Q. Implement Stack using Arrays

#include<bits/stdc++.h>

using namespace std;

class Stack{

public:

int \*arr;

int Top;

Stack(){

int size = 1000;

arr = new int[size];

Top = -1;

}

void push(int val){

Top++;

arr[Top] = val;

}

int top(){

if(Top == -1){

cout<<"Stack is empty"<<endl;

return -1;

}

else{

return arr[Top];

}

}

void pop(){

if(Top == -1){

cout<<"Stack is empty"<<endl;

return;

}

Top--;

}

int size(){

return Top+1;

}

bool isEmpty(){

if(Top==-1){

return true;

}

else{

return false;

}

}

};

int main(){

Stack s;

// cout<<s.top()<<endl;

// s.pop();

s.push(4);

s.push(5);

s.push(6);

while(s.size()>0){

cout<<s.top()<<endl;

s.pop();

}

cout<<endl;

}

// Advantage

Easy to implement.

Memory is saved as pointers are not involved.

// disadvantage

It is not dynamic i.e., it doesn’t grow and shrink depending on needs at runtime. [But in case of dynamic sized arrays like vector in C++, list in Python, ArrayList in Java, stacks can grow and shrink with array implementation as well].

The total size of the stack must be defined beforehand.

-------------------------------------------------------------------------------------------------------------------------

Stack using Linked List

#include<bits/stdc++.h>

using namespace std;

struct StackNode{

int data;

StackNode \*next;

StackNode(int d){

data = d;

next = NULL;

}

};

class Stack{

public:

StackNode \*Top;

int size;

Stack(){

int size = 0;

Top = NULL;

}

void push(int val){

StackNode\* element = new StackNode(val);

element->next = Top;

Top = element;

size++;

}

int top(){

if(Top == NULL){

cout<<"Stack is empty"<<endl;

return -1;

}

else{

return Top->data;

}

}

void pop(){

if(Top == NULL){

cout<<"Stack is empty"<<endl;

return;

}

StackNode \*temp = Top;

Top = Top->next;

delete temp;

size--;

}

int Size(){

return size;

}

bool isEmpty(){

if(Top==NULL){

return true;

}

else{

return false;

}

}

void print(){

StackNode \*current = Top;

while(current != NULL){

cout<<current->data<<" ";

current = current->next;

}

cout<<endl;

}

};

int main(){

Stack s;

// cout<<s.top()<<endl;

// s.pop();

s.push(4);

s.push(5);

s.push(6);

s.print();

s.pop();

s.print();

}

Advantages:

The linked list implementation of a stack can grow and shrink according to the needs at runtime.

Disadvantages:

Requires extra memory due to the involvement of pointers.

QUEUE:

A Queue is defined as a linear data structure that is open at both ends and the operations are performed in First In First Out (FIFO) order.

Queue using Array

#include<bits/stdc++.h>

using namespace std;

class Queue{

public:

int \*arr;

int cursize,rear,front,size;

Queue(int size){

cursize = 0;

rear = 0;

front = 0;

this->size = size;

arr = new int[size];

}

void push(int val){

if(cursize == size){

cout<<"Queue is Full"<<endl;

return;

}

arr[rear%size] = val;

rear++;

cursize++;

}

int Front(){

if(cursize == 0){

cout<<"Queue is empty"<<endl;

return -1;

}

return arr[front%size];

}

void pop(){

if(cursize == 0){

cout<<"Queue is empty"<<endl;

return;

}

cursize--;

arr[front%size] = -1;

front++;

}

int Size(){

return cursize;

}

bool isEmpty(){

if(cursize==0){

return true;

}

else{

return false;

}

}

void print(){

for(int i = front; i<rear; i++){

cout<<arr[i%size]<<" ";

}

cout<<endl;

}

};

int main(){

Queue q(3);

q.Front();

q.pop();

q.push(4);

q.push(5);

cout<<q.Front()<<endl;

}

Advantage:

Easy to implement

Disadvantage:

Static Data Structure, fixed size.

Queue using LinkedList

#include<bits/stdc++.h>

using namespace std;

class QueueNode{

public:

int val;

QueueNode \*next;

QueueNode(int data){

val = data;

next = NULL;

}

};

class Queue{

public:

QueueNode\* front, \*rear;

int size;

Queue(){

rear = NULL;

front = NULL;

size = 0;

}

void push(int val){

QueueNode\* temp = new QueueNode(val);

if(front == NULL){

front = temp;

rear = temp;

}

else{

rear->next = temp;

rear = temp;

}

size++;

}

int Front(){

if(front == NULL){

cout<<"Queue is empty"<<endl;

return -1;

}

return front->val;

}

void pop(){

if(front == NULL){

cout<<"Queue is empty"<<endl;

return;

}

QueueNode\* temp = front;

front = front->next;

delete temp;

size--;

}

int Size(){

return size;

}

bool isEmpty(){

return front == NULL;

}

void print(){

QueueNode\* current = front;

while(current != NULL){

cout<<current->val<<" ";

current = current->next;

}

cout<<endl;

}

};

int main(){

Queue q;

q.Front();

q.pop();

q.push(4);

q.push(5);

cout<<q.Front()<<endl;

q.print();

q.pop();

q.print();

cout<<q.Front()<<endl;

q.push(6);

q.print();

cout<<q.Size()<<endl;

}

-------------------------------------------------------------------------------------------------------------------------

Stack using 2 queues

// by making push operation costly

class MyStack {

public:

queue<int>q1,q2;

MyStack() {

}

void push(int x) {

q2.push(x);

while(!q1.empty()){

q2.push(q1.front());

q1.pop();

}

swap(q1,q2);

}

int pop() {

int top = q1.front();

q1.pop();

return top;

}

int top() {

return q1.front();

}

bool empty() {

return q1.size()==0;

}

// TC - O(N)

// SC - O(2\*N)

};

// by making pop operation costly

One by one dequeue everything except the last element from q1 and enqueue to q2.

Dequeue the last item of q1, the dequeued item is the result, store it.

Swap the names of q1 and q2

Return the item stored in step 2.

void pop()

{

if (q1.empty())

return;

// Leave one element in q1 and

// push others in q2.

while (q1.size() != 1) {

q2.push(q1.front());

q1.pop();

}

// Pop the only left element

// from q1

q1.pop();

// swap the names of two queues

queue<int> q = q1;

q1 = q2;

q2 = q;

}

void push(int x) { q1.push(x); }

-------------------------------------------------------------------------------------------------------------------------

Stack using 1 queue

class MyStack {

public:

queue<int>q;

MyStack() {

}

void push(int x) {

q.push(x);

for(int i=0; i<q.size()-1; i++){

q.push(q.front());

q.pop();

}

}

int pop() {

int top = q.front();

q.pop();

return top;

}

int top() {

return q.front();

}

bool empty() {

return q.size()==0;

}

// TC - O(N)

// SC - O(N)

};

-------------------------------------------------------------------------------------------------------------------------

Queue using stacks

approach 1:

stack<int>st1,st2;

MyQueue() {

}

void push(int x) {

while(!st1.empty()){

st2.push(st1.top());

st1.pop();

}

st1.push(x);

while(!st2.empty()){

st1.push(st2.top());

st2.pop();

}

}

int pop() {

int Top = st1.top();

st1.pop();

return Top;

}

int peek() {

return st1.top();

}

bool empty() {

return st1.empty();

}

// TC - push-O(N)

// SC - O(2N)

approach 2:

stack<int>st1,st2;

MyQueue() {

}

void push(int x) {

st1.push(x);

}

int pop() {

if(st2.size()){

int Top = st2.top();

st2.pop();

return Top;

}

else{

while(!st1.empty()){

st2.push(st1.top());

st1.pop();

}

int Top = st2.top();

st2.pop();

return Top;

}

}

int peek() {

if(st2.size()){

int Top = st2.top();

return Top;

}

else{

while(st1.size()){

st2.push(st1.top());

st1.pop();

}

return st2.top();

}

}

bool empty() {

return (st1.size()+st2.size())==0;

}

// TC - push-O(1)(amotized)

// SC - O(2N)

approach 3: using only one stack

stack<int>st;

MyQueue() {

}

void push(int x) {

PushHelper(x);

}

void PushHelper(int x){

if(st.size() == 0){

st.push(x);

return;

}

int data = st.top();

st.pop();

PushHelper(x);

st.push(data);

return;

}

int pop() {

int Top = st.top();

st.pop();

return Top;

}

int peek() {

return st.top();

}

bool empty() {

return st.size()==0;

}

// TC - pop(O(N))

// SC - O(N)

-------------------------------------------------------------------------------------------------------------------------

Design min stack

approach 1: using 2 stacks

stack<ll>st1,st2;

MinStack() {

}

void push(ll val) {

if(st1.empty()){

st1.push(val);

st2.push(val);

}

else{

st1.push(val);

st2.push(min(val,st2.top()));

}

}

void pop() {

st1.pop();

st2.pop();

}

int top() {

return st1.top();

}

int getMin() {

return st2.top();

}

// TC - O(1)

// SC - O(2N)

approach2: using pair

stack<pair<ll,ll>>st;

MinStack() {

}

void push(ll val) {

if(st.empty()){

st.push({val,val});

}

else{

st.push({val,min(val,st.top().second)});

}

}

void pop() {

st.pop();

}

int top() {

return st.top().first;

}

int getMin() {

return st.top().second;

}

// TC - O(1)

// SC - O(2N)

approach 3: using only one stack

stack<ll>st;

ll mini;

MinStack() {

mini = 1e10;

}

void push(ll val) {

if(st.empty()){

mini = val;

st.push(val);

}

else{

if(val >= mini){

st.push(val);

}

else{

st.push(2\*val\*1LL-mini);

mini = min(val,mini);

}

}

}

void pop() {

ll Top = st.top();

st.pop();

if(Top < mini){

mini = 2\*mini - Top;

}

}

int top() {

if(st.top() < mini){

return mini;

}

else{

return st.top();

}

}

int getMin() {

return mini;

}

// TC - O(1)

// SC - O(N)

/\* pop intution

st.top() < mini

modified < mini

what we were pushing?

val < mini (-3<-2)

val - mini < 0

val + val - mini < val

2\*val - mini < val

modified < val

what was that val that val was mini

modified < val

hence proved

mini modification while poping intution

mini = (2\*mini-st.top())

mini = (2\*mini-(2\*val-prev\_min))

val is current mini only, i.e. mini

mini = prev\_min

\*/

-------------------------------------------------------------------------------------------------------------------------

stack in two arrays

#include <bits/stdc++.h>

using namespace std;

class TwoStack {

public:

// Initialize TwoStack.

int top1;

int top2;

int \*arr;

int size;

TwoStack(int s) {

// Write your code here.

size = s;

arr = new int[size];

top1 = (size/2) + 1;

top2 = size/2;

}

// Push in stack 1.

void push1(int num) {

if(top1 > 0){

top1--;

arr[top1] = num;

}

else{

// cout<<"Stack Overflow"<<endl;

return;

}

}

// Push in stack 2.

void push2(int num) {

if(top2 < size - 1){

top2++;

arr[top2] = num;

}

else{

// cout<<"Stack Overflow"<<endl;

return;

}

}

// Pop from stack 1 and return popped element.

int pop1() {

if(top1 <= (size/2)){

int top = arr[top1];

top1++;

return top;

}

else{

// cout<<"Stack Underflow"<<endl;

return -1;

}

}

// Pop from stack 2 and return popped element.

int pop2() {

if(top2 >= (size/2) +1 ){

int top = arr[top2];

top2--;

return top;

}

else{

// cout<<"Stack Underflow"<<endl;

return -1;

}

}

};

--> 2nd approach

space utilization

#include <bits/stdc++.h>

using namespace std;

class TwoStack {

public:

// Initialize TwoStack.

int top1;

int top2;

int \*arr;

int size;

TwoStack(int s) {

// Write your code here.

size = s;

arr = new int[size];

top1 = -1;

top2 = size;

}

// Push in stack 1.

void push1(int num) {

if(top2 - top1 > 1){

top1++;

arr[top1] = num;

}

else{

// cout<<"Stack Overflow"<<endl;

return;

}

}

// Push in stack 2.

void push2(int num) {

if(top2 - top1 > 1){

top2--;

arr[top2] = num;

}

else{

// cout<<"Stack Overflow"<<endl;

return;

}

}

// Pop from stack 1 and return popped element.

int pop1() {

if(top1 >= 0){

int top = arr[top1];

top1--;

return top;

}

else{

// cout<<"Stack Underflow"<<endl;

return -1;

}

}

// Pop from stack 2 and return popped element.

int pop2() {

if(top2 < size){

int top = arr[top2];

top2++;

return top;

}

else{

// cout<<"Stack Underflow"<<endl;

return -1;

}

}

};

-------------------------------------------------------------------------------------------------------------------------

N stacks in an array

brute force : partition array into n/k parts

optimised:

#include <bits/stdc++.h>

class NStack

{

int \*arr, \*top, \*next;

int n, s, freespot;

public:

// Initialize your data structure.

NStack(int N, int S)

{

n = N;

s = S;

freespot = 0;

arr = new int[s];

top = new int[n];

next = new int[s];

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

top[i] = -1;

}

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

next[i] = i+1;

}

next[s-1] = -1;

}

// Pushes 'X' into the Mth stack. Returns true if it gets pushed into the stack, and false otherwise.

bool push(int x, int m)

{

if(freespot == -1){

return false;

}

else{

// find index

int index = freespot;

// update freespot

freespot = next[index];

// insert element into array

arr[index] = x;

// update next

next[index] = top[m-1];

// update top

top[m-1] = index;

return true;

}

}

// Pops top element from Mth Stack. Returns -1 if the stack is empty, otherwise returns the popped element.

int pop(int m)

{

if(top[m-1]==-1){

return -1;

}

else{

int index = top[m-1];

top[m-1] = next[index];

next[index] = freespot;

freespot = index;

return arr[index];

}

}

// TC - O(1)

// SC - O(N+S)

};

-------------------------------------------------------------------------------------------------------------------------

Delete middle element of a stack

void solve(stack<int>&s, int count, int size){

if(count == size/2){

s.pop();

return;

}

int num = s.top();

s.pop();

solve(s,count+1,size);

s.push(num);

}

void deleteMid(stack<int>&s, int size)

{

solve(s,0,size);

}

// TC - O(N)

// SC - O(1)

-------------------------------------------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------------------------------------------

Insert An Element At Its Bottom In A Given Stack

void solve(stack<int>&s, int x){

if(s.size() == 0){

s.push(x);

return;

}

int num = s.top();

s.pop();

solve(s,x);

s.push(num);

}

stack<int> pushAtBottom(stack<int>&s, int x)

{

solve(s,x);

return s;

}

// TC - O(N)

// SC - O(1)

-------------------------------------------------------------------------------------------------------------------------

Reverse Stack Using Recursion

void insertAtBottom(stack<int>&s, int num){

if(s.size() == 0 ){

s.push(num);

return;

}

int x = s.top();

s.pop();

insertAtBottom(s,num);

s.push(x);

}

void reverseStack(stack<int> &s) {

if(s.size()==0){

return;

}

int num = s.top();

s.pop();

reverseStack(s);

insertAtBottom(s,num);

}

// TC - O(N^2)

// SC - O(1)

-------------------------------------------------------------------------------------------------------------------------

Sort a Stack

void sortt(stack<int>&s, int num){

if(s.size() == 0 || s.top()<num){

s.push(num);

return;

}

int x = s.top();

s.pop();

sortt(s,num);

s.push(x);

}

void sortStack(stack<int> &s)

{

if(s.size() == 0){

return;

}

int num = s.top();

s.pop();

sortStack(s);

sortt(s,num);

}

// TC - O(N^2)

// SC - O(1)

-------------------------------------------------------------------------------------------------------------------------

Redundant Brackets

https://www.codingninjas.com/studio/problems/redundant-brackets\_975473?leftPanelTab=2

bool findRedundantBrackets(string &s)

{

stack<char>st;

int n = s.size();

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

if(s[i] == '(' || s[i] == '+' || s[i] == '-' || s[i] == '/' || s[i] == '\*'){

st.push(s[i]);

}

else{

if(s[i] == ')'){

bool rebundant = true;

while(st.top() != '('){

if(st.top() == '+' || st.top() == '-' || st.top() == '/' || st.top() == '\*'){

rebundant = false;

}

st.pop();

}

if(rebundant){

return true;

}

st.pop();

}

}

}

return false;

}

// TC - O(2\*N)

// SC - O(N)

-------------------------------------------------------------------------------------------------------------------------

Minimum Cost To Make String Valid

<https://www.codingninjas.com/codestudio/problems/minimum-cost-to-make-string-valid_1115770?leftPanelTab=0>

Brute force : generate all combos of bracket then chek 2^n

opt:

int findMinimumCost(string &s) {

int n = s.size();

if(n%2){

return -1;

}

stack<char>st;

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

if(s[i] == '{'){

st.push(s[i]);

}

else{

if(!st.empty() && st.top()=='{'){

st.pop();

}

else{

st.push(s[i]);

}

}

}

int a = 0, b = 0;

while(!st.empty()){

if(st.top()=='{'){

b++;

}

else{

a++;

}

st.pop();

}

return (a+1)/2 + (b+1)/2;

}

-------------------------------------------------------------------------------------------------------------------------

Next Greater Element

<https://practice.geeksforgeeks.org/problems/next-larger-element-1587115620/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article>

vector<long long> nextLargerElement(vector<long long> arr, int n){

stack<long long>st;

vector<long long> ans(n);

for(int i=n-1; i>=0; i--){

if(st.empty()){

st.push(arr[i]);

ans[i]=-1;

}

else{

while(!st.empty() && st.top()<=arr[i]){

st.pop();

}

if(st.empty()){

ans[i]=-1;

}

else{

ans[i] = st.top();

}

st.push(arr[i]);

}

}

return ans;

}

// TC - O(2\*N)

// SC - O(N)

-------------------------------------------------------------------------------------------------------------------------

Next Greater Element I

<https://leetcode.com/problems/next-greater-element-i/description/>

vector<int> nextGreaterElement(vector<int>& nums1, vector<int>& nums2) {

int m = nums1.size();

int n = nums2.size();

stack<int>st;

vector<int> ans(n);

unordered\_map<int, int>mp;

for(int i=n-1; i>=0; i--){

if(st.empty()){

st.push(nums2[i]);

mp[nums2[i]]=-1;

}

else{

while(!st.empty() && st.top()<=nums2[i]){

st.pop();

}

if(st.empty()){

mp[nums2[i]]=-1;

}

else{

mp[nums2[i]] = st.top();

}

st.push(nums2[i]);

}

}

vector<int>res(m);

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

res[i] = mp[nums1[i]];

}

return res;

}

// TC - O(N)

// SC - O(N)

// TC - O(2\*N)

// SC - O(N)

-------------------------------------------------------------------------------------------------------------------------

Next Greater Element II

<https://leetcode.com/problems/next-greater-element-ii/description/>

vector<int> nextGreaterElements(vector<int>& nums) {

int n = nums.size();

stack<int>st;

vector<int> ans(n);

for(int i=2\*n-1; i>=0; i--){

if(i<n){

while(!st.empty() && st.top()<=nums[i%n]){

st.pop();

}

if(st.empty()){

ans[i]=-1;

}

else{

ans[i] = st.top();

}

}

st.push(nums[i%n]);

}

return ans;

}

// TC - O(2\*N)

// SC - O(N)

-------------------------------------------------------------------------------------------------------------------------

Next Smallest Element

vector<int> help\_classmate(vector<int> arr, int n)

{

stack<int>st;

vector<int> ans(n);

for(int i=n-1; i>=0; i--){

if(st.empty()){

st.push(arr[i]);

ans[i]=-1;

}

else{

while(!st.empty() && st.top()>=arr[i]){// just change < sign to > sign from NGE

st.pop();

}

if(st.empty()){

ans[i]=-1;

}

else{

ans[i] = st.top();

}

st.push(arr[i]);

}

}

return ans;

}

-------------------------------------------------------------------------------------------------------------------------

Online Stock Span

<https://leetcode.com/problems/online-stock-span/description/>

stack<int>st;

vector<int>nums;

int ind = 0;

int next(int price) {

nums.push\_back(price);

int ans = 0;

while(st.size() && nums[st.top()]<=price){

st.pop();

}

if(st.empty()){

ans = ind+1;

}

else{

ans = (ind - st.top());

}

st.push(ind);

ind++;

return ans;

}

// TC - O(2\*N)

// SC - O(N)

-------------------------------------------------------------------------------------------------------------------------

Maximal Rectangle

<https://leetcode.com/problems/maximal-rectangle/description/>

int largestRectangleArea(vector<int>& heights) {

int n = heights.size();

int leftsmall[n],rightsmall[n];

stack<int>st;

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

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

st.pop();

}

if(st.empty()){

leftsmall[i] = 0;

}

else{

leftsmall[i] = st.top()+1;

}

st.push(i);

}

while(st.size()){

st.pop();

}

for(int i=n-1; i>=0; i--){

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

st.pop();

}

if(st.empty()){

rightsmall[i] = n-1;

}

else{

rightsmall[i] = st.top()-1;

}

st.push(i);

}

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

// cout<<leftsmall[i]<<" ";

// }

// cout<<endl;

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

// cout<<rightsmall[i]<<" ";

// }

// cout<<endl;

int maxA = 0;

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

maxA = max(maxA,(rightsmall[i]-leftsmall[i]+1)\*heights[i]);

}

return maxA;

}

int maximalRectangle(vector<vector<char>>& matrix) {

int n = matrix.size();

int m = matrix[0].size();

int maxA = 0;

vector<int>heights(m,0);

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

for(int j=0; j<m; j++){

if(i==0){

if(matrix[i][j]=='1'){

heights[j] = 1;

}

}

else{

if(matrix[i][j]=='1'){

heights[j]++;

}

else{

heights[j] = 0;

}

}

}

maxA = max(maxA,largestRectangleArea(heights));

}

return maxA;

}

// TC - O(N\*M)

// SC - O(N)

-------------------------------------------------------------------------------------------------------------------------

<https://practice.geeksforgeeks.org/problems/the-celebrity-problem/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article>

The Celebrity Problem

int celebrity(vector<vector<int> >& m, int n)

{

stack<int>st;

// push all elements into stack

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

st.push(i);

}

// get 2 elements and compare them

while(st.size()>1){

int a = st.top();

st.pop();

int b = st.top();

st.pop();

if(m[a][b]==1){

st.push(b);

}

else{

st.push(a);

}

}

int cel = st.top();

int ct = 0;

// col check

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

if(m[cel][i]==0){

ct++;

}

}

if(ct != n){

return -1;

}

ct = 0;

// row check

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

if(m[i][cel]==1){

ct++;

}

}

if(ct != n-1){

return -1;

}

return cel;

}

// TC - O(N)

// SC - O(N)

-------------------------------------------------------------------------------------------------------------------------

<https://leetcode.com/problems/longest-valid-parentheses/description/>

Longest Valid Parentheses

int longestValidParentheses(string s) {

int n = s.size();

stack<int>st;

int ans = 0;

st.push(-1);

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

int t = st.top();

if(t!=-1 && s[t]=='(' && s[i]==')'){

st.pop();

ans = max(ans,i-st.top());

}

else{

st.push(i);

}

}

return ans;

}

// TC - O(N)

// SC - O(N)

-------------------------------------------------------------------------------------------------------------------------

Evaluation of Postfix Expression

<https://practice.geeksforgeeks.org/problems/evaluation-of-postfix-expression1735/1>

int evaluatePostfix(string s)

{

stack<int>st;

int n = s.size();

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

if(s[i]>='0' && s[i]<='9'){

st.push(s[i]-'0');

}

else{

int a = st.top();

st.pop();

int b = st.top();

st.pop();

if(s[i]=='+'){

st.push(b+a);

}

else if(s[i]=='-'){

st.push(b-a);

}

else if(s[i]=='\*'){

st.push(b\*a);

}

else{

st.push(b/a);

}

}

}

return st.top();

}

-------------------------------------------------------------------------------------------------------------------------

<https://practice.geeksforgeeks.org/problems/stack-permutations/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article>

Stack Permutations

int isStackPermutation(int n,vector<int> &a,vector<int> &b){

stack<int>st;

int j=0;

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

st.push(a[i]);

while(!st.empty() && st.top()==b[j]){

j++;

st.pop();

}

}

if(st.empty()){

return true;

}

return false;

}

-------------------------------------------------------------------------------------------------------------------------

Implement deque using array

class Deque

{

public:

int \*arr;

int front, rear, size;

Deque(int n)

{

size = n;

arr = new int[size];

front = -1;

rear = -1;

}

// Pushes 'X' in the front of the deque. Returns true if it gets pushed into the deque, and false otherwise.

bool pushFront(int x)

{

if(isFull()){

return false;

}

else if(isEmpty()){

front = rear = 0;

}

else if(front==0){

front = size - 1;

}

else{

front--;

}

arr[front] = x;

return true;

}

// Pushes 'X' in the back of the deque. Returns true if it gets pushed into the deque, and false otherwise.

bool pushRear(int x)

{

if(isFull()){

return false;

}

else if(isEmpty()){

front = rear = 0;

}

else if(rear==size-1){

rear = 0;

}

else{

rear++;

}

arr[rear] = x;

return true;

}

// Pops an element from the front of the deque. Returns -1 if the deque is empty, otherwise returns the popped element.

int popFront()

{

if(isEmpty()){

return -1;

}

int ans = arr[front];

arr[front] = -1;

if(front == rear){

front = rear = -1;

}

else if(front==size-1){

front = 0;

}

else{

front++;

}

return ans;

}

// Pops an element from the back of the deque. Returns -1 if the deque is empty, otherwise returns the popped element.

int popRear()

{

if(isEmpty()){

return -1;

}

int ans = arr[rear];

arr[rear] = -1;

if(front == rear){

front = rear = -1;

}

else if(rear==0){

rear = size-1;

}

else{

rear--;

}

return ans;

}

// Returns the first element of the deque. If the deque is empty, it returns -1.

int getFront()

{

if(isEmpty()){

return -1;

}

return arr[front];

}

// Returns the last element of the deque. If the deque is empty, it returns -1.

int getRear()

{

if(isEmpty()){

return -1;

}

return arr[rear];

}

// Returns true if the deque is empty. Otherwise returns false.

bool isEmpty()

{

if(front == -1){

return true;

}

else{

return false;

}

}

// Returns true if the deque is full. Otherwise returns false.

bool isFull()

{

if((front==0 && rear==size-1) || (front!=0 && rear==(front-1)%(size-1))){

return true;

}

return false;

}

};

-------------------------------------------------------------------------------------------------------------------------

Reverse Queue

<https://practice.geeksforgeeks.org/problems/queue-reversal/1>

// brute force : using stack

// opt : recursion

void reverseq(queue<int>&q){

if(q.size() == 0){

return;

}

int num = q.front();

q.pop();

reverseq(q);

q.push(num);

}

queue<int> rev(queue<int> q)

{

reverseq(q);

return q;

}

// TC - O(N)

// SC - O(1)

-------------------------------------------------------------------------------------------------------------------------

<https://practice.geeksforgeeks.org/problems/first-negative-integer-in-every-window-of-size-k3345/1>

**First negative integer in every window of size k**

vector<long long> printFirstNegativeInteger(long long int a[],

long long int n, long long int k) {

deque<int>dq;

vector<long long >ans;

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

if(dq.size() && dq.front() == i-k){

dq.pop\_front();

}

if(a[i] < 0){

dq.push\_back(i);

}

if(i>=k-1){

if(dq.size()==0){

ans.push\_back(0);

}

else{

ans.push\_back(a[dq.front()]);

}

}

}

return ans;

// TC - O(n)

// SC - O(k)

}

-------------------------------------------------------------------------------------------------------------------------

https://practice.geeksforgeeks.org/problems/reverse-first-k-elements-of-queue/1

**Reverse First K elements of Queue**

void reversek(queue<int>&q, int k){

if(k == 0){

return;

}

int num = q.front();

q.pop();

reversek(q,k-1);

q.push(num);

}

queue<int> modifyQueue(queue<int> q, int k) {

reversek(q,k);

int rem = q.size() - k;

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

q.push(q.front());

q.pop();

}

return q;

}

// TC - O(N)

// Sc - O(1)

-------------------------------------------------------------------------------------------------------------------------

<https://practice.geeksforgeeks.org/problems/first-non-repeating-character-in-a-stream1216/1>

**First non-repeating character in a stream**

string FirstNonRepeating(string a){

int n = a.size();

queue<char>q;

string ans;

int vis[26] = {0};

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

q.push(a[i]);

vis[a[i]-'a']++;

while(q.size() && vis[q.front()-'a']>1){

q.pop();

}

if(q.size()==0){

ans.push\_back('#');

}

else{

ans.push\_back(q.front());

}

}

return ans;

}

// TC - O(N)

// SC - O(26\*2);

-------------------------------------------------------------------------------------------------------------------------

**Circular tour**

**Brute : try from all starting point O(N^2)**

[**https://practice.geeksforgeeks.org/problems/circular-tour-1587115620/1**](https://practice.geeksforgeeks.org/problems/circular-tour-1587115620/1)

int tour(petrolPump p[],int n)

{

int deficient = 0;

int balance = 0;

int start = 0;

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

balance += (p[i].petrol-p[i].distance);

if(balance < 0){

deficient += balance;

balance = 0;

start = i+1;

}

}

if(balance + deficient >=0){

return start;

}

return -1;

}

// TC - O(N)

// SC - O(1)

-------------------------------------------------------------------------------------------------------------------------

<https://practice.geeksforgeeks.org/problems/interleave-the-first-half-of-the-queue-with-second-half/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article>

**Interleave the First Half of the Queue with Second Half**

vector<int> rearrangeQueue(queue<int> &q){

int n = q.size();

vector<int>ans(n);

for(int i=0; i<n; i+=2){

ans[i] = q.front();

q.pop();

}

for(int i=1; i<n; i+=2){

ans[i] = q.front();

q.pop();

}

return ans;

}

// TC - O(N)

// SC - O(1)

-------------------------------------------------------------------------------------------------------------------------

<https://leetcode.com/problems/remove-duplicate-letters/submissions/985026582/>