第一题：给定 pushed 和 popped 两个序列，只有当它们可能是在最初空栈上进行的推入 push 和弹出 pop 操作序列的结果时，返回 true；否则，返回 false 。

****输入：****pushed = [1,2,3,4,5], popped = [4,5,3,2,1]****输出：****true****解释：****我们可以按以下顺序执行：

push(1), push(2), push(3), push(4), pop() -> 4,

push(5), pop() -> 5, pop() -> 3, pop() -> 2, pop() -> 1

出处：<https://leetcode-cn.com/contest/weekly-contest-112/problems/validate-stack-sequences/>

答案：<https://github.com/Black-Mamba24/leetcode-python/blob/master/946_ValidateStackSequence_stack.py>

第二题：谈谈在1.8中HashMap什么时候扩容，如何扩容？

答案：

何时扩容：当map中元素个数超过threshold时扩容，threshold = capacity \* loadFactor。capacity是2的倍数，用户可以指定，loadFactor是负载因子，用户可以指定，默认为0.75。

如何扩容：

新数组容量是原数组的两倍，除非原数组的长度已经超过最大长度，这种情况不会扩容。因为是双倍扩容，所以原来落在x位置上的元素，扩容后要么落在x上，要么落在x + capacity位置上，这是成倍扩容隐含的好处。

接着创建新数组，遍历原数组，如果原数组某个点位是单独的一个点，则再次hash放入新数组。

如果是链表，则会遍历链表，将其分解为两个链表，一个位于当前点位，另一个位于x + capacity点位，每个链表设置头尾节点，尾节点便于插入。遍历结束后，将两个链表放在合适的位置。其判断属于哪个链表的方法为用节点的hash & capacity，原数组的范围是0 ~capacity - 1，因此落在原位的节点与操作后一定是0，否则一定不是0。

如果是红黑树，跟链表的思路类似。HashMap中树节点不仅包含parent、left、right，同时也包含next、prev。拥有树和链表的双重特性。一个节点是链表还是红黑树取决于链表的长度和树的元素个数，两者相互转换，引入红黑树的原因是优化当链表过长时查找性能线性降低的问题，红黑树是一颗二叉搜索平衡树，它的查找时间复杂度固定为O(lgn)。分解一棵树时也会准备两个链表，原理跟分解链表相同，如果链表的长度没超过树的阈值，则不会树化，直接存为链表；如果数量仍比较多，则需要将其树化。

师妹回答：

当内个，hashmap当中，kv占据了，不对，，初始大小，默认16，四分之三，就会扩容，扩容两倍。0.75是个阈值，不可以自己设置，好像也可以，最终是可以。不知道为啥扩两倍。

Hashmap用什么数据结构存的。

1.8 可以放null，key。

Hash冲突了怎么办，链表，一直往后连。链表的问题，查询慢。优化查询，不知道。

Hashmap线程不安全，不是同步的，实现线程安全，用concurrenthashmap。

初始化hashmap，默认16。立即会有16个。

形式，kv，可以containsKey，错了，如何检查是否包含某个value,让我思考一下，，不会

让我思考一下。。Put、get、isEmpty，，，不知道了

第三题：谈谈对swap的理解，swap的功能，swap的工作原理，swapness参数的含义，什么时候应该使用或避免使用swap。

答案：

Swap的功能：swap是解决物理内存不足问题，会将部分内存分页换出到磁盘上，防止出现内存溢出错误，扩大了应用寻址范围。使用free -m查看Swap部分，使用

Swap的工作原理：

内存可粗略分为两类，一类是匿名内存（anon），一类是页缓存（file），例如Java的堆、栈都对应匿名内存，如果使用mmap映射文件会用到file内存，file内存时堆外内存的一部分，即page cache，即free -m命令的Cached部分，也简称为缓存，这部分也是JVM进程和操作系统公共的内存区域。如果使用FileInputStream或FileOutputStream，也会占用file内存，但这部分内存不是公共的，只有操作系统能访问，因此在做文件读写时就会有上下文切换，JVM从这块堆外内存中读写文件。

两部分内存分别维护两个链表，分别记录活跃（active）和不活跃（inactive）内存页。操作系统使用LRU策略，管理链表，找出哪些页面应该被换出。扫描active，找出不活跃的放入inactive尾部，扫描inactive，找出active放入active尾部，会换出inactive队首的页面。(LRU算法的实现)

对于匿名内存swap会生效，操作系统中kswapd进程会定时唤醒，换出inactive的页面。对于page cahce即file内存，操作系统会有pdflush等刷新缓存的进程负责将数据写回磁盘文件或释放，取决于页面是否为脏页，脏页写回，干净页释放。

当kswapd执行时，何时应该换出，换出多少页面应该如何确定呢，结合swapness参数继续解答

Swapness参数含义：

内存回收 = 换出匿名内存 + 写回或清空页缓存，swapness就是平衡两者关系的。当swapness越大，会越倾向于（aggressive）使用swap（磁盘上），即换出匿名内存，swapness越小，会更少的使用swap，进而更多清理页缓存。当swapness == 100，两者有相同优先级，当swapness == 0时如果发生物理内存不足时才会使用swap。

可用内存 = free + cached，即页缓存也是可用内存的一部分。内存会设置3种水位，分别是min、low、high，度量的是可用内存的大小。当可用内存非常多时，即在high的右边不会发生swap，当低于high时则会开始swap，直到高于high且低于low，这也是为什么内存还有空闲物理内存，操作系统就会执行页面换出，使用swap空间，kswapd的目标是可用内存大小高于high水位。当位于min左边，即空闲内存非常少，当再次分配内存时会触发direct reclaim，强制回收内存，此处不多解释。以上解释了什么时候触发swap操作，接着要换出多少内存呢？我们可以认为用清理page cache来代替swap out，因为page cache中会有干净页面，这时不发生磁盘IO，而swap out一定跟磁盘交互。这是关键。水位的min可以通过设置/proc/sys/vm/min\_free\_kbytes指定，不应配置太小，操作系统会根绝min和物理内存大小计算low和high。

Swap的使用场景：

如果是延时敏感型应用，应尽量不使用swap，这些应用往往拥有大内存，通常不会用完内存，因此也很少的使用swap空间。同时也不应关闭swap，防止出现内存溢出。

如果对延时不敏感，应该配置合适的swapness，保证时刻都有足够的空闲内存。

师妹回答：

Top命令，内存部分，total，buff，swap，usr，use，我可是专门学过top的。

Swap在你物理内存不够用时，会把不常用的信息对象程序，swap出去到swap区，虚拟内存，比内存慢，比磁盘快。根据自己的应用决定swap的大小，不是越大越好。

什么时候回swap出去，物理内存不足时，重新分配内存，发现不够用了。标机最后访问时间，访问频次指标，很久没被访问。换进来，程序执行时调用到他，会产生上下文切换，影响性能，上下文切换的原因是地址发生改变。

好处：物理内存不足时，暂时弥补，比放在磁盘快很多

坏处：导致不停的swap in， swap out，cpu使用率上升，系统性能下降。

认为控制：

swapon -s观察现在的swap配置是多少

/proc/swapness，没改过，现在是10%

使用场景：

在程序大对象，文件类的，应该更多的使用swap

对象被频繁使用情况下，应该避免swap