栈

栈，先入后出的容器

https://docs.oracle.com/javase/8/docs/api/java/util/Stack.html

队列

先进先出排队

https://docs.oracle.com/javase/8/docs/api/java/util/Queue.html

双端队列

https://docs.oracle.com/javase/8/docs/api/java/util/Deque.html

题目

有效的括号

```java
class Solution {
    public boolean isValid(String s) {
        if(s.isEmpty()){
            return true;
        }
        Deque<Character> deque = new LinkedList<>();
        for(char c : s.toCharArray()){
            if(c == '('){
                deque.addFirst(')');
            }else if(c == '['){
                deque.addFirst(']');
            }else if(c == '{'){
                deque.addFirst('}');
            }else if(deque.isEmpty() || c != deque.removeFirst()){
                return false;
            }
        }

        return deque.isEmpty();
    }
}
```

最小栈

```java
class MinStack {
    private final Deque<Integer> deque = new LinkedList<>();
    private final Deque<Integer> minDeque = new LinkedList<>();
    /** initialize your data structure here. */
    public MinStack() {

    }

    public void push(int x) {
        deque.addFirst(x);
        if(minDeque.isEmpty()){
            minDeque.addFirst(x);
        }else{
            int top = minDeque.peekFirst();
            if(x<=top) minDeque.addFirst(x);
        }
    }

    public void pop() {
        int top = deque.removeFirst();
        int min = minDeque.peekFirst();
        if(top==min) minDeque.removeFirst();
    }

    public int top() {
       return deque.peekFirst();
    }

    public int getMin() {
       return minDeque.peekFirst();
    }
}

/**
 * Your MinStack object will be instantiated and called as such:
 * MinStack obj = new MinStack();
 * obj.push(x);
 * obj.pop();
 * int param_3 = obj.top();
 * int param_4 = obj.getMin();
 */
```

柱状图中最大的矩形

```
// 暴力
// for i->0,n-2
//  for j->i+1,n-1
//    (i,j) 最小高度, area
//    update max-area;

// 暴力2
// for i->0, n-1;
// find left bound, right bound
// area = height[i] * (right-left)
// update max-area

// stack
//

class Solution {
    public int largestRectangleArea(int[] heights) {
        Deque<Integer> deque = new LinkedList<>();
        deque.addFirst(-1);
        int maxArea = 0;
        for(int i=0; i<heights.length; i++){
            // 固定右边界
            while(deque.peekFirst()!=-1 && heights[deque.peekFirst()] >
heights[i]){
                int index = deque.removeFirst();
                int height = heights[index];
                maxArea = Math.max(maxArea, height*(i-deque.peekFirst()-1));
            }
            deque.addFirst(i);
        }
        while (deque.peekFirst() != -1) {
            int index = deque.removeFirst();
            int height = heights[index];
            maxArea = Math.max(maxArea, height*(heights.length-
deque.peekFirst()-1));
        }
        return maxArea;
    }
}
```

接雨水

```
class Solution {
    public int trap(int[] height) {
        Deque<Integer> deque = new LinkedList<>();
        int sum = 0;
        for(int i=0;i<height.length;i++){
            while(!deque.isEmpty()&&height[i]>height[deque.peekFirst()]){
                int curIdx = deque.removeFirst();
                // 如果栈顶元素一直相等，那么全都pop出去，只留第一个。
                while (!deque.isEmpty() && height[deque.peekFirst()] ==
height[curIdx]) {
                    deque.removeFirst();
                }
             if(!deque.isEmpty()){
                int stackTop = deque.peekFirst();
                int high = Math.min(height[stackTop], height[i]) -
height[curIdx];
                sum = sum + high * (i - stackTop-1);
              }
            }
            deque.addFirst(i);
        }
        return sum;
    }
}
```