/\*\*

\* 01背包问题 体积 价值 求最大体积下的最大价值

\* @param {Array} weights 物体体积数组

\* @param {Array} values 物品价值数组

\* @param {Number} W 最大体积数

\*/

function knapsack(weights, values, W) {

const n = weights.length;

const f = new Array(W + 1).fill(0);

for (let i = 0; i < n; i++) {

for (let j = W; j >= weights[i]; j--) {

f[j] = Math.max(f[j], f[j - weights[i]] + values[i]);

}

}

return f[W];

}

/\*\* 爬楼梯 猴子爬山 1或2

\* @param {number} n

\* @return {number}

\*/

const climbStairs = function(n) {

const sqrt\_5 = Math.sqrt(5);

const fib\_n = Math.pow((1 + sqrt\_5) / 2, n + 1) - Math.pow((1 - sqrt\_5) / 2,n + 1);

return Math.round(fib\_n / sqrt\_5);

};

/\*

\* 无重复字符的最大子串

\* \*/

const lengthOfLongestSubstring = function (str) {

if (str.length <= 1) {

return str.length

}

let left = 0;let right = 1;let max = 0;let temp

while (right < str.length) {

temp = str.slice(left, right)

if (temp.includes(str[right])) {

left++

continue

} else {

right++

}

max = Math.max(right - left, max)

}

return max

};

//以矩形左上角的点为基点，x为横坐标，y为纵坐标，w为矩形的宽度，h为矩形的高度

let recA = {x: 1, y: 6, w: 4, h: 4}

function getRect(recA, recB) { //求出两个矩形相交形成的矩形

let rectLeft = null,rectRight = null,rectTop = null,rectBottom = null;

if (recA.x < recB.x) {

rectLeft = recA

rectRight = recB

} else {

rectRight = recA

rectLeft = recB

}

if (rectLeft.x + rectLeft.w <= rectRight.x) {

return {x: -1, y: -1, w: -1, h: -1}

}

if (recA.y < recB.y) {

rectTop = recB

rectBottom = recA

} else {

rectTop = recA

rectBottom = recB

}

if (rectBottom.y + rectBottom.h <= rectTop.x) {

return {x: -1, y: -1, w: -1, h: -1}

}

w = (rectLeft.x + rectLeft.w) < (rectRight.x + rectRight.w) ? (rectLeft.x + rectLeft.w - rectRight.x) : (rectRight.x + rectRight.w - rectRight.x)

h= (rectBottom.y + rectBottom.h) < (rectTop.y + rectTop.h) ? (rectBottom.y + rectBottom.h - rectTop.y) : (rectTop.y + rectTop.h - rectTop.y)

return { // 返回相交的矩形 两两比较

x: rectRight.x,

y: rectBottom.y,

w,

H,

}

}