语言知识点记录

Java数组切片 java.util.Arrays.copyOf(arr,k);可以当做数组切片

Split方法里面所有保留字需要加转义符号”\\”

剑指offer 22， 链表倒数第k个节点

用快慢指针：

不需要遍历两遍就能找到。

链表删除节点，如果给了head，就查到这个节点 把上个节点next改到下一个。 如果没给head，就依次往后覆盖，并把倒数第二个的next 改成null。

归并排序：

class Solution {

    public int[] smallestK(int[] arr, int k) {

        if(k==0){return new int[0];}

        if(k==arr.length){return arr;}

        // if(k<=arr.length/2){

        //     retrun find\_min(arr,k);

        // }

        // else{

        //     return find\_max(arr,arr.length-k);

        // }

        sort(arr,0,arr.length-1);

        return java.util.Arrays.copyOf(arr,k);

    }

    public static void sort(int[] arr,int start, int end){

        int mid=(start+end)/2;

        if(start<end){

            sort(arr,start,mid);

            sort(arr,mid+1,end);

            merge(arr,start,end,mid);

        }

    }

    public static void merge(int[] arr,int start, int end, int mid){

        int [] tmp=new int[end-start+1];

        int first\_idx=start;

        int sec\_idx=mid+1;

        int count=0;

        while(first\_idx <=mid && sec\_idx<=end){

            if(arr[first\_idx]<arr[sec\_idx]){

                tmp[count]=arr[first\_idx];

                first\_idx+=1;

                count+=1;

            }

            else{

                tmp[count]=arr[sec\_idx];

                sec\_idx+=1;

                count+=1;

            }

        }

        while(first\_idx<=mid){

            tmp[count]=arr[first\_idx];

            count+=1;

            first\_idx+=1;

        }

        while(sec\_idx<=end){

            tmp[count]=arr[sec\_idx];

            count+=1;

            sec\_idx+=1;

        }

        int idx=0;

        for(int i=start;i<=end;i++){

            arr[i]=tmp[idx];

            idx+=1;

        }

    }

环形链表判断也可以通过快慢指针！ 如果快慢指针能相遇则说明有环！

随机数转换：

1. Rand7(1,2…7) 转 rand10 ：大转小可以任意转， 例如rand7转rand2， 可以令123返回1，567返回2，如果随机到4就重新调用rand7，直到不是4为止。
2. 公式 (rand(x)-1)\*y+rand(y)= rand(x\*y),用10进制很好理解，rand(0-9)\*10，那么就是0，10.。。。90， 加上随机的个位数 正好是1-100等概率随机。

(Rand7-1) \*rand7 =rand(49),然后可以用1的方式 只留下 rand(40)，可以直接mod10，也可以整除以4.

素数检测法：找出小于n的所有质数。非素数可以分解为质数的因数，因此从质数开始乘以其他数能够枚举出所有非素数。

埃氏检测，构造辅助Boolean[] n 数组，从1开始标记质数，从第一个质数k=2开始，标记所有小于n的1\*k, 2\*k,3\*k,4\*k的数。下一个未被标记的数一定是质数，然后再用这个数每局非素数。以此类推。

class Solution {

    public int countPrimes(int n) {

        if(n==0){return 0;}

        if(n==1){return 0;}

        int count=0;

        boolean[] helper=new boolean[n];

        helper[0]=true;

        helper[1]=true;

        for(int i=1;i<n;i++){

            if(helper[i]==false){

                count+=1;

                for(int j=2\*i;j<n;j+=i){

                    helper[j]=true;

                }

            }

        }

        return count;

    }

}

Java判断字符串相等要用s.equals(b)! 直接==是判断地址是否一致(String是个类)。

Shuffle 算法设计：

要么直接用Collections.shuffle()，要么自己写<https://juejin.cn/post/6974945439929860127>

从第0位开始，在当前idx和末尾中间生成随机数，然后swap(idx,rand)，以此类推。直到最后一位。 很简单，相当于随机挑出来1个之后，再从剩下的随机挑下一个。。以此类推,只不过节省空间放在了原位。

**注意int 和long 的范围！时刻考虑是否溢出！！！！ 看题目里给出的范围来确定用int 还是 long， 大概要是log（）还是多少的复杂度。**

例如 -2^32 – 2^32-1是 int的最大值！！！看到这个数 就使用long不用int！！！

Java优先队列用法：

注意 Integer要和 compare函数里的参数类型对应!

PriorityQueue<Integer>a =new PriorityQueue<Integer>(new Comparator<Integer>(){

Public int compare(Integer e1, Integer e2){

Return e2-e1;(队列返回最大值，降序)

Return e1-e2;(队列返回最小值，升序，默认)

}

})

位运算 ^ 可以用来判断两数相等！ A^A=0,

剑指offer 04 二维数组查找

二维数组从左到右和从上到下都是递增，则数组可以看成以右上角为根节点，左边比他小，下边比它大的一个二叉搜索树！！.

212 单词搜索II, 递归搜索中如何避免重复路径？？回溯法， 在递归调用前把当前节点抹去，然后在调用的返回前恢复，这样就相当于每次递归删除掉了本次操作，回溯。

    public boolean search\_word(int idx\_m,int idx\_n,String word,Set<String>set){

char cur\_char=board[idx\_m][idx\_n];

        borad[idx\_m][idx\_n]=’#’

        if(word.length()==0){return true;}

        char tmp=word.charAt(0);

        String next\_word=word.substring(1);

        boolean out=false;

        System.out.println(idx\_m+" "+idx\_n+" "+word);

        if( idx\_m<m-1 && board[idx\_m+1][idx\_n]==tmp){out=out ||search\_word(idx\_m+1,idx\_n,next\_word,set); }

        if( idx\_n<n-1 && board[idx\_m][idx\_n+1]==tmp){out=out ||search\_word(idx\_m,idx\_n+1,next\_word,set); }

        if( idx\_m>0 && board[idx\_m-1][idx\_n]==tmp){out=out ||search\_word(idx\_m-1,idx\_n,next\_word,set); }

        if( idx\_n>0 && board[idx\_m][idx\_n-1]==tmp){out=out ||search\_word(idx\_m,idx\_n-1,next\_word,set); }

        board[idx\_m][idx\_n]=cur\_char;

        return out;

    }

Java字符串比较 s.compareTo(b) 返回 s-b，小于0则s<b

快排模板：

Partition函数

首先，先写partition模板```

def partition(nums, left, right):

pivot = nums[left]#初始化一个待比较数据

i,j = left, right

while(i < j):

while(i<j and nums[j]>=pivot): #从后往前查找，直到找到一个比pivot更小的数

j-=1

nums[i] = nums[j] #将更小的数放入左边

while(i<j and nums[i]<=pivot): #从前往后找，直到找到一个比pivot更大的数

i+=1

nums[j] = nums[i] #将更大的数放入右边

#循环结束，i与j相等

nums[i] = pivot #待比较数据放入最终位置

return i #返回待比较数据最终位置

快速排序

复习一下快速排序：

#快速排序

def quicksort(nums, left, right):

if left < right:

index = partition(nums, left, right)

quicksort(nums, left, index-1)

quicksort(nums, index+1, right)

arr = [1,3,2,2,0]

quicksort(arr, 0, len(arr)-1)

print(arr)

Arrays.sort()和 collection.reverse() 等工具类要注意. Collection 是ArrayList<>()

Java 栈和队列

Stack<> =new Stach<>();，本质上是个列表，所以有empty方法。

Empty(),push(),pop(),peek(), 还有个search，返回1-based position，注意pop和peek空栈会报错，所以要先用empty判断下。

队列

Queue<>=new LinkedList<>();本质上是队列，

Offer(),poll(),peek(),空时返回null，所以判断可以用peek()!=null，

Add, remove, element,空时报错，

滑动窗口思想：

可重复元素，找出所有加起来等于target的组合，注意 [2,2,3]和[2,3,2]属于一种组合， 因此，递归回溯时，需要先把candidate 排序后，不允许选择比上次idx更小的元素，来避免这种重复。

但是如果考虑顺序，例如全排列问题，就需要一个used Set或者数组来记录已经用过的数，这个时候可以选择idx更小的元素。

class Solution {

    //经典递归回溯,算法，可重复和不可重复 两种写法注意下。

    //注意要先给candidates拍个序。

    //排序后，相当于每次从candidate中选一个，由于最终结果是无序的，为了避免重复，我们规定下一次选取的数字必须大于等于上次的数字，不能回头这样就能避免重复！！！由于candidate已经排好序，只需要下一次选取的idx大于等于上一次的就行。

    //什么时候使用 used 数组，什么时候使用 begin 变量

    /

// 有些朋友可能会疑惑什么时候使用 used 数组，什么时候使用 begin 变量。这里为大家简单总结一下：

// 排列问题，讲究顺序（即 [2, 2, 3] 与 [2, 3, 2] 视为不同列表时），需要记录哪些数字已经使用过，此时用 used 数组；

// 组合问题，不讲究顺序（即 [2, 2, 3] 与 [2, 3, 2] 视为相同列表时），需要按照某种顺序搜索，此时使用 begin 变量。

    public List<List<Integer>>out=new ArrayList<>();

    public List<List<Integer>> combinationSum(int[] candidates, int target) {

        Arrays.sort(candidates);

        List<Integer>combination=new ArrayList<>();

        helper(candidates,target,combination,0);

        return out;

    }

    public void helper(int[] candidates, int target, List<Integer>combination,int idx){

        if(target<0){

            return;

        }

        if(target==0){

            out.add(new ArrayList<Integer>(combination));

            return;

        }

        for(int i=idx;i<candidates.length;i++){

            combination.add(candidates[i]);

            helper(candidates,target-candidates[i],combination,i);

            combination.remove(combination.size()-1);

        }

    }

}