第一周学习总结

一、Git & Github:

查看git版本：git --version

初试化git仓库：git init

设置用户名称和e-mail地址：git config --global user.name “用户名”

git config --global user.name.email “邮箱”

查看配置信息：git config –global –list

查看git仓库当前状态：git status

暂存操作：git add learn\_git.html

把所有已修改的文件添加到暂存区：**git add .**

提交操作：git commit -m “week01”

把本地文件推送到远程仓库上：

git remote add origin [git@github.com:wlgwlg/learn\_git.git](mailto:git@github.com:wlgwlg/learn_git.git)

git push -u origin master

二、数组、链表和跳表

1、数组：

优点：地址连续，访问任意一项都是常数级别O(1);

缺点：删除，增加元素时间复杂度高，需要遍历，一般为O(n)，插入之前保证数组的size足够；数组size不够时需要扩容和拷贝等低级操作。

2、链表

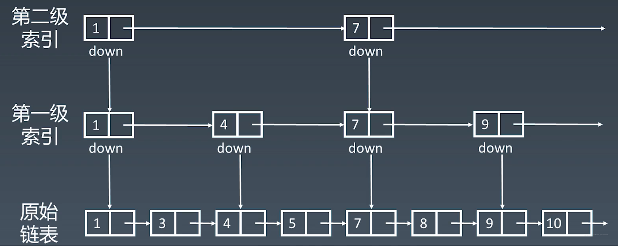
单链表、双向链表、循环链表

优点：灵活，插入不需要扩容、移动其他元素或者拷贝等操作O(1);

缺点：占用内存，遍历查找不方便，必须挨个遍历，不能用下标访问，一般为O(n)。

3、跳表

为了优化链表缺陷（lookup查找）时间复杂度为O(n)，就有了跳表，实质就是空间换时间，增加索引：每一级索引跳前一级索引的倍数：2-4-8-16



三、栈、队列、优先队列、双端队列

**栈**

**特点：**栈的最大特点就是后进先出（LIFO）。对于栈中的数据来说，所有操作都是在栈的顶部完成的，只可以查看栈顶部的元素，只能够向栈的顶部压⼊数据，也只能从栈的顶部弹出数据。栈顶元素反映了在嵌套的层次关系中，最近的需要匹配的元素。

**实现：**利用一个单链表来实现栈的数据结构。而且，因为我们都只针对栈顶元素进行操作，所以借用单链表的头就能让所有栈的操作在 O(1) 的时间内完成。

应用场景：在解决某个问题的时候，只要求关心最近一次的操作，并且在操作完成了之后，需要向前查找到更前一次的操作。

注意：如果打算用一个数组外加一个指针来实现相似的效果，那么，一旦数组的长度发生了改变，哪怕只是在最后添加一个新的元素，时间复杂度都不再是 O(1)，而且，空间复杂度也得不到优化。

**队列（Queue）**

**特点：**和栈不同，队列的最大特点是先进先出（FIFO），就好像按顺序排队一样。对于队列的数据来说，我们只允许在队尾查看和添加数据，在队头查看和删除数据。

**实现：**可以借助双链表来实现队列。双链表的头指针允许在队头查看和删除数据，而双链表的尾指针允许我们在队尾查看和添加数据。

应用场景：直观来看，当我们需要按照一定的顺序来处理数据，而该数据的数据量在不断地变化的时候，则需要队列来帮助解题

**优先队列**

**特点：**能保证每次取出的元素都是队列中优先级别最高的。优先级别可以是自定义的，例如，数据的数值越大，优先级越高；或者数据的数值越小，优先级越高。优先级别甚至可以通过各种复杂的计算得到。优先队列的底层实现是堆。

**应用场景：**从一堆杂乱无章的数据当中按照一定的顺序（或者优先级）逐步地筛选出部分乃至全部的数据。

**双端队列（Deque）**

**特点：**双端队列和普通队列最大的不同在于，它允许我们在队列的头尾两端都能在 O(1) 的时间内进行数据的查看、添加和删除

**实现：**与队列相似，我们可以利用一个双链表实现双端队列

**应用场景：**双端队列最常用的地方就是实现一个长度动态变化的窗口或者连续区间，而动态窗口这种数据结构在很多题目里都有运用