春节7天练|Day2：栈、队列和递归

你好，我是王争。初二好！

为了帮你巩固所学，真正掌握数据结构和算法，我整理了数据结构和算法中，必知必会的30个代码实现，分7天发布出来，供你复习巩固所用。今天是第二篇。

和昨天一样，你可以花一点时间，来完成测验。测验完成后，你可以根据结果，回到相应章节，有针对性地进行复习。

关于栈、队列和递归的几个必知必会的代码实现

栈

- 用数组实现一个顺序栈
- 用链表实现一个链式栈
- 编程模拟实现一个浏览器的前进、后退功能

队列

- 用数组实现一个顺序队列
- 用链表实现一个链式队列
- 实现一个循环队列

递归

- 编程实现斐波那契数列求值f(n)=f(n-1)+f(n-2)
- 编程实现求阶乘n!
- 编程实现一组数据集合的全排列

对应的LeetCode练习题（@Smallfly 整理）

栈

- Valid Parentheses（有效的括号）

英文版：https://leetcode.com/problems/valid-parentheses/

中文版：https://leetcode-cn.com/problems/valid-parentheses/

- Longest Valid Parentheses（最长有效的括号）

英文版：https://leetcode.com/problems/longest-valid-parentheses/

中文版：https://leetcode-cn.com/problems/longest-valid-parentheses/

- Evaluate Reverse Polish Notatio（逆波兰表达式求值）

英文版：https://leetcode.com/problems/evaluate-reverse-polish-notation/

中文版：https://leetcode-cn.com/problems/evaluate-reverse-polish-notation/

队列

- Design Circular Deque（设计一个双端队列）

英文版：https://leetcode.com/problems/design-circular-deque/

中文版：https://leetcode-cn.com/problems/design-circular-deque/

- Sliding Window Maximum（滑动窗口最大值）

英文版：https://leetcode.com/problems/sliding-window-maximum/

中文版：https://leetcode-cn.com/problems/sliding-window-maximum/

递归

- Climbing Stairs（爬楼梯）

英文版：https://leetcode.com/problems/climbing-stairs/

中文版：https://leetcode-cn.com/problems/climbing-stairs/

昨天的第一篇，是关于数组和链表的，如果你错过了，点击文末的“上一篇”，即可进入测试。

祝你取得好成绩！明天见！

精选留言：

- ALAN 2019-02-08 14:14:37
  import java.util.Arrays;

```
/**
*
*Stack 1 solution
*/
public class StackArray {

public Object[] arr = new Object[10];
public int count;

public void push(Object ele) {
if (count == arr.length) { // expand size
arr = Arrays.copyOf(arr, arr.length * 2);
}
arr[count] = ele;
count++;
}

public Object pop() {
if (count == 0)
return null;
if (count < arr.length / 2) {
arr = Arrays.copyOf(arr, arr.length / 2);
}
return arr[--count];

}
}

/**
*
*Stack 2 solution
*/
```

```
class StackLinked {
Node head;
Node tail;

public void push(Object ele) {

if (head == null) {
head = new Node(ele);
tail = head;
} else {
Node node = new Node(ele);
tail.next = node;
node.prev = tail;
tail = node;
}
}

public Object pop() {
if (tail == null)
return null;
Node node = tail;
if (tail == head) {
head = null;
tail = null;
} else
tail = tail.prev;
return node;

}
}
class Node {
Node prev;
```

```
Node next;
Object value;

public Node(Object ele) {
value = ele;
}
} [1赞]
```

- 吴... 2019-02-06 15:29:19
  之前有个类似的题，走楼梯，装苹果，就是把苹果装入盘子，可以分为有一个盘子为空（递归），和全部装满没有空的情况，找出状态方程，递归就可以列出来了。我觉得最关键是要列出状态方程，之前老师类似于说的不需要关注特别细节，不要想把每一步都要想明白，快速排序与递归排序之类的算法，之前总是想把很细节的弄懂，却发现理解有困难。[1赞]

- 李皮皮皮皮皮 2019-02-05 21:22:16
  基础数据结构和算法是基石，灵活运用是解题的关键。栈，队列这些数据结构说到底就是给顺序表添加约束，更便于解决某一类问题。学习中培养算法的设计思想是非常关键的。而且思想是可以通用的。之前读《暗时间》一书，收获颇深。书中介绍之正推反推我在做程序题时竟出奇的好用。[1赞]

- 神盾局闹别扭 2019-02-10 15:40:28
  Valid parentheses c++实现

```
class Solution {

enum ParentheseStatus {
invalid = 0,
sameTypeofParenthese = 1,
differentTypeofParenthese = 2
};
char a[3] = {'(', '[', '{' };
char b[3] = {')', ']', '}' };

ParentheseStatus Checkparenthese(char strStartParenthese, char strEndParenthese) {
int idx = 0;
for (; idx < 3; idx++)
```

```
{
if (strStartParenthese == a[idx])
break;
}
if (idx == 3)
return invalid;
return (b[idx] == strEndParenthese)? sameTypeofParenthese: differentTypeofParenthese;
}
public:
bool isValid(string s) {

stack<char> st;
int len = s.length();
for (int idx = 0; idx < len; idx++)
{
if (!st.empty()) {
ParentheseStatus emRt = Checkparenthese(st.top(), s[idx]);
if (invalid == emRt)
return false;
if (sameTypeofParenthese == emRt) {
st.pop();
}
else
st.push(s[idx]);
}
else
st.push(s[idx]);
}

return (st.empty() ? true : false);

}
```

```
};
```

- yingyingqin 2019-02-09 22:59:52

  全排列 C++实现

```
void digui(vector<int> res, int i,vector<int> curres)
{
if (i == res.size())
{
for (auto ci : curres)
cout << ci << " ";
cout << endl;
return;
}

for (int k = i; k < res.size(); k++)
{
int temp = res[k];
res[k] = res[i];
res[i] = temp;
curres.push_back(res[i]);
digui(res, i + 1,curres);
curres.pop_back();
}
}

void quanpailie(vector<int> res)
{//全排列
vector<int> curres;
digui(res, 0, curres);
}
```

  循环队列 C++实现

```
class cyclequeue{
public:
cyclequeue(){
}
bool insert(int num){
if ((curend+1)%100 == curfirst)
{
cout << "the queue all used." << endl;
return false;
}
arrque[curend] = num;
curend = (curend + 1) % 100;
return true;
}

int deque()
{
if (curfirst == curend)
{
cout << "there is nothing in queue." << endl;
return -1;
}
else
{
int temp = arrque[curfirst];
curfirst = (curfirst + 1) % 100;
return temp;
}

}

private:
```

```
int arrque[100];//申请一个大小为100的数组
int curfirst = 0;//当前队列队头元素所在位置
int curend = 0;//当前队列队尾元素所在位置
};
```

- 神盾局闹别扭 2019-02-09 19:47:25

全排列实现：

```
void Dopermute(char *pstr, char *pBegin)
{
if (*pBegin == '\0')
printf("%s\n", pstr);

for (char *pCur = pBegin; *pCur != '\0'; pCur++)
{

char temp = *pBegin;
*pBegin = *pCur;
*pCur = temp;

Dopermute_v2(pstr, pBegin + 1);

temp = *pBegin;
*pBegin = *pCur;
*pCur = temp;

}
}
void Permute(char* pstr)
{
if (pstr == nullptr)
return;
Dopermute(pstr, pstr);
```

```
}
```

- molybdenum 2019-02-09 16:33:46
  老师新年好 这是我的作业

  https://blog.csdn.net/github_38313296/article/details/86819684

- 你看起来很好吃 2019-02-09 16:09:15
  爬楼梯python代码实现，需要使用散列表存储已经计算过的数字，这样可以降低时间复杂度，否则Leetcode会报超时错误：

```
class Solution:
def __init__(self):
self.buf = {1:1, 2:2}

def climbStairs(self, n: 'int') -> 'int':
if n in self.buf:
return self.buf[n]

res = self.climbStairs(n-1) + self.climbStairs(n-2)
self.buf[n] = res

return res
```

- 你看起来很好吃 2019-02-09 15:39:52
  设计双端队列python代码：

```
class MyCircularDeque:

def __init__(self, k: 'int'):
self.data = [-1] * k
self.capacity = k
self.real_cap = 0
self.__front, self.__rear = 0, 1

def insertFront(self, value: 'int') -> 'bool':
```

```
        if self.real_cap == self.capacity:
        return False # deque is full now
        else:
        self.real_cap += 1
        self.data[self.__front] = value
        self.__front = (self.__front - 1 + self.capacity) % self.capacity

        return True


        def insertLast(self, value: 'int') -> 'bool':
        if self.real_cap == self.capacity:
        return False
        else:
        self.real_cap += 1
        self.data[self.__rear] = value
        self.__rear = (self.__rear + 1 + self.capacity) % self.capacity

        return True


        def deleteFront(self) -> 'bool':
        if self.isEmpty():
        return False
        else:
        self.real_cap -= 1
        self.__front = (self.__front + 1 + self.capacity) % self.capacity
        self.data[self.__front] = -1

        return True
```

```
def deleteLast(self) -> 'bool':
if self.isEmpty():
return False
else:
self.real_cap -= 1
self.__rear = (self.__rear - 1 + self.capacity) % self.capacity
self.data[self.__rear] = -1

return True



def getFront(self) -> 'int':
return self.data[(self.__front + 1 + self.capacity) % self.capacity]



def getRear(self) -> 'int':
return self.data[(self.__rear - 1 + self.capacity) % self.capacity]



def isEmpty(self) -> 'bool':
return self.real_cap == 0



def isFull(self) -> 'bool':
return self.real_cap == self.capacity
```

- 你看起来很好吃 2019-02-09 14:32:29

  逆波兰表达式python实现，时间复杂度O(n), 空间复杂度O(1)，

```
class Solution:
def evalRPN(self, tokens: 'List[str]') -> 'int':
data = []
```

```
opera = {'+', '-', '*', '/'}
for item in tokens:
if item in opera:
if item == '+':
r = data.pop() + data.pop()
data.append(r)
elif item == '-':
a, b = data.pop(), data.pop()
data.append(b - a)
elif item == '*':
a, b = data.pop(), data.pop()
data.append(b * a)
elif item == '/':
a, b = data.pop(), data.pop()
data.append(int(b /a))
else:
data.append(int(item))



return data.pop()
```

- 纯洁的憎恶 2019-02-09 13:00:13

  1.维护一个栈，顺序遍历括号序列，若与栈首括号匹配成功，则出栈并遍历下一个括号。遍历完毕后若栈为空则返回true。

  2.我比较笨，用空间降低逻辑复杂度吧。申请长度为n的bool数组S，初始化全为false，记录匹配成功的情况。遍历括号字符串A，若当前字符与栈首对应的字符不匹配，或栈为空，则将字符在A数组中的下标入栈。若字符与栈首对应的字符匹配，则出栈，并将它们在A中下标对应S中的位置设置为true。遍历A结束后，再扫一遍S，输出连续true最长的位数。

  3.读取字符，若为数字则入栈，若为运算符则连续出栈两次，根据运算符计算，将结果入栈。输出最终结果即可。

  4.用数组实现循环双端队列。

5.每个窗口计算一次最大值，时间复杂度O（nk）。感觉有更好的方法，其实只要通过队列维护每个窗口的最大值，以及最大值右侧的次大值即可（实现细节需要打磨），这样时间复杂度为O（n）。

6.寻找递归公式
f（0）= 0;
f（1）= 1;
f（2）= 2;
f（3）= 2+1;
f（n）= f（n-1）+ f（n-2）； {n大于2}

- 老杨同志 2019-02-08 15:07:43

```
package com.jxyang.test.geek.day2;
//爬梯子、斐波那契数列
class Solution {
public int climbStairs(int n) {
if(n<=0){
return 0;
}else if(n<2){
return 1;
}
int[] status = new int[n+1];
status[0]=1;
status[1]=1;
for(int i =2;i<=n;i++){
status[i] = status[i-1]+status[i-2];
}
return status[n];
}

public static void main(String[] args) {
Solution solution = new Solution();
System.out.println(solution.climbStairs(2));
```

```
System.out.println(solution.climbStairs(3));

}
}
```

- 老杨同志 2019-02-08 00:14:13

全排列

```
import java.util.ArrayList;
import java.util.List;

//全排列
public class FullPermutation {
public static void main(String[] args) {
FullPermutation full = new FullPermutation();
int[] arr = {1,2,3,4};
full.printAllSort(arr);
}

public void printAllSort(int[] arr) {
if(arr==null || arr.length==0){
return;
}
if(arr.length==1){
System.out.println(arr[0]);
}

List<List<Integer>> result = _printAllSort(arr);
for(List list :result){
System.out.println(list);
}
}
```

```
private List<List<Integer>> _printAllSort(int[] tmpArr) {
//结束条件
List<List<Integer>> result = new ArrayList<>();
if(tmpArr.length==2){
List<Integer> subList = new ArrayList<>();
List<Integer> subList2 = new ArrayList<>();
subList.add(tmpArr[0]);
subList.add(tmpArr[1]);
subList2.add(tmpArr[1]);
subList2.add(tmpArr[0]);
result.add(subList);
result.add(subList2);
return result;
}
//当前层处理
for(int i=0;i<tmpArr.length;i++){
//顺序拿出一个参数，其余交给下一层处理
int tmp = tmpArr[i];
int[] arr = new int[tmpArr.length - 1];
int offset = 0;
for(int j=0;j<tmpArr.length;j++){
if(i!=j){
arr[offset] = tmpArr[j];
offset++;
}
}
List<List<Integer>> nextLevelResult = _printAllSort(arr);
//处理下一层结果（当前值加到结果的前面、后面）
for(List<Integer> nextList:nextLevelResult){
List<Integer> appendList = new ArrayList<>();
appendList.add(tmp);
appendList.addAll(nextList);
```

```
result.add(appendList);
/* nextList.add(tmp);
result.add(nextList);*/

}
}
return result;
}
}
```

- 你看起来很好吃 2019-02-08 00:12:23

有效的括号python代码实现：

```
class Solution:
def isValid(self, s):
"""
:type s: str
:rtype: bool
"""
stack = []
paren_map = {')':'(', ']':'[', '}':'{'}

for item in s:
if item in ['(', '[', '{']:
stack.append(item)
else:
if not stack:
return False
elif paren_map[item] != stack.pop():
return False
return not stack
```

这题使用栈是没有问题的，我觉得最巧妙的一点就是对匹配的符号建立字典，通过字典去栈里查找，这样效率最高

- 老杨同志 2019-02-06 23:02:36

```
package com.jxyang.test.geek.day2;
//链表实现栈
public class LinkStock<T> {
private Node<T> head;
public static void main(String[] args) {
LinkStock<Integer> stock = new LinkStock<Integer>();
stock.push(1).push(2).push(3).push(4).push(5);
Integer tmp =null;
while ((tmp = stock.pop())!=null){
System.out.println(tmp);
}
}
public LinkStock<T> push(T val){
if(val!=null){
Node tmp = new Node(val);
tmp.setNext(head);
head = tmp;
}
return this;
}
public T pop(){
if(head==null){
return null;
}else{
Node<T> tmp = head;
head = head.getNext();
return tmp.getValue();
}
}
}
```

老杨同志 2019-02-06 23:02:23

```
import java.util.ArrayList;

//数组实现队列,固定大小，暂时未实现扩容
public class ArrayQueue<T> {
private Object[] arr;
private int capacity;
private int head = 0;
private int tail = 0;
public ArrayQueue(){
this(16);//默认初始化16个
}
public ArrayQueue(int capacity){
assert capacity>0;
this.capacity = capacity;
this.arr = new Object[capacity];
}
public void offer(T val){
if(head-capacity == tail){
//没空间了
throw new RuntimeException("queue is full");
}
arr[(head+1)%capacity] = val;
head++;
}
public T poll(){
if(tail == head){
//没空间了
throw new RuntimeException("queue is empty");
}
T tmp = (T)arr[(tail+1)%capacity];
tail++;
```

```
return tmp;
}

public static void main(String[] args) {
ArrayQueue queue = new ArrayQueue(3);
queue.offer(1);queue.offer(2);queue.offer(3);
//queue.offer(4);//报错
System.out.println(queue.poll());
System.out.println(queue.poll());
System.out.println(queue.poll());
//System.out.println(queue.poll());//报错

}
}
```

- 失火的夏天 2019-02-06 22:51:05

  自己手动实现一个双端队列，其实只要会自己写实现一个链表，思路基本是一致的。用好头尾指针就可以解决一切问题，因为代码太长，就只贴上核心部分了，

```
// 双端队列
private static class DequeNode{
int val;
DequeNode prev;
DequeNode next;

DequeNode(int val){
this.val = val;
}
}

private DequeNode head;
private DequeNode tail;
private int length;
```

```
private int size = 0;

public boolean insertFront(int value) {
if (isFull()){
return false;
}
DequeNode newNode = new DequeNode(value);
if (isEmpty()){
head = tail = newNode;
} else {
newNode.next = head;
head.prev = newNode;
head = newNode;
}
this.size++;
return true;
}

public boolean insertLast(int value) {
if (isFull()){
return false;
}
DequeNode newNode = new DequeNode(value);
if (isEmpty()){
head = tail = newNode;
} else {
newNode.prev = tail;
tail.next = newNode;
tail = newNode;
}
this.size++;
return true;
```

```
}

public boolean deleteFront() {
if (isEmpty()){
return false;
}
head = head.next;
if (head != null){
head.prev = null;
}
this.size--;
return true;
}

public boolean deleteLast() {
if (isEmpty()){
return false;
}
tail = tail.prev;
if (tail != null){
tail.next = null;
}
this.size--;
return true;
}
```

- 失火的夏天 2019-02-06 22:48:57

```
// 有效的括号
public boolean isValid(String s) {
Map<Character,Character> map = new HashMap<>();
map.put(')','(');
map.put(']','[');
```

```
map.put('}','{');
Stack<Character> stack = new Stack<>();
for (int i = 0;i<s.length();i++){
Character c1 = s.charAt(i);
Character c2 = map.get(c1);
if (c2 == null){
stack.push(c1);
}else if(stack.isEmpty() || !c2.equals(stack.pop())){
return false;
}

}
return stack.isEmpty();
}

// 爬楼梯
public int climbStairs(int n) {
if(n <= 1){
return 1;
}else if(n == 2){
return 2;
}else {
int one = 1;
int two = 2;
int sum = 0;
for(int i = 2;i<n;i++){
sum = one + two;
one = two;
two = sum;
}
return sum;
}
```

}

- 黄丹 2019-02-06 20:09:47

王争老师新年快乐呀，我今天走亲戚去啦，队列的两题还没做TaT。下面放上栈和递归的四题的解题思路和代码
栈是一种受限制的线性表，只允许在栈顶进行操作（插入，取出，取值），Java已经为我们封装了一个这样的数据结构Stack, 对应的函数是（push,pop,peak）

1. Valid Parentheses（有效的括号）

解题思路：使用栈来做,遍历字符数组，当遇到 {,(,[时就入栈，当遇到],),} 时就出栈，如果栈为空或者取出的字符不匹配时，这表明不是有效的括号，返回false，当字符数组遍历完后，如果栈为空，代表这是有效括号，返回true，否则返回false。
代码：
https://github.com/yyxd/leetcode/blob/master/src/leetcode/stack/Problem20_ValidParentheses.java

2. Longest Valid Parentheses（最长有效括号）

解题思路：这一题我是用栈做的，但也可以用队列来做，复杂度也是O(n)，这里的小trick是将数组的下标入栈，当”)”匹配到”(”时，可以利用数组下标来计算当前有效括号的长度，
代码：https://github.com/yyxd/leetcode/blob/master/src/leetcode/stack/Problem32_LongestValidParentheses.java

3. Evaluate Reverse Polish Notation（逆波兰表达式求值）

解题思路：这一题是很中规中矩的用栈去做，将操作数始终放在栈顶，遇到操作符时取出栈顶的两个操作数进行相应操作，之前写过一个编译器，解析四元式时进行计算就是讲操作数放在栈顶进行操作的。
代码：https://github.com/yyxd/leetcode/blob/master/src/leetcode/stack/Problem150_EvlRPN.java

4. Climbing Stairs（爬楼梯）

解题思路：想到达第n级台阶时，可以选择从第n-1级台阶向上迈一步，也可以选择从第n-2级台阶向上迈两步，因此到达第n级台阶时有f(n) = f(n-1)+f(n-2)级台阶，这就和斐波那契数列一样。可以用递归做也可以用动态规划做。
代码很简单就不放了

- _CountingStars 2019-02-06 19:23:14

阶乘 go 语言实现
package main

import "fmt"

```
func factorial(n int) int {
if n == 0 || n == 1 {
return 1
}
return n * factorial(n-1)
}

func main() {
fmt.Println(factorial(5))
}
```