# 初始化参数

## 1- 数组

1 标准初始化

int[] arr1 = new int[]{1,2,3,4,5};

2 初始化包含1234的数组

int[] arr2 = new int {1,2,3,4};

3 长度为4的数组, 所有元素均为 0

int[] arr3 = new int[4]

4 二维数组 4行 3列

int[][] arr2 = new int[4][3];

## 2- 初始化最大值最小值 – 要补2和3, 4和5的区别

int num = Integer.MAX\_VALUE;

int num = Integer.MIN\_VALUE;

## 2- 列表

LinkedList<int> list = new LinkedList<>();

## 3- 列表

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

//嵌套列表初始化 //添加元素

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

res.add(temp);

//嵌套列表初始化 //添加元素

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

res.add(new ArrayList<Integer>(temp));

## 3- 链表 括号中是( value, 指向下一个节点)

ListNode res = new ListNode(0, nextNode);

## 4- 哈希map

HashMap<Integer,Integer> map = new HashMap();

// map中有内部类Entry, entrySet()方法, 获取map中所有

for(Map.Entry<Integer,Integer> entry : map.entrySet()){

System.out.println("key= "+ key + " and value= " + map.get(key));

}

// 包含keySey() 获取map中所有key值的集合

Set<String> set = map.keySet();

for (String string : set) {

System.out.println(s + "," + map.get(s));

}

## 5- map

Map<Integer, Integer> map = new HashMap<>();

## 6- 哈希set

// hashset有无序性和不可重复性

HashSet<Integer> set = new HashSet<>();

## 7- 元组

// 元祖用来存放不同的元素集合

Tuple<String,Integer,String> person = new Tuple<>("张三",18,"男");

## 8- 队列 和双端队列

//一头进,一头出, 先进先出顺序

Queue<String> que = new LinkedList<>();

//允许两头都进，两头都出，

Deque<Integer> deque = new ArrayDeque<>();

## 9- 堆栈

//后进先出顺序

Stack<int> stack = new Stack<>();

## 10- 二叉树

TreeNode root = new TreeNode(rootValue);

## 11- StringBuilder

// StringBuilder 和Buffer 用来面对字符串经常需要改动的情况,

// String S1 = "abc”; S1 + "def” 时, string 会快

// String S1 = "abc”; String S2 = "def”; S1 + S2 时, builder就会更快

StringBuilder subString = new StringBuilder();

## 12- 优先堆

// 默认是小顶堆, 最上元素是最小的

PriorityQueue<Integer> minHeap = new PriorityQueue<>();

PriorityQueue<Integer> maxHeap = new PriorityQueue<Integer>(10,new Comparator<Integer>()); //大顶堆，容量10

## 

# 常用方法(循环,排序,最大值.etc)

## 1- 排序

Arrays.sort(arr); //默认从小到大进行sort（）排序

## 2- 链表排序

Collections.sort(list);

## 3- 最大值,最小值

int side = Math.min(a,b);

int side = Math.max(a,b);

## 4- 数值转换

1. int 转换 list

//int n 转成list

List<Integer> resList = Arrays.asList(n);

# 常用模板 (二分法, 递归,回溯 岛屿问题沉没land)

## 1. 78题回溯模板

public static void collectSubset(int[] nums, int start, List<Integer> subList, List<List<Integer>> res) {

if (start > nums.length) {return;}

// 每次都将当前子集添加到res中, 从空开始

res.add(new ArrayList<Integer>(subList));

//回溯模板

for (int i = start; i < nums.length; i++) {

// 将 i 添加到list中,(从索引0开始)

subList.add(nums[i]);

//回溯遍历从当前元素到数组结尾的所有可能性

collectSubset(nums, i + 1, subList, res);

// 删除最后一个元素

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

}

}

}

## 2. 34题二分查找模板

private int binarySearch(int[] nums, int target, boolean isSearchingLeft) {

int left = 0;

int right = nums.length - 1;

//idx为符合target的边界值索引

int idx = -1;

//创建循环, 当左边界大于有边界时循环完全部可能性, 思路类似双指针

while (left <= right) {

int mid = left + (right - left) / 2;

if (nums[mid] > target) {

right = mid - 1;

} else if (nums[mid] < target) {

left = mid + 1;

}

else {

idx = mid;

if (isSearchingLeft) {

right = mid - 1;

} else {

left = mid + 1;

}

}

}

return idx;

}

}

## 3. 230题二叉树第K小元素

public int kthSmallest(TreeNode root, int k) {

if (root == null) {return k;}

Stack<TreeNode> stack = new Stack<TreeNode>();

TreeNode node = root;

while (node != null || !stack.isEmpty()) {

while (node != null) {

stack.push(node);

node = node.left;

}

if (!stack.isEmpty()) {

k--;

node = stack.pop();

if (k == 0) {return node.val;}

node = node.right;

}

}

return k;

}

## 4. 215题数组第K大元素 (不能用排序)

public int findKthLargest(int[] nums, int k) {

PriorityQueue<Integer> minHeap = new PriorityQueue<>();

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

minHeap.offer(nums[i]);

}

for (int i = k; i < nums.length; i++) {

if (nums[i] > minHeap.peek()) {

minHeap.poll();

minHeap.offer(nums[i]);

}

}

return minHeap.peek();

}

## 5. 102题二叉树层序遍历

while (!que.isEmpty()) {

TreeNode temp = que.poll();

take++;

sublist.add(temp.val);

if (temp.left != null) {que.add(temp.left);}

if (temp.right != null) {que.add(temp.right);}

if (take == push) {

result.add(new ArrayList<>(sublist));

sublist.clear();

take = 0;

push = que.size();

}

}

## 6. 200题岛屿问题 (沉没陆地模板)

//沉没陆地模板

private void DFSMarking(char[][] grid, int i, int j) {

//判断i和j符合矩阵大小(不超边界), 和当前元素是否为0(海水)

if (i < 0 || j < 0 || i >= n || j >= m || grid[i][j] == '0')

return;

//如果不为0, 就沉没陆地, 然后递归调用, 遍历上下左右所有相邻的陆地进行沉没

grid[i][j] = '0';

DFSMarking(grid, i + 1, j);

DFSMarking(grid, i - 1, j);

DFSMarking(grid, i, j + 1);

DFSMarking(grid, i, j - 1);

}

## 6. 42题积水问题 (双指针)

public int trap(int[] height) {

int left = 0;

int right = height.length - 1;

// 初始化左右边界,边界内才可以积水

int leftH = height[left];

int rightH = height[right];

int rain = 0;

while (left < right) {

// 对比左右边界移动短的一边

if (leftH < rightH) {

left++;

// 移动后, 取当前高度和指针指向高度的最大值(木桶边界)

leftH = Math.max(leftH, height[left]);

// 边界高度减去当前高度就是当前索引处可积水容量

rain += leftH - height[left];

}

//移动右指针,原理同上

else {

right--;

rightH = Math.max(height[right], rightH);

rain += rightH - height[right];

}

}

return rain;

}