Stack

Thursday, 3 August 2023 6:32 AM

1) Implement two stacks in an Array

Notes

→ use stacks, the array from middle.

→ use flags both the stacks and expand half the stack2

→ Middle use of stack

→ if mid ≤ 1 then in stack1, and mid then in stack 2, then we can add any more thing to stack1 even if not have space in the array

Pseudocode

→ Keep both stacks from the corners of the array.

→ then we can insert thing in any stack as long as we have space.

```
class TwoStacks:
    def __init__(self, n):
        self.size = n
        self.arr = [None]*n
        self.top1 = -1
        self.top2 = self.size

    def push1(self, x):
        if self.top1 < self.top2 - 1:
            self.top1 += 1
            self.arr[self.top1] = x
            return True
        return False

    def push2(self, x):
        if self.top1 < self.top2 - 1:
            self.top2 -= 1
            self.arr[self.top2] = x
            return True
        return False

    def pop1(self):
        if self.top1 >= 0:
            x = self.arr[self.top1]
            self.top1 -= 1
            return x
        return None

    def pop2(self):
        if self.top2 < self.size:
            x = self.arr[self.top2]
            self.top2 += 1
            return x
        return None

    def size1(self):
        return self.top1 + 1

    def size2(self):
        return self.size - self.top2
```

2) Implement K stacks in an array

Q → stack number with value in the range [0, 1, ... k]

3)

3) Check Span Problem

Notes

```
def findSpan(arr):
    for i in range(len(arr)):
        span = 1
        j = i-1
        while j >=0 and arr[j] <= arr[i]:
            span += 1
            j -= 1
        print(span, end=" ")
```

Time : $O(n^2)$

Space : $O(1)$

Pseudocode

$$\text{span} = \left(\begin{array}{c}\text{index of current}\\ \text{element}\end{array}\right) - \left(\begin{array}{c}\text{index of closest greater}\\ \text{element on left side}\end{array}\right)$$

(*) If there's a greater element on left side, or else,

(*) (index of current element) + 1

```
def findSpan(arr):
    st = []
    st.append(0)
    print(1, end=" ")
    for i in range(1, n):
        while (len(st) >=0 and arr[st[-1]] <= arr[i]):
            st.pop()
        span = (i-1) if len(st) == 0 else i - st[-1]
        print(span, end=" ")
        st.append(i)
```

4) Previous Greater Element

Notes

```
def previous(arr):
    for i in range(len(arr)):
        ng = -1
        for j in range(i-1, -1, -1):
            if arr[j] > arr[i]:
                ng = arr[j]
                break
        print(ng, end=" ")
```

Time : $O(n^2)$

Space : $O(1)$

Pseudocode

```
def previous(arr):
    st = []
    for i in range(len(arr)):
        while len(st)>0 and st[-1] <= arr[i]:
            st.pop()
        if len(st) == 0 if else st[-1]
        print(ng, end=" ")
        st.append(arr[i])
```

5) Next Greater Element

Notes

```
def nextGreater(arr):
    for i in range(len(arr)):
        ng = -1
        for j in range(i+1, len(arr)):
            if arr[i] < arr[j]:
                ng = arr[j]
                break
        print(ng, end=" ")
```

Time : $O(n^2)$

Space : $O(1)$

Pseudocode

```
def nextGreater(arr):
    st = []
    res = [None]* len(arr)
    for i in range(len(arr)-1, -1, -1):
        while len(st)>0 and st[-1] <= arr[i]:
            st.pop()
        res[i] = -1 if len(st)==0 else st[-1]
        st.append(arr[i])
    for x in res:
        print(x, end=" ")
```

Time : $O(n)$

Space : $O(n)$

6) Largest Area in Histogram

Notes

```
def getMaxArea(arr):
    res = 0
    for i in range(len(arr)):
        cur = arr[i]
        for j in range(i-1, -1, -1):
            if arr[j] >= arr[i]:
                cur += arr[i]
            else:
                break
        for j in range(i+1, n):
            if arr[j] >= arr[i]:
                cur += arr[i]
            else:
                break
        res = max(res, cur)
    return res
```

Time : $O(n^2)$

Space : $O(1)$

Pseudocode

Method 1 : Brute Force

① Initialize res = 0

② Find Previous Smaller Element for every element in $O(n)$ time.

③ Find Next Smaller Element for every element in $O(n)$ time.

④ Do following for every element arr[i]

```
curl = arr[i]
curr += (i - pr[i] - 1) * arr[i]
curr += (nr[i] - i - 1) * arr[i]
res = max(res, curr)
return res
```

arr[ ]=	6	2	5	4	1	5	6
pr[ ]=	-1	-1	1	1	-1	4	5
nr[ ]=	1	4	3	4	7	7	7

res = 6 + 10 + 6 = 22

7) Largest Rectangle with all 1's

Hard

Notes

→ Consider every cell as a starting point. $O(n^2)$

→ Consider all sizes of rectangle with current cell as starting point. $O(n^2)$

→ For all current rectangle, check if it has all 1's, if yes then update the ans if the current area is more. $O(n^2)$

Time : $O(n^2 * n^2)$

Pseudocode

→ Run a loop from 0 to R-1

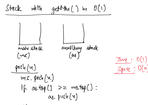
→ update the histogram for the current row

→ find the largest area in the histogram and update answer if required

```
def maxRectangle(mat):
    res = largestHist(mat[0])
    for i in range(1, len(mat)):
        for j in range(len(mat[i])):
            if mat[i][j]:
                mat[i][j] += mat[i-1][j]
        res = max(res, largestHist(mat[i]))
    return res
```

Time : $O(n^2)$

8) Stack with getMin() in O(1)




```
push(x)
    mS.push(x)
    if mS.top() >= mS.top():
        mS.push(x)
pop()
    if mS.top() == mS.top():
        mS.pop()
    mS.pop()
```

Time : $O(1)$

Space : $O(n)$

Design a Stack with getMin() in O(1) space

Hint method (Assuming all elements positive)

```
push(x)
pop()
getMin()
peek()
```

O(1) time.

Auxspace : O(1)

```
Stack S
int* min
void push(int x):
    if S.isEmpty():
        min = x
        S.push(x)
    else if x < min:
        S.push(2*x - min)
        min = x
    else:
        S.push(x)
int pop():
    if S.top() < 0:
        int t = min
        min = 2*min - t
        return t
    else:
        return S.top()
int getMin():
    return min
int peek():
    if S.top() < 0:
        return min
    return S.top()
void isEmpty():
    return S.isEmpty()
```

Fix negative numbers

```
void push(int x):
    if S.isEmpty():
        min = x
        S.push(x)
    else if (x < 2*min):
        S.push(2*x - min)
        min = x
    else:
        S.push(x)
int pop():
    int t = S.top()
    if t < 2*min:
        int temp = min
        min = 2*min - t
        return temp
    else:
        return t
int getMin():
    return min
int peek():
    int t = S.top()
    if t < 2*min:
        return min
    else:
        return t
void isEmpty():
    return S.isEmpty()
```

How does this work?

The push (2*x - min) only when x < min, so (2*x - min) is always going to be less than or equal to x, and x is going to be the new min.

9) Infix, Postfix and Prefix Introduction

Infix : x+y

Postfix : xyz+

Prefix : +xy

Advantages :

(*) Do not require parenthesis, precedence rules and associativity rules

(*) Can be evaluated by writing a program that traverses the given expression exactly once.

Precedence and Associativity

Operator	Associativity
^	Right to left
*, /	Left to right
+, -	Left to right

Precedence ↑

^ → exponent

Postfix and Prefix

Infix	Postfix	Prefix
x+y*z	+x*yz	x*y+z
(x+y)*z	*+xyz	*+yxz

Steps for postfix conversion

x+y*z# ⇒ (x+(y*z)#)

⇒ (x+(yz*))#

⇒ xyz*+

(x+y)*z# ⇒ ((x+y)*z)#

⇒ ((xy+)*z)#

⇒ xy+z*#

10) Infix to Postfix

① Create an empty stack, st

② Do following for every character x from left to right.

③ If x '+' :

(*) operand → output it.

(*) left parenthesis → push to stack.

(*) right parenthesis → pop from st until left parenthesis is found output the popped operators

(*) operator → If st is empty push x to st.

Else compare with st.top.

(i) Higher precedence (than st.top) push to st

(ii) Lower precedence, pop st.top and output until a lower precedence operator is found. Then push x to st.

(iii) Equal precedence, see associativity.

④ Pop and output everything from st.

Algorithm for evaluation of postfix

① Create an empty stack, st

② Traverse through every symbol x, if given postfix.

(*) If x is an operand push to st

(*) Else, x is an operator

(i) op1 = st.pop()

(ii) op2 = st.pop()

(iii) Compute op2 x op1 and push result to st

⑤ return st.pop()

11) Infix to Prefix

Notes ① Fully parenthesis

② Check associativity from Innermost to outermost

Pseudocode (using stack)

① Write fully parenthesis expression

② Reverse the expression

③ Create two stacks (operands and operators)

④ Create a loop from 0 to length of expression

⑤ If operand, push to operand stack

⑥ If operator, push to operator stack

⑦ If ')', push to operator stack

⑧ If '(', pop operator stack and create a new operand by concatenating the top two operands and push back to operand stack

⑨ Repeat ⑤ to ⑧ until operator stack is empty

⑩ Reverse the final output

12) Next Greater Element II

Hard

Notes

→ Run a loop from 0 to 2n-1

→ Find the next greater element in the array using stack

```
def nextGreaterElements(nums):
    st = []
    res = [None]* len(nums)
    for i in range(2*len(nums)-1, -1, -1):
        while len(st)>0 and st[-1] <= nums[i%len(nums)]:
            st.pop()
        if len(st) == 0 if else st[-1]
        st.append(nums[i%len(nums)])
    return res
```

13) Trapping Rain Water

Notes

→ Find the left and right boundary of the current cell

→ Find the minimum of the two boundary and subtract the current cell height from it.

→ Add it to the total water

```
def getWater(arr):
    res = 0
    for i in range(len(arr)):
        lmax = 0
        for j in range(i, -1, -1):
            if arr[j] > lmax:
                lmax = arr[j]
        rmax = 0
        for j in range(i, len(arr)):
            if arr[j] > rmax:
                rmax = arr[j]
        res += min(lmax, rmax) - arr[i]
    return res
```

Time : $O(n^2)$

Space : $O(1)$

Pseudocode

```
def getWater(arr):
    st = []
    res = 0
    for i in range(len(arr)):
        while st and arr[i] >= arr[st[-1]]:
            top = st.pop()
            if not st:
                continue
            distance = i - st[-1] - 1
            bounded_height = min(arr[i], arr[st[-1]]) - arr[top]
            res += distance * bounded_height
        st.append(i)
    return res
```

14) Stock Span Problem

Notes

```
def calculateSpan(price):
    for i in range(len(price)):
        span = 1
        j = i-1
        while j >=0 and price[j] <= price[i]:
            span += 1
            j -= 1
        print(span, end=" ")
```

Time : $O(n^2)$

Space : $O(1)$

Pseudocode

```
def calculateSpan(price):
    st = []
    st.append(0)
    print(1, end=" ")
    for i in range(1, n):
        while (len(st) >=0 and price[st[-1]] <= price[i]):
            st.pop()
        span = (i+1) if len(st) == 0 else i - st[-1]
        print(span, end=" ")
        st.append(i)
```

15) Remove K digits

Hard

Notes

→ Create a loop from 0 to n-1

→ Check if the current digit is smaller than the previous digit, if yes then pop the previous digit from the stack.

→ Repeat this until the stack is empty or the current digit is more than the previous digit.

→ Add the current digit to the stack.

→ Repeat this until the stack is empty or the number of digits to be removed is zero.

→ Remove the leading zeros from the stack.

→ Return the stack as a string.

```
def removeKdigits(num, k):
    st = []
    for i in range(len(num)):
        while st and int(st[-1]) > int(num[i]):
            st.pop()
        st.append(num[i])
    while k > 0:
        st.pop()
        k -= 1
    while st and st[0] == '0':
        st.pop(0)
    return ''.join(st) if st else '0'
```

16) Evaluate Reverse Polish Notation

Notes

```
def evalRPN(tokens):
    st = []
    for i in tokens:
        if i not in "+-*/":
            st.append(int(i))
        else:
            b = st.pop()
            a = st.pop()
            if i == '+':
                st.append(a+b)
            elif i == '-':
                st.append(a-b)
            elif i == '*':
                st.append(a*b)
            else:
                st.append(int(a/b))
    return st[0]
```

17) Exclusive Time of Functions

Notes

```
def exclusiveTime(n, logs):
    st = []
    res = [0]*n
    for i in range(len(logs)):
        if logs[i][2] == 'start':
            st.append(logs[i])
        else:
            start = st.pop()
            if start[0] == logs[i][0]:
                res[start[0]] += logs[i][1] - start[1] + 1
            else:
                res[start[0]] += logs[i][1] - start[1]
                for j in range(len(st)-1, -1, -1):
                    if st[j][0] == logs[i][0]:
                        st[j][1] += logs[i][1] - start[1] + 1
                        break
    return res
```

18) Next Greater Element III

Medium

Notes

```
def nextGreaterElement(n):
    st = []
    res = [None]* len(str(n))
    for i in range(len(str(n))-1, -1, -1):
        while len(st)>0 and st[-1] <= str(n)[i]:
            st.pop()
        res[i] = -1 if len(st)==0 else st[-1]
        st.append(str(n)[i])
    for x in res:
        print(x, end=" ")
```

Time : $O(n^2)$

Space : $O(n)$

19) Largest Area in Histogram

Notes

```
def getMaxArea(arr):
    res = 0
    for i in range(len(arr)):
        cur = arr[i]
        for j in range(i-1, -1, -1):
            if arr[j] >= arr[i]:
                cur += arr[i]
            else:
                break
        for j in range(i+1, n):
            if arr[j] >= arr[i]:
                cur += arr[i]
            else:
                break
        res = max(res, cur)
    return res
```

Time : $O(n^2)$

Space : $O(1)$

Better Solution $O(n^2)$ time.

① Initialize res = 0

② Find Previous Smaller Element for every element in $O(n)$ time.

③ Find Next Smaller Element for every element in $O(n)$ time.

④ Do following for every element arr[i]

```
curl = arr[i]
curr += (i - pr[i] - 1) * arr[i]
curr += (nr[i] - i - 1) * arr[i]
res = max(res, curr)
return res
```

arr[ ]=	6	2	5	4	1	5	6
pr[ ]=	-1	-1	1	1	-1	4	5
nr[ ]=	1	4	3	4	7	7	7

res = 6 + 10 + 6 = 22

Optimal Solution

```
def getMaxArea(arr):
    st = []
    res = 0
    for i in range(len(arr)):
        while st and arr[i] <= arr[st[-1]]:
            top = st.pop()
            if not st:
                continue
            distance = i - st[-1] - 1
            bounded_height = arr[top]
            res = max(res, bounded_height * distance)
        st.append(i)
    while st:
        top = st.pop()
        if not st:
            continue
        distance = len(arr) - st[-1] - 1
        bounded_height = arr[top]
        res = max(res, bounded_height * distance)
    return res
```

20) Largest Rectangle with all 1's

Hard

Notes

→ Consider every cell as a starting point. $O(n^2)$

→ Consider all sizes of rectangle with current cell as starting point. $O(n^2)$

→ For all current rectangle, check if it has all 1's, if yes then update the ans if the current area is more. $O(n^2)$

Time : $O(n^2 * n^2)$

Pseudocode

→ Run a loop from 0 to R-1

→ update the histogram for the current row

→ find the largest area in the histogram and update answer if required

```
def maxRectangle(mat):
    res = largestHist(mat[0])
    for i in range(1, len(mat)):
        for j in range(len(mat[i])):
            if mat[i][j]:
                mat[i][j] += mat[i-1][j]
        res = max(res, largestHist(mat[i]))
    return res
```

Time : $O(n^2)$

21) Remove Duplicate Letters

Notes

```
def removeDuplicateLetters(s):
    st = []
    for i in range(len(s)):
        if s[i] not in st:
            st.append(s[i])
    res = ''.join(st)
    return res
```

Pseudocode

```
def removeDuplicateLetters(s):
    st = []
    for i in range(len(s)):
        while st and s[i] <= st[-1] and s[i] in st and s[i] != st[-1]:
            st.pop()
        if s[i] not in st:
            st.append(s[i])
    res = ''.join(st)
    return res
```

22) Minimum Cost of Ropes

Notes

```
def getMinCost(arr):
    res = 0
    while len(arr) > 1:
        a = arr.pop(0)
        b = arr.pop(0)
        res += a + b
        arr = sorted(arr + [a+b])
    return res
```

Time : $O(n^2)$

Space : $O(1)$

Pseudocode

```
def getMinCost(arr):
    st = []
    res = 0
    while len(st) > 1:
        a = st.pop()
        b = st.pop()
        res += a + b
        st = sorted(st + [a+b])
    return res
```

23) Minimum Number of Coins

Notes

```
def getMinCoins(coins, amount):
    res = 0
    for i in range(len(coins)-1, -1, -1):
        while amount >= coins[i]:
            res += 1
            amount -= coins[i]
    return res
```

Time : $O(n^2)$

Space : $O(1)$

Pseudocode

```
def getMinCoins(coins, amount):
    st = []
    res = 0
    for i in range(len(coins)-1, -1, -1):
        while amount >= coins[i]:
            st.append(coins[i])
            amount -= coins[i]
    return len(st)
```

24) Sum of Subarray Minimums

Notes

```
def getSumOfSubarrayMinimums(arr):
    res = 0
    for i in range(len(arr)):
        for j in range(i, len(arr)):
            res += min(arr[i:j+1])
    return res
```

Time : $O(n^3)$

Space : $O(1)$

Pseudocode

```
def getSumOfSubarrayMinimums(arr):
    st = []
    res = 0
    for i in range(len(arr)):
        while st and arr[i] <= arr[st[-1]]:
            st.pop()
        if not st:
            res += arr[i] * (i+1)
        else:
            res += arr[i] * (i - st[-1])
        st.append(i)
    return res
```

25) Find the Pivot Index

Notes

```
def pivotIndex(nums):
    for i in range(len(nums)):
        if sum(nums[:i]) == sum(nums[i+1:]):
            return i
    return -1
```

Pseudocode

```
def pivotIndex(nums):
    st = []
    res = 0
    for i in range(len(nums)):
        while st and nums[i] <= nums[st[-1]]:
            st.pop()
        if not st:
            res += nums[i] * (i+1)
        else:
            res += nums[i] * (i - st[-1])
        st.append(i)
    return res
```

26) Find the Pivot Index II

Hard

Notes

```
def pivotIndex(nums):
    st = []
    res = 0
    for i in range(len(nums)):
        while st and nums[i] <= nums[st[-1]]:
            st.pop()
        if not st:
            res += nums[i] * (i+1)
        else:
            res += nums[i] * (i - st[-1])
        st.append(i)
    return res
```

27) Minimum Number of Coins II

Notes

```
def getMinCoins(coins, amount):
    res = 0
    for i in range(len(coins)-1, -1, -1):
        while amount >= coins[i]:
            res += 1
            amount -= coins[i]
    return res
```

Pseudocode

```
def getMinCoins(coins, amount):
    st = []
    res = 0
    for i in range(len(coins)-1, -1, -1):
        while st and st[-1] <= coins[i]:
            st.pop()
        if not st:
            res += amount * (i+1)
        else:
            res += amount * (i - st[-1])
        st.append(coins[i])
    return res
```