数据结构与算法

一、数据结构

集合结构 线性结构 树形结构 图形结构

- 1.1、集合结构 说白了就是一个集合，就是一个圆圈中有很多个元素，元素与元素之间没有任何关系 这个很简单
- 1.2、线性结构 说白了就是一个条线上站着很多个人。 这条线不一定是直的。也可以是弯的。也可以是值的 相当于一条线被分成了好几段的样子 （发挥你的想象力）。 线性结构是一对一的关系
- 1.3、树形结构 说白了 做开发的肯定或多或少的知道 xml 解析 树形结构跟他非常类似。也可以想象成一个金字塔。树形结构是一对多的关系
- 1.4、图形结构 这个就比较复杂了。他呢 无穷。无边 无向（没有方向）图形机构 你可以理解为多对多 类似于我们人的交集关系

数据结构的存储

数据结构的存储一般常用的有两种 顺序存储结构 和 链式存储结构

- 2.1 顺序存储结构

发挥想象力啊。 举个列子。数组。1-2-3-4-5-6-7-8-9-10。这个就是一个顺序存储结构 ，存储是按顺序的 举例说明啊。 栈。做开发的都熟悉。栈是先进后出 ，后进先出的形式 对不对 ？！ 他的你可以这样理解

hello world 在栈里面从栈底到栈顶的逻辑依次为 h-e-l-l-o-w-o-r-l-d 这就是顺序存储 再比如 队列 ，队列是先进先出的对吧，从头到尾 h-e-l-l-o-w-o-r-l-d 就是这样排对的

- 2.2 链式存储结构

再次发挥想象力 这个稍微复杂一点 这个图片我一直弄好 ，回头找美工问问，再贴上 例如 还是一个数组

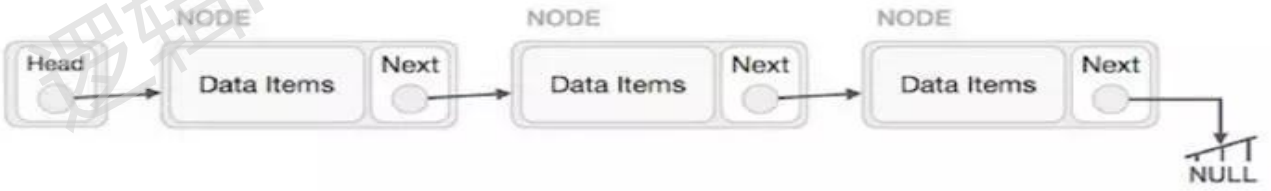
1-2-3-4-5-6-7-8-9-10 链式存储就不一样了 1(地址)-2(地址)-7(地址)-4(地址)-5(地址)-9(地址)-8(地址)-3(地址)-6(地址)-10(地址)。每个数字后面跟着一个地址 而且存储形式不再是顺序 ，也就说顺序乱了，1（地址）1 后面跟着的这个地址指向的是 2，2 后面的地址指向的是 3，3 后面的地址指向是谁你应该清楚了吧。他执行的时候是 1(地址)-2(地址)-3(地址)-4(地址)-5(地址)-6(地址)-7(地址)-8(地址)-9(地址)-10(地址)，但是存储的时候就是完全随机的。明白了？！

单向链表\双向链表\循环链表

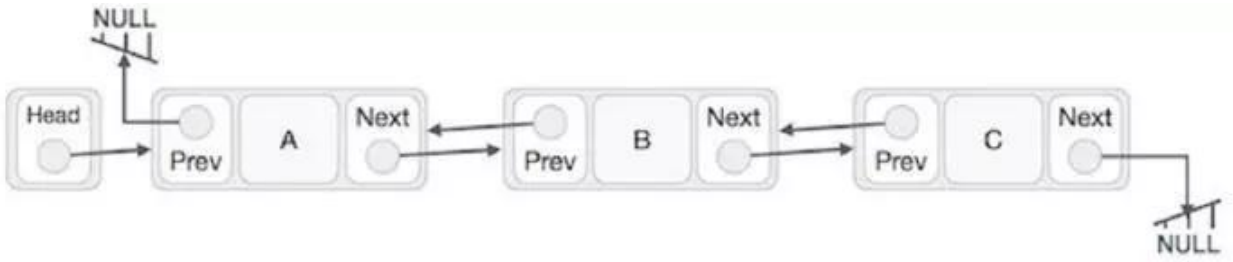
还是举例子。理解最重要。不要去死记硬背 哪些什么。定义啊。逻辑啊。理解才是最重要滴

- 3.1 单向链表

A->B->C->D->E->F->G->H. 这就是单向链表 H 是头 A 是尾 像一个只有一个头的火车一样 只能一个头拉着跑



- 3.2 双向链表



数组和链表区别：
数组：数组元素在内存上连续存放，可以通过下标查找元素；插入、删除需要移动大量元素，比较适用元素很少变化的情况
链表：链表中的元素在内存中不是顺序存储的，查找慢，插入、删除只需要对元素指针重新赋值，效率高

- 3.3 循环链表

循环链表是与单向链表一样，是一种链式的存储结构，所不同的是，循环链表的最后一个结点的指针是指向该循环链表的第一个结点或者表头结点，从而构成一个环形的链。发挥想象力
A->B->C->D->E->F->G->H->A. 绕成一个圈。就像蛇吃自己的这就是循环 不需要去死记硬背哪些理论知识。

二叉树 / 平衡二叉树

- 4.1 什么是二叉树

树形结构下,两个节点以内 都称之为二叉树 不存在大于 2 的节点 分为左子树 右子树 有顺序 不能颠倒 ，懵逼了吧，你肯定会想这是什么玩意，什么左子树右子树 ，都什么跟什么鬼？ 现在我以普通话再讲一遍，你把二叉树看成一个人 ，人的头呢就是树的根 ，左子树就是左手，右子树就是右手，左右手可以都没有（残疾嘛，声明一下，绝非歧视残疾朋友，勿怪，勿怪就是举个例子，i am very sorry） ，左右手呢可以有一个，就是不能颠倒。这样讲应该明白了吧

二叉树有五种表现形式

1.空的树（没有节点）可以理解为什么都没 像空气一样
2.只有根节点。 （理解一个人只有一个头 其他的什么都没，说的有点恐怖）
3.只有左子树 （一个头 一个左手 感觉越来越写不下去了）
4.只有右子树
5.左右子树都有

二叉树可以转换成森林 树也可以转换成二叉树。这里就不介绍了 你做项目绝对用不到
数据结构大致介绍这么多吧。理解为主，别死记，死记没什么用

二、算法面试题（一）

1、不用中间变量,用两种方法交换 A 和 B 的值

```
// 1.中间变量
void swap(int a, int b) {
   int temp = a;
   a = b;
   b = temp;
}

// 2.加法
void swap(int a, int b) {
   a = a + b;
   b = a - b;
   a = a - b;
}

// 3.异或（相同为0，不同为1。 可以理解为不进位加法）
void swap(int a, int b) {
   a = a ^ b;
   b = a ^ b;
   a = a ^ b;
}
```

2、 求最大公约数

```
/** 1.直接遍历法 */
int maxCommonDivisor(int a, int b) {
    int max = 0;
    for (int i = 1; i <=b; i++) {
        if (a % i == 0 && b % i == 0) {
            max = i;
        }
    }
    return max;
}
/** 2.辗转相除法 */
int maxCommonDivisor(int a, int b) {
    int r;
    while(a % b > 0) {
        r = a % b;
        a = b;
        b = r;
    }
    return b;
}

// 扩展：最小公倍数 = (a * b)/最大公约数
```

3、模拟栈操作

- 栈是一种数据结构，特点：先进后出 -
- 练习：使用全局变量模拟栈的操作

```
#include <stdio.h>
#include <stdbool.h>
#include <assert.h>

//保护全局变量：在全局变量前加 static 后，这个全局变量就只能在本文件中使用
static int data[1024];//栈最多能保存 1024 个数据
static int count = 0;//目前已经放了多少个数(相当于栈顶位置)
```

```
//数据入栈 push
void push(int x){
    assert(!full());//防止数组越界
    data[count++] = x;
}
//数据出栈 pop
int pop(){
    assert(!empty());
    return data[--count];
}
//查看栈顶元素 top
int top(){
    assert(!empty());
    return data[count-1];
}

//查询栈满 full
bool full() {
    if(count >= 1024) {
        return 1;
    }
     return 0;
}

//查询栈空 empty
bool empty() {
    if(count <= 0) {
        return 1;
    }
    return 0;
}

int main(){
    //入栈
    for (int i = 1; i <= 10; i++) {
        push(i);
    }

    //出栈
    while(!empty()){
        printf("%d ", top()); //栈顶元素
        pop(); //出栈
    }
    printf("\n");

    return 0;
}
```

4、排序算法

选择排序、冒泡排序、插入排序三种排序算法可以总结为如下：
都将数组分为已排序部分和未排序部分。

1.选择排序将已排序部分定义在左端，然后选择未排序部分的最小元素和未排序部分的第一个元素交换。

2.冒泡排序将已排序部分定义在右端，在遍历未排序部分的过程执行交换，将最大元素交换到最右端。

3.插入排序将已排序部分定义在左端，将未排序部分元的第一个元素插入到已排序部分合适的位置。

4.1、选择排序

- 【选择排序】：最值出现在起始端
- 第 1 趟：在 n 个数中找到最小(大)数与第一个数交换位置
- 第 2 趟：在剩下 n-1 个数中找到最小(大)数与第二个数交换位置
- 重复这样的操作...依次与第三个、第四个...数交换位置
- 第 n-1 趟，最终可实现数据的升序（降序）排列。

```
void selectSort(int *arr, int length) {
    for (int i = 0; i < length - 1; i++) { //趟数
        for (int j = i + 1; j < length; j++) { //比较次数
            if (arr[i] > arr[j]) {
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
    }
}
```

4.2、冒泡排序

- 【冒泡排序】：相邻元素两两比较，比较完一趟，最值出现在末尾
- 第 1 趟：依次比较相邻的两个数，不断交换（小数放前，大数放后）逐个推进，最值最后出现在第 n 个元素位置
- 第 2 趟：依次比较相邻的两个数，不断交换（小数放前，大数放后）逐个推进，最值最后出现在第 n-1 个元素位置
- …… ……
- 第 n-1 趟：依次比较相邻的两个数，不断交换（小数放前，大数放后）逐个推进，最值最后出现在第 2 个元素位置

```
void bubbleSort(int *arr, int length) {
    for(int i = 0; i < length - 1; i++) { //趟数
        for(int j = 0; j < length - i - 1; j++) { //比较次数
            if(arr[j] > arr[j+1]) {
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
        }
    }
}
```

5、折半查找（二分查找）

折半查找：优化查找时间（不用遍历全部数据）
折半查找的原理：

- 1> 数组必须是有序的
- 2> 必须已知 min 和 max（知道范围）
- 3> 动态计算 mid 的值，取出 mid 对应的值进行比较
- 4> 如果 mid 对应的值大于要查找的值，那么 max 要变小为 mid-1
- 5> 如果 mid 对应的值小于要查找的值，那么 min 要变大为 mid+1

// 已知一个有序数组，和一个 key，要求从数组中找到 key 对应的索引位置

```
int findKey(int *arr, int length, int key) {
    int min = 0, max = length - 1, mid;
    while (min <= max) {
        mid = (min + max) / 2; //计算中间值
        if (key > arr[mid]) {
            min = mid + 1;
        } else if (key < arr[mid]) {
            max = mid - 1;
        } else {
            return mid;
        }
    }
    return -1;
}
```


逻辑iOS技术号

三、算法面试题（二）

字符串反转

给定字符串 "hello,world",实现将其反转。输出结果：dlrow,olleh

```
- (void)charReverse
{
    NSString * string = @"hello,world";

    NSLog(@"%@",string);

    NSMutableString * reverString = [NSMutableString stringWithString:string];

    for (NSInteger i = 0; i < (string.length + 1)/2; i++) {

        [reverString replaceCharactersInRange:NSMakeRange(i, 1) withString:
        [string substringWithRange:NSMakeRange(string.length - i - 1, 1)]];

        [reverString replaceCharactersInRange:NSMakeRange(string.length - i - 1, 1) withString:
        [string substringWithRange:NSMakeRange(i, 1)]];
    }

    NSLog(@"reverString:%@",reverString);

    //C
    char ch[100];

    memcpy(ch, [string cStringUsingEncoding:NSUTF8StringEncoding], [string length]);

   //设置两个指针，一个指向字符串开头，一个指向字符串末尾
    char * begin = ch;

    char * end = ch + strlen(ch) - 1;

//遍历字符数组，逐步交换两个指针所指向的内容，同时移动指针到对应的下个位置，直至begin>=end
    while (begin < end) {

        char temp = *begin;

        *(begin++) = *end;

        *(end--) = temp;
    }

    NSLog(@"reverseChar[]:%s",ch);
}
```


逻辑iOS技术号

序数组合并

将有序数组 {1, 4, 6, 7, 9} 和 {2, 3, 5, 6, 8, 9, 10, 11, 12} 合并为 {1, 2, 3, 4, 5, 6, 6, 7, 8, 9, 9, 10, 11, 12}

```
- (void)orderListMerge
{
    int aLen = 5,bLen = 9;

    int a[] = {1,4,6,7,9};

    int b[] = {2,3,5,6,8,9,10,11,12};

    [self printList:a length:aLen];

    [self printList:b length:bLen];

    int result[14];

    int p = 0,q = 0,i = 0;//p和q分别为a和b的下标，i为合并结果数组的下标

    //任一数组没有达到s边界则进行遍历
    while (p < aLen && q < bLen) {

        //如果a数组对应位置的值小于b数组对应位置的值,则存储a数组的值，并移动a数组的下标与合并结果数组的下
        if (a[p] < b[q]) result[i++] = a[p++];

        //否则存储b数组的值，并移动b数组的下标与合并结果数组的下标
        else result[i++] = b[q++];
    }

    //如果a数组有剩余，将a数组剩余部分拼接到合并结果数组的后面
    while (++p < aLen) {

        result[i++] = a[p];
    }

    //如果b数组有剩余，将b数组剩余部分拼接到合并结果数组的后面
    while (q < bLen) {

        result[i++] = b[q++];
    }

    [self printList:result length:aLen + bLen];
}
- (void)printList:(int [])list length:(int)length
{
    for (int i = 0; i < length; i++) {

        printf("%d ",list[i]);
    }

    printf("\n");
}
```


逻辑iOS技术号

HASH 算法

- 哈希表
  例：给定值是字母 a，对应 ASCII 码值是 97，数组索引下标为 97。
  这里的 ASCII 码，就算是一种哈希函数，存储和查找都通过该函数，有效地提高查找效率。
- 在一个字符串中找到第一个只出现一次的字符。如输入"abaccdeff"，输出'b'字符(char)是一个长度为 8 的数据类型，因此总共有 256 种可能。每个字母根据其 ASCII 码值作为数组下标对应数组种的一个数字。数组中存储的是每个字符出现的次数。

```
- (void)hashTest
{
    NSString * testString = @"hhaabccdeef";

    char testCh[100];

    memcpy(testCh, [testString cStringUsingEncoding:NSUTF8StringEncoding], [testString length]);

    int list[256];

    for (int i = 0; i < 256; i++) {

        list[i] = 0;
    }

    char *p = testCh;

    char result = '\0';

    while (*p != result) {

        list[*(p++)]++;
    }

    p = testCh;

    while (*p != result) {

        if (list[*p] == 1) {

            result = *p;

            break;
        }

        p++;
    }

    printf("result:%c",result);
}
```


逻辑iOS技术号

查找两个子视图的共同父视图

思路:分别记录两个子视图的所有父视图并保存到数组中，然后倒序寻找,直至找到第一个不一样的父视图。

```
- (void)findCommonSuperViewsWithView:(UIView *)view1 view2:(UIView *)view2
{
    NSArray * superViews1 = [self findSuperViews:view1];

    NSArray * superViews2 = [self findSuperViews:view2];

    NSMutableArray * resultArray = [NSMutableArray array];

    int i = 0;

    while (i < MIN(superViews1.count, superViews2.count)) {

        UIView *super1 = superViews1[superViews1.count - i - 1];

        UIView *super2 = superViews2[superViews2.count - i - 1];

        if (super1 == super2) {

            [resultArray addObject:super1];

            i++;

        }else{

            break;
        }
    }

    NSLog(@"resultArray:%@",resultArray);

}
- (NSArray <UIView *>*)findSuperViews:(UIView *)view
{
    UIView * temp = view.superview;

    NSMutableArray * result = [NSMutableArray array];

    while (temp) {

        [result addObject:temp];

        temp = temp.superview;
    }

    return result;
}
```

求无序数组中的中位数

中位数：当数组个数 n 为奇数时，为(n + 1)/2，即是最中间那个数字；当 n 为偶数时，为(n/2 + (n/2 + 1))/2,即是中间两个数字的平均数。
首先要先去了解一些几种排序算法:iOS 排序算法
思路:

- 1.排序算法+中位数
  首先用冒泡排序、快速排序、堆排序、希尔排序等排序算法将所给数组排序，然后取出其中位数即可。
- 2.利用快排思想