1. Implement stack using array

//Function to push an integer into the stack.

void MyStack :: push(int x)

{

top++;

arr[top] = x;

}

//Function to remove an item from top of the stack.

int MyStack :: pop()

{

if(top==-1) return -1;

else

{

top--;

return arr[top+1];

}

}

1. Implement stack using linked list

//Function to push an integer into the stack.

void MyStack ::push(int x)

{

StackNode \*temp = new StackNode(x);

temp->next = top;

top = temp;

}

//Function to remove an item from top of the stack.

int MyStack ::pop()

{

if(top==NULL) return -1;

StackNode \*temp = top;

top = top->next;

return temp->data;

}

1. Operations on stack

//Function to push an element into the stack.

void insert(stack<int> &s,int x)

{

s.push(x);

}

//Function to remove top element from stack.

void remove(stack<int> &s)

{

s.pop();

}

//Function to print the top element of stack.

void headOf\_Stack(stack<int> &s)

{

int x=s.top();

cout<<x<<" "<<endl;

}

//Function to search an element in the stack.

bool find(stack<int> s, int val)

{

bool exists=false;

stack<int> temp;

while(s.empty()!=true)

{

int ans = s.top();

s.pop();

temp.push(ans);

if(ans==val)

{

exists = true;

break;

}

}

while(temp.empty()!=true)

{

s.push(temp.top());

temp.pop();

}

if(exists==true){

return true;

}

else{

return false;

}

}

1. Removing consecutive duplicates

class Solution

{

public:

//Function to remove consecutive duplicates from given string using Stack.

string removeConsecutiveDuplicates(string s)

{

stack<char> c;

for(int i=s.length()-1; i>=0; i--)

c.push(s[i]);

char val = c.top();

c.pop();

string str = "";

str = str+val;

while(c.empty()!=true)

{

char temp = c.top();

c.pop();

if(val!=temp)

str = str+temp;

val = temp;

}

return str;

}

};

1. Removing consecutive duplicates – 2

string removePair(string str)

{

stack<char> f,b;

for(int i=str.length()-1; i>=0; i--)

{

b.push(str[i]);

}

f.push(b.top());

b.pop();

while(b.empty()!=true)

{

if(f.top()==b.top())

{

f.pop();

b.pop();

if(f.empty())

{

if(b.empty())

break;

else

{

f.push(b.top());

b.pop();

}

}

}

else

{

f.push(b.top());

b.pop();

}

}

string s = "";

while(!f.empty())

{

s = f.top() + s;

f.pop();

}

while(!b.empty())

{

s = s + b.top();

b.pop();

}

return s;

}

1. Parenthesis checker

char op(char in)

{

if(in=='(') return ')';

if(in=='{') return '}';

if(in=='[') return ']';

if(in==')') return '(';

if(in=='}') return '{';

if(in==']') return '[';

}

bool ispar(string x)

{

stack<char> ch;

for(int i = 0; i<x.length(); i++)

{

char c = x[i];

if(c=='(' || c=='{' || c=='[')

ch.push(c);

else

{

if(!ch.empty() && c==op(ch.top()))

ch.pop();

else

return false;

}

}

if(!ch.empty()) return false;

return true;

}

1. Implement two stacks in an array

//Function to push an integer into the stack1.

void twoStacks :: push1(int x)

{

top1++;

arr[top1] = x;

}

//Function to push an integer into the stack2.

void twoStacks ::push2(int x)

{

top2--;

arr[top2] = x;

}

//Function to remove an element from top of the stack1.

int twoStacks ::pop1()

{

if(top1==-1) return -1;

top1--;

return arr[top1+1];

}

//Function to remove an element from top of the stack2.

int twoStacks :: pop2()

{

if(top2==size) return -1;

top2++;

return arr[top2-1];

}

1. Get min at pop

//Function to push all the elements into the stack.

stack<int> \_push(int arr[],int n)

{

stack<int> val;

allmin.push(arr[0]);

val.push(arr[0]);

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

{

val.push(arr[i]);

if(arr[i]<=allmin.top())

allmin.push(arr[i]);

}

return val;

}

//Function to print minimum value in stack each time while popping.

void \_getMinAtPop(stack<int>s)

{

while(!s.empty())

{

cout<<allmin.top()<<" ";

if(s.top()==allmin.top())

allmin.pop();

s.pop();

}

}

1. Delete middle element of stack

class Solution

{

public:

//Function to delete middle element of a stack.

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

{

stack<int> cpy;

for(int i=0; i<sizeOfStack/2;i++)

{

cpy.push(s.top());

s.pop();

}

s.pop();

while(!cpy.empty())

{

s.push(cpy.top());

cpy.pop();

}

}

};

1. Test
2. test
3. Stock span problem

vector <int> calculateSpan(int price[], int n)

{

vector<int> ans;

stack<int> g;

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

{

if(g.empty())

{

ans.push\_back(i+1);

g.push(i);

}

else if(price[i]>=price[g.top()])

{

g.pop();

i--;

}

else

{

ans.push\_back(i-g.top());

g.push(i);

}

}

return ans;

}

1. next greater element

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

vector<long long> ans;

stack<long long> g;

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

{

if(g.empty())

{

g.push(arr[i]);

ans.push\_back(-1);

}

else if(arr[i]>=g.top())

{

g.pop();

i++;

}

else

{

ans.push\_back(g.top());

g.push(arr[i]);

}

}

reverse(ans.begin(), ans.end());

return ans;

}

1. max rectangular area in a histogram

long long getMaxArea(long long arr[], int n)

{

vector<int> fans;

vector<int> bans;

stack<int> f;

stack<int> b;

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

{

if(f.empty())

{

fans.push\_back(i+1);

f.push(i);

}

else if(arr[i]<=

arr[f.top()])

{

f.pop();

i--;

}

else

{

fans.push\_back(i-f.top());

f.push(i);

}

}

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

{

if(b.empty())

{

bans.push\_back(n-i);

b.push(i);

}

else if(arr[i]<=arr[b.top()])

{

b.pop();

i++;

}

else

{

bans.push\_back(b.top()-i);

b.push(i);

}

}

reverse(bans.begin(), bans.end());

long long area = 0;

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

area = max(area, abs(fans[i]+bans[i]-1)\*arr[i]);

return area;

}

1. the celebrity problem

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

{

int arrsumR[n] = {0};

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

{

for(int j=0; j<n; j++)

{

arrsumR[j]+=M[i][j];

}

}

int arrsumL[n] = {0};

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

{

for(int j=0; j<n; j++)

{

arrsumL[i]+=M[i][j];

}

}

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

{

int sumR=0, sumL=0;

for(int j = 0; j<n; j++)

{

sumR+=M[i][j];

sumL+=M[j][i];

}

if(sumR==0 && sumL==n-1)

return i;

}

return -1;

}

1. maximum of minimum for every window size

vector <int> maxOfMin(int arr[], int n)

{

vector<int> ans(n);

int lft[n];

int rt[n];

stack<int> st;;

lft[0]=-1;

rt[n-1]=n;

st.push(0);

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

while(!st.empty()&&arr[st.top()]>=arr[i])st.pop();

if(st.empty())

lft[i]=-1;

else{

lft[i]=st.top();

}

st.push(i);

}

while(!st.empty())

st.pop();

st.push(n-1);

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

while(!st.empty()&&arr[st.top()]>=arr[i])st.pop();

if(st.empty())

rt[i]=n;

else{

rt[i]=st.top();

}

st.push(i);

}

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

int len=rt[i]-lft[i]-1;

if(ans[len-1]<arr[i])

ans[len-1]=arr[i];

}

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

if(ans[i]<ans[i+1])

ans[i]=ans[i+1];

}

return ans;

}