<https://algo.itcharge.cn/>

<https://chat.qwen.ai/>

# 常用数据结构&API

## java.util.TreeSet

特点：

1. 有序（默认自动升序）
2. 去重集合
3. 不允许null
4. 插入、删除、查找时间复杂度：O(log n)

基本操作

|  |  |
| --- | --- |
| add(E e) | 添加元素 |
| remove(Object o) | 删除元素 |
| contains(Object o) | 判断是否包含元素 |
| size() | 返回元素个数 |
| isEmpty() | 是否为空 |

Set<Integer> set\_1 = new TreeSet<>();  
set\_1.add(3);  
set\_1.add(1);  
set\_1.add(2);  
System.*out*.println(set\_1); // [1, 2, 3]  
  
Set<Integer> set\_2 = new TreeSet<>(new Comparator<Integer>() {  
 @Override  
 public int compare(Integer o1, Integer o2) {  
 return o2 - o1;  
 }  
});

// 或者

// Set<Integer> set\_2 = new TreeSet<>((o1, o2) -> o2 - o1);

// TreeSet<Integer> set\_2 = new TreeSet<>((a, b) -> b.compareTo(a));  
set\_2.add(1);  
set\_2.add(3);  
set\_2.add(2);  
System.*out*.println(set\_2); // [3, 2, 1]

System.out.println("集合: " + set); // [5, 10, 20]

System.out.println("大小: " + set.size()); // 3

System.out.println("包含10? " + set.contains(10)); // true

set.remove(20);

排序相关方法

|  |  |
| --- | --- |
| first() | 返回最小元素 |
| last() | 返回最大元素 |
| headSet(E toElement) | 返回小于toElement的子集（不包含 toElement） |
| tailSet(E fromElement) | 返回大于等于fromElement的子集 |
| subSet(E from, E to) | 返回[from, to)范围内的子集 |

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

set.addAll(Arrays.asList(5, 10, 15, 20, 25));

System.out.println("最小值: " + set.first()); // 5

System.out.println("最大值: " + set.last()); // 25

System.out.println("小于15: " + set.headSet(15)); // [5, 10]

System.out.println("大于等于15: " + set.tailSet(15)); // [15, 20, 25]

System.out.println("10到20之间: " + set.subSet(10, 20)); // [10, 15]

**实用方法**

|  |  |
| --- | --- |
| lower(E e) | 返回小于e的最大元素，不存在则返回null |
| floor(E e) | 返回小于等于e的最大元素 |
| higher(E e) | 返回大于e的最小元素 |
| ceiling(E e) | 返回大于等于e的最小元素 |
| pollFirst() | 获取并移除最小元素 |
| pollLast() | 获取并移除最大元素 |

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

set.addAll(Arrays.asList(5, 10, 15, 20));

System.out.println("lower(12): " + set.lower(12)); // 10

System.out.println("floor(10): " + set.floor(10)); // 10

System.out.println("higher(12): " + set.higher(12)); // 15

System.out.println("ceiling(10): " + set.ceiling(10)); // 10

System.out.println("弹出最小: " + set.pollFirst()); // 5

System.out.println("当前集合: " + set); // [10, 15, 20]

System.out.println("弹出最大: " + set.pollLast()); // 20

System.out.println("当前集合: " + set); // [10, 15]

遍历方式

TreeSet<Integer> set = new TreeSet<>(Arrays.asList(3, 1, 4, 1, 5));

// 方式1：增强 for 循环

for (Integer num : set) {

System.out.print(num + " "); // 1 3 4 5

}

// 方式2：迭代器（升序）

Iterator<Integer> it = set.iterator();

while (it.hasNext()) {

System.out.print(it.next() + " ");

}

// 方式3：降序遍历

Iterator<Integer> descIt = set.descendingIterator();

while (descIt.hasNext()) {

System.out.print(descIt.next() + " "); // 5 4 3 1

}

# 二分查找

## 模板

int binarySearch(int[] nums, int target) {  
 int left = 0;  
 int right = nums.length - 1; // 注意  
 while(left <= right) {  
 int mid = left + (right - left) / 2;  
 if(nums[mid] == target)  
 return mid;  
 else if (nums[mid] < target)  
 left = mid + 1; // 注意  
 else if (nums[mid] > target)  
 right = mid - 1; // 注意  
 }  
 return -1;  
}  
  
int left\_bound(int[] nums, int target) {  
 int left = 0, right = nums.length - 1;  
 // 搜索区间为 [left, right]  
 while (left <= right) {  
 int mid = left + (right - left) / 2;  
 if (nums[mid] < target) {  
 // 搜索区间变为 [mid+1, right]  
 left = mid + 1;  
 } else if (nums[mid] > target) {  
 // 搜索区间变为 [left, mid-1]  
 right = mid - 1;  
 } else if (nums[mid] == target) {  
 // 收缩右侧边界  
 right = mid - 1;  
 }  
 }  
 // 检查出界情况  
 if (left >= nums.length || nums[left] != target)  
 return -1;  
 return left;  
}  
  
int right\_bound(int[] nums, int target) {  
 int left = 0, right = nums.length - 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) {  
 // 这⾥改成收缩左侧边界即可  
 left = mid + 1;  
 }  
 }  
 // 这⾥改为检查 right 越界的情况，⻅下图  
 if (right < 0 || nums[right] != target)  
 return -1;  
 return right;  
}

# 双指针

## 模板

# 滑动窗口

## 固定长度滑动窗口

## 不定长度滑动窗口

# 深度优先遍历（DFS）

# 广度优先遍历（BFS）

# 回溯算法

# 贪心算法

# 动态规划

# todo

冯唐的散文《一万次的春和景明》