春节7天练讲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！

极客时间
数据结构与算法之美
为工程师量身打造的数据结构与算法私教课
王争
前 Google 工程师
新版升级：点击「请朋友读」，10位好友免费读，邀请订阅更有现金奖励。

精选留言

abner
java用链表实现一个链式栈
代码如下：

```
package stack;

public class LinkedStack {

private Node top = null;

public static class Node {

private String data;
private Node next;

public Node(String data, Node next) {
this.data = data;
this.next = next;
}

public String getData() {
return data;
}
}

public void push(String item) {
Node newNode = new Node(item, null);
if (top == null) {
top = newNode;
} else {
newNode.next = top;
top = newNode;
}
}

public String pop() {
if (top == null) {
return null;
}
String value = top.data;
top = top.next;
return value;
}

public void printAll() {
Node pNode = top;
while (pNode != null) {
System.out.print(pNode.data + " ");
pNode = pNode.next;
}
System.out.println();
}

public static void main(String[] args) {
LinkedStack linkedStack = new LinkedStack();
linkedStack.push("haha");
linkedStack.push("nihao");
linkedStack.printAll();
}
}
```

2019-02-12 13:46

abner

java用递归实现斐波那契数列

代码如下：

```
package recursion;

public class Fib {

public long calFib(long n) {
if (n == 0 || n == 1) {
return 1;
} else {
return calFib(n - 1) + calFib(n - 2);
}
}

public static void main(String[] args) {
```

```
Fib fib = new Fib();
long result = fib.calFib(5);
System.out.println(result);
}
}
```

2019-02-11 17:24

abner

java用递归实现求解n!

代码如下：

```
package recursion;

public class Fac {

public long calFac(long n) {
if (n == 0) {
return 1;
}
return calFac(n - 1) * n;
}

public static void main(String[] args) {
Fac fac = new Fac();
long result = fac.calFac(10);
System.out.println(result);
}
}
```

2019-02-11 17:07

abner

java用数组实现一个顺序队列

代码如下：

```
package queue;

public class ArrayQueue {

private String[] data;
private int size;
private int head;
private int tail;

public ArrayQueue(int capacity) {
data = new String[capacity];
size = capacity;
head = 0;
tail = 0;
}

public boolean enqueue(String value) {
if (tail == size) {
return false;
}
data[tail] = value;
tail++;
return true;
}

public String dequeue() {
if (tail == 0) {
return null;
}
String value = data[head];
head++;
return value;
}
}
```

2019-02-11 16:50

abner

java用数组实现一个顺序栈

代码如下：

```
package stack;

public class ArrayStack {

private String[] data;
private int count;
private int size;

public ArrayStack(int n) {
this.data = new String[n];
this.count = 0;
this.size = n;
}

public boolean push(String value) {
if (count == size) {
return false;
} else {
data[count] = value;
count++;
return true;
}
}

public String pop() {
if (count == 0) {
return null;
} else {
count--;
return data[count];
}
}
}
```

2019-02-11 16:12

kai

1. 编程实现斐波那契数列求值 f(n)=f(n-1)+f(n-2）

```
public class Fibonacci {
public static int fib(int n) {
if (n <= 0) {
return 0;
}
if (n == 1) {
return 1;
}

return fib(n-1) + fib(n-2);
}
}
```

2. Climbing Stairs（爬楼梯）

```
public class ClimbStairs {
public int climbFloor(int n) {
if (n == 1 || n == 2) {
return n;
}

return climbFloor(n - 1) + climbFloor(n - 2);
}

public int climbFloorIter(int n) {
if (n == 1 || n == 2) {
return n;
}

int jump1 = 1;
int jump2 = 2;
int jumpN = 0;

for (int i = 3; i <= n; i++) {
jumpN = jump1 + jump2;

jump1 = jump2;
jump2 = jumpN;
}

return jumpN;
}
}
```

3. Sliding Window Maximum（滑动窗口最大值)

```
import java.util.ArrayList;
import java.util.LinkedList;
```

```
public class MaxNumOfSlidingWindow {
public ArrayList<Integer> maxInWindows(int [] num, int size)
{
ArrayList<Integer> res = new ArrayList<>();

if (num == null || num.length <= 0 || size <= 0 || size > num.length) {
return res;
}

LinkedList<Integer> qMax = new LinkedList<>(); // 双端队列：左端更新max,右端添加数据

int left = 0;

for (int right = 0; right < num.length; right++) {
// 更新右端数据
while (!qMax.isEmpty() && num[qMax.peekLast()] <= num[right]) {
qMax.pollLast();
}

qMax.addLast(right);

// 更新max：如果max的索引不在窗口内,则更新
if (qMax.peekFirst() == right - size) {
qMax.pollFirst();
}

// 待窗口达到size，输出max
if (right >= size-1) {
res.add(num[qMax.peekFirst()]);
left++;
}
}

return res;
}
}
```

2019-02-11 10:25

ALAN

```
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;
}
}
```

2019-02-08 14:14

吴...

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。

2019-02-06 15:29

李皮皮皮皮

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。

2019-02-05 21:22

李汶泽

```
//用链表实现顺序栈
#include<stdlib.h>
#define true 1
#define false 0
#define ok 1
#define error 0
#define infeasible 1
#define overflow 0
#define stack_size 50
typedef struct{
int *base;
int *top;
int stacksize;
}sqstack;

//构造一个空栈
int create_stack(sqstack *s)
{
s->base=(int *)malloc(5*sizeof(int)); //开始分配50个整形空间
if(!s->base) exit(overflow);
s->top=s->base;
s->stacksize=5;
return 0;
}

//插入新元素为栈顶元素
int stack_push(sqstack *s)
{
int e;
if(s->top - s->base>=5)
{ //栈满， 追加存储空间
s->base = (int *)realloc(s->base,(5+1)*sizeof(int));
if(!s->base) exit(overflow);//存储分配失败
s->top = s->base + 5;//新扩充空间后的栈顶指针位置
s->stacksize += 1;
}
printf("请输入要入栈的值:");
scanf("%d",&e);
*s->top++ = e;
return 0;
}

//出栈
int stack_pop(sqstack *s)
{
if(s->base == s->top) {printf("栈为空！不能出栈！"); return error;}
--s->top;
return 0;
}

//打印栈
int stack_top(sqstack *s)
{
int *w;
printf("The stack is :");
w=s->base;
while(w!=s->top)
{
printf(" %d ",*w++);
}
printf("\n");
}
```

2019-02-15 00:49

李汶泽

```
//数组实现顺序栈
public class MyStack {
Object[] object;
private int count;
MyStack(int size){
this.object = new Object[size];
count = 0;
}
public void push(Object h) {
if( (count+1) >= object.length) {
Object[] ob = new Object[2*object.length];
System.arraycopy(object, 0, ob, 0, count);
this.object = ob;
}
object[count] = h;
count++;
}
```

```
public Object pop() {
if(count==0) {
return null;
}else {
count--;
return object[count-1];
}
}
public Object peek() {
if(count==0) {
return null;
}else {
return object[count-1];
}
}
public void removeAll() {
while(count!=0) {
this.pop();
count--;
}
}
public boolean empty() {
if(count==0) {
return true;
}else
return false;
}
public int getCount() {
return this.count;
}
```

2019-02-15 00:46

吴...

递归爬楼梯

```
#include<iostream>
using namespace std;
int floor(int n){
if(n == 0) return 1;
else if(n == 1) return 1;
else return floor(n - 1) + floor(n - 2);
}
int main(){
int n;
cin>>n;
cout<<floor(n)<<endl;
}
```

2019-02-14 11:41

abner

java实现一个循环队列

代码如下：

```
package queue;
```

```java
public class CircularQueue {

private String[] data;
private int size;
private int head;
private int tail;

public CircularQueue(int capacity) {
data = new String[capacity];
size = capacity;
head = 0;
tail = 0;
}

public boolean enqueue(String item) {
if ((tail + 1) % size == head) {
return false;
}
data[tail] = item;
tail = (tail + 1) % size;
return true;
}

public String dequeue() {
if (head == tail) {
return null;
}
String value = data[head];
head = (head + 1) % size;
return value;
}

public void printAll() {
if (0 == size) {
return ;
}
for (int i = head;i % size != tail;i++) {
System.out.print(data[i] + " ");
}
System.out.println();
}

public static void main(String[] args) {
CircularQueue circularQueue = new CircularQueue(5);
circularQueue.enqueue("hello1");
circularQueue.enqueue("hello2");
circularQueue.enqueue("hello3");
circularQueue.enqueue("hello4");
circularQueue.dequeue();
circularQueue.printAll();
}
}
```

2019-02-12 20:20

abner

java实现一个链式队列

代码如下：

```
package queue;

public class LinkedQueue {

private Node head = null;
private Node tail = null;

public static class Node {

private String data;
private Node next;

public Node(String data, Node next) {
this.data = data;
this.next = next;
}

public String getData() {
return data;
}
}

public void enqueue(String item) {
if (tail == null) {
Node newNode = new Node(item, null);
head = newNode;
tail = newNode;
} else {
tail.next = new Node(item, null);
tail = tail.next;
}

}

public String dequeue() {
if (head == null) {
return null;
} else {
String value = head.data;
head = head.next;
if (head == null) {
tail = null;
}
return value;
}
}

public void printAll() {
Node pNode = head;
while (pNode != null) {
System.out.print(pNode.data + " ");
pNode = pNode.next;
}
System.out.println();
}

public static void main(String[] args) {
LinkedQueue linkedQueue = new LinkedQueue();
linkedQueue.enqueue("hello");
linkedQueue.enqueue("nihao");
linkedQueue.dequeue();
linkedQueue.printAll();
}

}
```

2019-02-12 19:29

神盾局闹别扭

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);

}
};
```

2019-02-10 15:40

yingyingqin

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
};
```

2019-02-09 22:59

神盾局闹别扭

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);
}
```

2019-02-09 19:47

molybdenum

老师新年好 这是我的作业

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

2019-02-09 16:33

你看起来很好吃

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
```

2019-02-09 16:09

你看起来很好吃

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
```

2019-02-09 15:39