Stack and Queue Worksheet

September 2024

# Stack

* 1. Given the head of a singly linked list, return true if it is a palindrome or false otherwise.

Input: head = [1,2,2,1] Output: true

Explanation: The linked list [1,2,2,1] reads the same forward and back- ward, so it is a palindrome.

bool isPalindrome(Node\* head){

Node\* slow = head;

Node\* fast = head;

//stack to store the first half of the list

Node\* stackTop = nullptr;

//push the first half onto the stack

while(fast != nullptr && fast->next != nullptr){

push(stackTop, slow->data);

slow = slow->next;

fast = fast->next->next;

}

//if the list has an odd number of elements, skip the middle one

if(fast != nullptr){

slow = slow->next;

}

while(slow != nullptr){

char topData = pop(stackTop);

if(topData != slow->data) {

return false;

}

slow = slow->next;

}

return true;

}

* 1. You are keeping the scores for a baseball game with strange rules. At the beginning of the game, you start with an empty record. You are given a list of strings operations, where operations[i] is the i-th operation you must apply to the record and is one of the following:
     + An integer x: Record a new score of x.
     + ’+’: Record a new score that is the sum of the previous two scores.
     + ’D’: Record a new score that is the double of the previous score.
     + ’C’: Invalidate the previous score, removing it from the record.

Return the sum of all the scores on the record after applying all the oper- ations.

Input: operations = [”5”,”2”,”C”,”D”,”+”] Output: 30

Explanation:

”5” - Add 5 to the record, record is now [5] ”2” - Add 2 to the record, record is now [5, 2]

”C” - Invalidate and remove the previous score, record is now [5] ”D” - Add 2 \* 5 = 10 to the record, record is now [5, 10]

”+” - Add 5 + 10 = 15 to the record, record is now [5, 10, 15]

The total sum is 5 + 10 + 15 = 30

int calculatePoints(vector<string>& operations){

vector<int> record;

for(string op : operations){

if(op == “C”){

//remove the last entry

if(!record.empty()){

record.pop();

}

} else if(op == “D”){

//double the previous score

if(!record.empty()){

record.push(2 \* record.top());

}

} else if(op == “+”){

//add the sum of the last two scores

int top = record.top(); //get the top score

record.pop() //remove the top to get the second top

int newTop = record.top(); //second top

record. push(top); //push back the original top score

record.push( top + newTop) //push the sum

} else {

//record a new integer score

record.push(stoi(op)); //convert string to int and push

}

}

//calculate the total sum of all scores

int total sum = 0;

while(!record.empty()){

totalSum += record.top();

record.pop;

}

return total sum;

}

* 1. You are given a string s. Your task is to remove all digits by doing this operation repeatedly:
     + Delete the first digit and the closest non-digit character to its left.

Return the resulting string after removing all digits. Input: s = ”ab1c2d3”

Output: ”abcd”

Explanation:

* Delete ’1’ and ’a’: ”b1c2d3”
* Delete ’2’ and ’b’: ”c2d3”
* Delete ’3’ and ’c’: ”d3”
* Delete ’3’ and ’d’: ””

The resulting string is ”abcd”.

string removeDigits(const string& s){

stack<char> charStack; //stack to keep track of non-digits

//result string to store the modified characters

string result = “”;

for(char ch : s){

if(isdigit(ch)){

//if the current char is a digit, pop closest non digit

if(!charStack.empty()){

charStack.pp();

}

} else {

//for non digit chars, push them to stack and add result

charStack.push(ch);

result += ch;

}

}

return result;

}

# Queue

* 1. Implement a last-in-first-out (LIFO) stack using only two queues. The implemented stack should support all the functions of a normal stack (push, top, pop, and empty).

Implement the MyStack class:

* + - void push(int x) Pushes element x to the top of the stack.
    - int pop() Removes the element on the top of the stack and returns it.
    - int top() Returns the element on the top of the stack.
    - boolean empty() Returns true if the stack is empty, false otherwise.

MyStack stack = new MyStack(); stack.push(1);

stack.push(2); stack.top(); // returns 2 stack.pop(); // returns 2

stack.empty(); // returns false

#include <iostream>

#include <queue>

using namespace std;

class MyStack{

private:

queue<int> queue1;

queue<int> queue2;

public:

MyStack(){};

void push(int x){

queue1.push(x);

}

int pop(){

//move all elements except the last one to queue 2

while(queue1.size() > 1){

queue2.push(queue1.front());

queue1.pop();

}

int topElement = queue1.front();

queue1.pop();

//swap queue1 and queue2

swap(queue1, queue2);

return topElement;

}

int top(){

//move all elements except the last one to queue2

while(queue1.size() > 1){

queue2.push(queue1.front());

queue1.pop();

}

//the lest element in queue1 is the top of the stack

int topElement = queue1.front();

//move the last element to queue2

queue2.push(queue1.front());

queue1.pop();

swap(queue1, queue2);

return topElement;

}

bool empty(){

return queue1.empty();

}

};

int main(){

MyStack stack;

stack.push(1);

stack.push(2);

cout << "Top of stack: " << stack.top() << endl;

cout << "Popped element: " << stack.pop() << endl;

cout << "Is empty: " << boolalpha << stack.empty() << endl;

}

Time complexities of function:

* Push – O(1)
* Pop – O(n)
* Top – O(n)
* Empty – O(1)
  1. Implement the RecentCounter class that counts the number of recent calls:
     + RecentCounter() Initializes the counter with zero recent calls.
     + int ping(int t) Adds a new call at time t, where t represents some time in milliseconds, and returns the number of calls that have happened in the past 3000 milliseconds (including the new call). Specifically, return the number of calls that have happened in the inclusive range [t - 3000, t]. It is guaranteed that every call to ping uses a strictly larger value of t than the previous call.

RecentCounter counter = new RecentCounter(); counter.ping(1); // returns 1

counter.ping(100); // returns 2 counter.ping(3001); // returns 3 counter.ping(3002); // returns 3

#include <queue>

#include <iostream>

using namespace std;

class RecentCounter{

private:

queue<int> calls;

public:

RecentCounter(){}

//adds a new call at time t, and returns the number of recent calls

int ping(int t){

//add the new call timestamp to the queue

calls.push(t);

//remove calls that are outside the 3000 ms window

while(!calls.empty() && calls.front() < (t - 3000)){

//the range is 1-3001 inclusive on both ends

calls.pop();

}

//the size of the queue is the number of recent calls

return calls.size();

}

};

int main(){

RecentCounter counter;

cout << "Number of calls after call of timestamp 1: " << counter.ping(1) << endl;

cout << "Number of calls after call of timestamp 100: " << counter.ping(100) << endl;

cout << "Number of calls after call of timestamp 3001: " << counter.ping(3001) << endl;

cout << "Number of calls after call of timestamp 200: " << counter.ping(200) << endl;

cout << "Number of calls after call of timestamp 3002: " << counter.ping(3002) << endl;

cout << "Number of calls after call of timestamp 500: " << counter.ping(500) << endl;

return 0;

}

* 1. Implement a function that checks whether a given string is a palindrome or not, ignoring spaces and punctuation. A palindrome is a word, phrase, number, or other sequence of characters that reads the same forward and backward. Use a deque (double-ended queue) to implement this solution efficiently.

Implement the Solution class:

* + - bool isPalindrome(string s) Returns true if the input string is a palindrome, false otherwise.

Solution sol = new Solution();

sol.isPalindrome(”A man, a plan, a canal: Panama”); // returns true sol.isPalindrome(”race a car”); // returns false

#include <deque>

#include <iostream>

#include <string>

#include <cctype>

using namespace std;

class Solution{

public:

bool isPalindrome(string s){

deque<char> charDeque;

//clean the string and push characters to deque

for(char c : s){

if(isalnum(c)){

charDeque.push\_back(tolower(c));

}

}

//compare characters from both ends

while(charDeque.size() > 1){

if(charDeque.front() != charDeque.back()){

return false;

}

charDeque.pop\_front();

charDeque.pop\_back();

}

return true;

}

};

int main(){

Solution solution;

cout << boolalpha;

//test cases

cout << "A man, a plan, a canal: Panama : " << solution.isPalindrome("A man, a plan, a canal: Panama") << endl;

cout << "race a car : " << solution.isPalindrome("race a car") << endl;

cout << "\" \" : " << solution.isPalindrome(" ") << endl;

cout << "Madam, I'm Adam : " << solution.isPalindrome("Madam, I'm Adam") << endl;

return 0;

}