, ~~2~~]

width = i − st.top() − 1 = 2   ∴ area = 5 * 2 = 10   maxArea = ~~2~~ 10.

& push(i)

0 1 2 3 4 5

→ [2, 1, 5, 6, 2, 3]   [1, 4]   area = 0   maxArea = 10

i = 5

now arr[st.top()] > currentElement ⇒ false ∴ push(i)

⇒ Last iteration to pop stack. ⇒ i = 6

0 1 2 3 4 5

→ [2, 1, 5, 6, 2, 3]   [1, 4, 5]   area = 3   maxArea = 10

[1, 4, ~~5~~]

ht = arr[st.top()] & pop() & as stack is not empty

width = i − st.top() − 1 = 1   ∴ area = 3 * 1 = 3   maxArea = 10

0 1 2 3 4 5

→ [2, 1, 5, 6, 2, 3]   [1, 4]   area = 3   maxArea = 10

[1, ~~4~~]

ht = arr[st.top()] & pop() & as stack is not empty

width = i − st.top() − 1 = 4   ∴ area = 2 * 4 = 8   maxArea = 10

0 1 2 3 4 5

→ [2, 1, 5, 6, 2, 3] [1,] area = 6 maxArea = 10

[~~1~~,]

ht = arr[st.top()] & pop() & as stack is empty

width = i = 6 ⇒ ∴ area = 1 * 6 = 6 maxArea = 10

∴ As stack is empty return maxArea = 10.

code →

TC → O(n)

SC → O(n)

```
class Solution {
public:
    int largestRectangleArea(vector<int>& heights) {
        stack < int > st;
        int maxArea = 0;
        int n = heights.size();

        for (int i = 0; i <= n; i++) {

            while (!st.empty() && (i == n || heights[st.top()] >= heights[i])) {

                int height = heights[st.top()];
                st.pop();
                int width;
                if (st.empty()){
                    width = i;
                } else {
                    width = i - st.top() - 1;
                }

                int area = width*height;
                maxArea = max(maxArea, area);
            }
            st.push(i);
        }
        return maxArea;
    }
};
```

(12) Sliding Window Maximum →

→ process first 'k' elements before pushing into result arr.

→ if dq.front() == i−k then pop-front (out of boundary case)

→ if nums[dq.back()] < nums[i] then pop-back
(meaningless to store smaller elements in window)

→ if i >= k−1 then push nums[dq.front()]

Eg nums = [1, 3, −1, −3, 5, 3, 6, 7] k = 3 res = [3, 3, 5, 5, 6, 7]

⇒ nums — deque — res

[1, 3, −1, −3, 5, 3, 6, 7]
0 1 2 3 4 5 6 7

deque: (empty) res: [ ]

i=0 [1, 3, −1, −3, 5, 3, 6, 7]
deque: 0
res: [ ]

i=1 [1, 3, −1, −3, 5, 3, 6, 7]
deque: 0
res: [ ]

→ dq.front == i−k → false
nums[0] < nums[i] (1 < 3)
∴ pop back & push i

deque: ~~0~~ 1

i=2 [1, 3, −1, −3, 5, 3, 6, 7]
deque: 1, 2
res: [3]

→ dq.front == i−k → false
nums[1] < nums[i] (3 < −1)
∴ false & push i

→ as i >= k−1
push nums[dq.front()] i.e. 3 into res

i=3 [1, 3, -1, -3, 5, 3, 6, 7] (indices 0 1 2 3 4 5 6 7)

deque: 1, 2, 3

res: [3, 3]

→ dq.front == i-K → false

$nums[2] < nums[i]$ (−1, −3)

∴ false & push i

→ as i >= k-1

push num[dq.front()] i.e. 3 into res

---

i=4 [1, 3, -1, -3, 5, 3, 6, 7] (indices 0 1 2 3 4 5 6 7)

order of pop

deque: ① ~~1~~, ③ ~~2~~, ② ~~3~~, 4

res: [3, 3, 5]

→ dq.front == i-K true ∴ pop front

$nums[3] < nums[i]$ (−3, 5) ∴ pop_back

$nums[2] < nums[i]$ (−1, 5) ∴ pop_back

& push(i)

→ as i >= k-1

push num[dq.front()] i.e. 5 into res

---

i=5 [1, 3, -1, -3, 5, 3, 6, 7] (indices 0 1 2 3 4 5 6 7)

order of pop

deque: 4, 5

res: [3, 3, 5, 5]

→ dq.front == i-K → false

$nums[4] < nums[i]$ (5, 3)

∴ false & push(i)

→ as i >= k-1

push num[dq.front()] i.e. 5 into res

---

i=6 [1, 3, -1, -3, 5, 3, 6, 7] (indices 0 1 2 3 4 5 6 7)

order of pop

deque: ② ~~4~~, ① ~~5~~, 6

res: [3, 3, 5, 5, 6]

→ dq.front == i-K → false

$nums[5] < nums[i]$ (3, 6) ∴ pop_back

$nums[4] < nums[i]$ (5, 6) ∴ pop_back

& push

→ as i >= k-1

push num[dq.front()] i.e. 6 into res

$\begin{array}{cccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \end{array}$

i = 7 [1, 3, -1, -3, 5, 3, 6, 7]

order of pop

① ~~6~~, 7

[3, 3, 5, 5, 6, 7]

→ dq.front == i − k → false

nums[6] < nums[i] ∴ pop_back (6 over nums[6], 7 over nums[i])

& push(i)

→ as i >= k − 1

push nums[dq.front()] i.e 7 into res

code →

TC → O(N)

SC → O(K)

```cpp
class Solution {
public:
    vector<int> maxSlidingWindow(vector<int>& nums, int k) {
        deque <int> dq;
        vector <int> ans;
        for (int i = 0; i < nums.size(); i++) {

            if (!dq.empty() && dq.front() == i - k)
                dq.pop_front();

            while (!dq.empty() && nums[dq.back()] < nums[i])
                dq.pop_back();

            dq.push_back(i);

            if (i >= k - 1)
                ans.push_back(nums[dq.front()]);
        }
        return ans;
    }
};
```

Find the rest on

https://linktr.ee/KarunKarthik

Follow Karun Karthik For More Amazing Content !