算法

基础

变量命名：索引 **i j k** 值 **val** 计数**cnt** 路径**p path** 访问位**vis** 常数 **MOD MAX** 坐标 **x y z** 数组大小 **m n**

TreeMap TreeSet 由红黑树实现有序map、set

```
int MAX=0x3f3f3f3f; //  最大整数 同时还能避免两数相加溢出int表示范围

int getIndex(int x,int y){
    return x*col+y;

String.charAt(idx); // String访问索引

Collection.sort(object,compertor);

Arrays.toString(arr);// 输出数组用于调试

// 数组、列表 之间的转换
Integer[] nums=lst.toArray(); //List转数组
List<Integer> list=nums.asList(); //数组转List
// 字符串、列表 之间的转换
char[] cs=str.toCharArray();//String转char数组
String s=String.valueOf(cs);// char数组转String

//linkedList实现了Deque接口 因此有push() poll() peek() getFirst() getLast()
addLast() addFirst() removeFirst() removeLast() removeLast() removeFirst()

// Character 常用方法
Character.isLetter();
Character.isDigit();
Character.isUpperCase(); Character.isLowerCase();
Character.toUpperCase(); Character.toLowerCase();
Character.isWhiteSpace();


// StringBuilder 常用方法
sb.insert(idx,ch)
sb.setCharAt(idx,ch) //一般使用char数组做修改 然后再由char数组转字符串
```

思想

有序 有序函数 最大值最小值 就要想二分 没有有序函数构造有序函数

当题目允许往任意方向移动时，考察的往往就不是 DP 了，而是图论。DP 问题是一类特殊的图论问题，DP 题虽然都属于图论范畴。但对于不是拓扑图的图论问题，我们无法使用 DP 求解。

递归算法的时间复杂度 = 递归深度 * 每次递归中的迭代次数

对于字符串序列

- 如果是单串 一维dp 和 二维dp i j对应从i..j
- 双串 二维dp 第一个串0..i 第二个串 0...j

回溯算法是自顶向下 DP算法是自底向上 本质上都是搜索

二叉树所有问题：1. 遍历一遍得出答案 对应回溯 2. 分解问题得出答案 对应动态规划

二叉树的所有问题，就是让你在前中后序位置注入巧妙的代码逻辑，去达到自己的目的。

二叉树后序 左右中 反向后续 右左中 === 逆序先序

递归问题：

1. 函数是干嘛的
2. 函数变量是什么
3. 得到的结果用来干什么

写递归算法的关键是要明确函数的「定义」是什么，然后相信这个定义，不要怀疑！利用这个定义推导最终结果，绝不要跳入递归的细节。

想 **该做什么** 和 **什么时候做** 前者是**递归定义问题**，后者是**前中后序问题**

DP问题：

1. 定义
2. 状态
3. 选择

问题规模的变化的导数的积分 等于 复杂度

模板

二分

```
int left=0,right=n-1;
while(left<=right){
    int mid=left+(right-left)/2;
    if(nums[mid] < target)
        left=mid+1;
    else if(nums[mid] > target)
        right=mid-1;
    else if(nums[mid] == target)
//      right=mid-1; 求左侧
//      left=mid+1;  求右侧
        return mid; //找值
}
//求左侧
//if(left>=n || nums[left]!=target) return -1;
```

```
//return left;

//求右侧
//if(right<0 || nums[right]!=target) return -1;
//return right;

return -1
```

求最大公因数

gcd 欧几里得 辗转相除法

原理：a 是大数 b是小数，大数、小数与a mod b 拥有一样的最大公因数d（证明：a=k*c+b -> c=a/c-b/c -> c是整数 除以d左右还是整数），而d一定小于等于a mod b

```
int gcd(int a, int b){
    return b==0 ? a : gcd(b,a%b);
}
```

快速排序

```
void quickSort(int[] nums,int lo,int hi){
    int k=partition(nums,lo,hi);
    quickSort(nums,lo,k-1);
    qucikSort(nums,k+1,hi);
}
int partition(int[] nums,int lo,int hi){
    if(lo==hi) return lo;
    int i=lo,j=hi+1;
    int v=nums[lo];
    while(true){
        while(nums[++i]<=v) if(i==hi) break;
        while(num[--j]>=v) if(j==lo) break;
        if(i>=j) break;
        swap(nums,i,j);
    }
    swap(nums,lo,j);
    return j;
}
int swap(int[] nums,int i,int j){
    int temp=nums[i];
    nums[i]=nums[j];
    nums[j]=temp;
}
```

归并排序

```
int n;
int temp=new int[n];
void sort(int[] nums,int lo,int hi){
    if(lo==hi) return;
    int mid=(lo+hi)/2;
    sort(nums,lo,mid);
    sort(nums,mid+1,hi);
    merge(nums,lo,mid,hi);
}
void merge(int nums[],int lo, int mid,int hi){
    int left=lo,right=mid+1;
    for(int i=lo;i<=hi;i++)
        temp[i]=nums[i];
    for(int k=lo;k<=hi;k++){
        if(left>mid) nums[k]=temp[right++];
        else if(right>hi) nums[k]=temp[left++];
        else if(temp[left]<=temp[right]) nums[k]=temp[left++];
        else if(temp[left]>temp[right])  nums[k]=temp[right++];
    }
}
```

翻转链表

```
//迭代 三指针
ListNode reverse(ListNode a) {
    ListNode pre, cur, nxt;
    pre = null; cur = a; nxt = a;
    while (cur != null) {
        // 顺序：下一个节点--前一个节点--中间节点--下一节点 Nxt放前面 避免cur更新后为null
访问cur.next越界
        nxt = cur.next;
        cur.next = pre;
        pre = cur;
        cur = nxt;
    }
    return pre;
}

//递归 神奇之处在于
//翻转之后 head的前驱节点就是head.next找到
ListNode reverse(ListNode head){
    if(head==null || head.next==null ) return head;
    ListNode last=reverse(head.next);
    head.next.next=head;
    head.next=null;
    return head;
}
```

洗牌

```
void shuffle(int[] nums) {
    int n = nums.length;
    Random rand = new Random();
    for (int i = 0 ; i < n; i++) {
        // 从 i 到最后随机选一个元素
        int r = i + rand.nextInt(n - i);
        swap(nums, i, r);
    }
}
```

矩阵快速幂

```
int[][] mul(int[][] a,int[][] b){
    int r=a.length,c=b[0].length,z=a[0].length;
    int[][] ans=new int[r][c];
    for(int i=0;i<r;i++)
        for(int j=0;j<c;j++)
            for(int k=0;k<z;k++)
                ans[i][j]+=a[i][k]*b[k][j];
    return ans;
}
// (b^n) * a
while(n>0){
    if((n&1)==1) a=mul(b,a);
    b=mul(b,b);
    n>>=1;
}
```

存图

```
// 邻接矩阵数组：w[a][b] = c 代表从 a 到 b 有权重为 c 的边
int[][] w = new int[N][N];

// 加边操作
void add(int a, int b, int c) {
    w[a][b] = c;
}

// 稀疏图 邻接矩阵 前向星
int[] he = new int[N], e = new int[M], ne = new int[M], w = new int[M];
int idx;

void add(int a, int b, int c) {
    e[idx] = b;
    ne[idx] = he[a];
    he[a] = idx;
    w[idx] = c;
    idx++;
}
```

```
// 类 存所有边
class Edge {
    // 代表从 a 到 b 有一条权重为 c 的边
    int a, b, c;
    Edge(int _a, int _b, int _c) {
        a = _a; b = _b; c = _c;
    }
}
```

最短路问题

n个顶点 m条边

Floyd 「多源汇最短路问题」 三重循环 枚举顺序 中间节点-起点-终点 -松弛操作　O（n3)

Dijkstra 「单源汇最短路问题」 每次找到「最短距离最小」且「未被更新」的点 t 标记点t　再用点 t 更新其他节点　复杂度 邻接矩阵 O(n2) 邻接表O(mn) 堆优化O(mlogn+n)：使用优先队列保存　《点和距离》

Bellman Ford 「单源点负权图求最短路」 每次过一遍所有边更新距离 O(mn)

并查集

```
int find(int a){
    if(p[a]!=a) p[a]=find(p[a]);
    return p[a];
}

int union(int a,int b){
    p[find(a)]=p[find(b)];
}

int query(int a,int b){
    return find(a)==find(b);
}
```

最小生成树

kruskal 所有边 排序 每次加最小边 并查集保证不出现回路 O(mlogm) 稀疏图

prim 从起点开始 每次找一个新的最近点 指导所有点找完 O(n2) 稠密图

字典树

```
int[][] tr=new int[100010][26];
int[] count=new int[100010];
int idx=0;

void insert(String s){
    char[] cs=s.toCharArray()
    int i=0;
    for(char c:cs){
        if(tr[i][c-'a']==0){
            tr[i][c-'a']=++idx;
        }
```

```
            i=tr[i][c-'a'];
        }
        count[i]++;
    }

    int query(String s){
        char[] cs=s.toCharArray()
        int i=0;
        for(char c:cs){
            if(tr[i][c-'a']==0){
                retrun 0;
            }
            else{
                i=tr[i][c-'a'];
            }
        }
        return count[i];
    }
```

树状数组

模版：单点修改 区间查询

区间修改 单点查询： 用树状数组维护差分数组的前缀和

区间修改 区间查询：建议用线段树

```
int[] tr;
int lowbit(int x){return x&-x;}
void add(int x,int v){
    for(int i=x;i<tr.length;i+=lowbit(i)) tr[i]+=v;
}
int ask(int x){
    int ans=0;
    for(int i=x;i!=0;i-=lowbit(i)) ans+=tr[i];
    return ans;
}
```

clarify

前序序列和中序序列构建二叉树

> 前序序列的第一个节点为根节点，通过该根节点可以将中序序列划分为左子树中序序列和右子树中序序列
>
> 可以通过左子树中序序列的个数来判断 左子树先序序列在哪里截至 即可划分出左子树先序序列和右子树先序序列
>
> 可以通过哈希表对中序序列中的节点进行查询优化，这样可以每次以O(1)复杂度查询根节点在中序序列的位置

Labuladong

链表

环形链表循环的位置、相交链表、三指针翻转链表、递归翻转链表、翻转前K个（存后继）、翻转第m到n个、K个一组翻转链表、判断回文链表（后序遍历 / 翻转+快慢指针）

二分查找

求左/右侧区间

快速排序

快排哨兵节点 左右指针 退出条件 最后调换 洗牌

动态规划

斐波那契数列，找零钱，最长递增子序列，最长回文字序列，目标和 最小路径和

01背包 子集背包 完全背包

编辑距离

单调栈

下一个更大的数字、每日温度、删除重复字母

数据结构

LRU least recently used 哈希链表

二叉树

114. 二叉树展开为链表（中等）

116. 填充每个节点的下一个右侧节点指针（中等）

236. 二叉树的最近公共祖先（中等）