Oaní da Silva da Costa

Teste de Lógica

Aqui é a versão em pdf, para o arquivo .js acesse:

https://pastebin.com/bp0QXnhh

caso não funcione:

https://drive.google.com/file/d/1glG3F3xoKHbvafvdg6ej0lJyqqXPrLOU/view?usp=sharing

Primo

Um número primo é aquele que é divisível apenas por **um** e por **ele mesmo**.
Obs: O número 1 não é primo.
Escreva uma função **otimizada** que, dado um número inteiro positivo, retorne true se o número for primo ou false caso contrário, com o menor número de iterações possível.
Imprima o resultado em tela da seguinte forma:
*"O número **num** é primo. Número de iterações necessárias: **count**"*
ou
*"O número **num** não é primo. Número de iterações necessárias: **count**"*

Palíndromo

Palíndromo, do grego palin (novo) e dromo (percurso), é toda palavra ou frase que quando lida ao contrário, desconsiderando espaços e pontuações, possui o mesmo sentido. Ex.: "asa", "ovo", "A base do teto desaba".

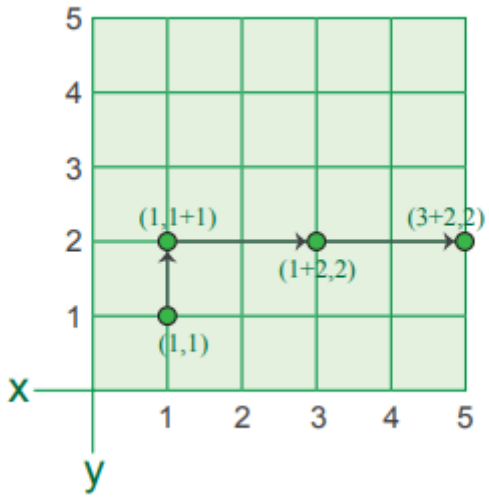
Escreva uma função que receba uma string como parâmetro e retorne true caso o valor dessa string seja um palíndromo ou false, caso contrário.

Movimento do bot

Existe um bot localizado em um par de coordenadas inteiras, (x, y). Ele pode ser movido para um outro par de coordenadas. Embora o bot possa se mover quantas vezes quiser, ele só pode fazer os dois tipos de movimentos a seguir:

1. Da posição (x, y) para a posição (x + y, y).
2. Da posição (x, y) para a posição (x, x + y).

Por exemplo, se o bot começa em (1, 1), ele pode fazer a seguinte sequência de movimentos: (1, 1) → (1, 2) → (3, 2) → (5, 2). Observe que o movimento sempre será para cima ou para a direita.



Escreva uma função que, dadas as coordenadas iniciais e finais, determine se o bot pode alcançar as coordenadas finais de acordo com as regras de movimento.

Descrição da função:

A função deve retornar true se o bot puder atingir seu objetivo, caso contrário, retorne false.

A função tem o(s) seguinte(s) parâmetro(s):

x1: valor inteiro, coordenada x inicial

y1: valor inteiro, coordenada y inicial

x2: valor inteiro, coordenada x final

y2: valor inteiro, coordenada y final

Solução:

```
/**
 * Oaní da Silva da Costa
 *
 * Teste de Lógica
 *
 * Escrito em Javascript usando Node
 * Foi pensado para ser aberto e executado em um arquivo só prontamente.
 *
 * Aqui é a versão em pdf, para o arquivo .js acesse:
 * ------------------------------------------------------
 * https://pastebin.com/bp0QXnhh
 * ------------------------------------------------------
 * caso não funcione:
 * ------------------------------------------------------
 *
https://drive.google.com/file/d/1glG3F3xoKHbvafvdg6ej0lJyqqXPrLOU/view?usp=sharing
 * ------------------------------------------------------
 */

const fs = require('fs'); // importa o modulo filesystem
const { DefaultDeserializer } = require('v8');
const prompt = require('prompt-sync')();; //importa o modulo prompt-sync
//o modulo prompt-sync deve ser instalado. Para isto: $ npm install prompt-sync
//No meu VScode não rodou, mas direto do terminal não tive problemas.

console.log("----------EX1----------");

/* Primo
* Um número primo é aquele que é divisível apenas por um e por ele mesmo.
* Obs: O número 1 não é primo.
*
* Escreva uma função otimizada que, dado um número inteiro positivo,
* retorne true se o número for primo ou false caso contrário,
* com o menor número de iterações possível.
* Imprima o resultado em tela da seguinte forma:
* “O número num é primo. Número de iterações necessárias: count”
* ou
* “O número num não é primo. Número de iterações necessárias: count”

Eu faria uma força bruta, simplesmente ir tentando para cada número se há resto para todos os divisores menores que ele.
```

```
Mas procurando vi que dá para otimizar isso e ele consegue achar em
poucas iterações.
Não é o mais otimizado possivel mas parece OK.

Material consultado:
https://www.knowprogram.com/c-programming/c-prime-number-using-function/
https://www.tutorialspoint.com/different-methods-to-find-prime-number-in-
python-program
https://en.wikipedia.org/wiki/Sieve_of_Atkin

Nesse link tem uma resposta fantastica sobre as otimizações: https:
//www.quora.com/Whats-the-best-algorithm-to-check-if-a-number-is-
prime/answer/Lavesh-Kaushik
Mas simplesmente copiar sem entender não seria legal. Fiz por mim mesmo
otimizando só um pouco de otimizações.

*/

function ehprimo(n) {
    let cont = 1;

    //não pode ser menor que 1
    if (n <= 1) {
        console.log("Não é primo");
        return false;
    }
    //caso de ser 2
    else if (n === 2) {
        console.log("É primo");
        console.log("Numero de iterações = " + cont);
        return true;
    } else if (n > 2 && n % 2 === 0) { //maior que 2 e não for par
        console.log("Não é primo");
        console.log("Numero de iterações = " + cont);
        return false;
    } else {
        // Só precisamos testar até a raiz de n. Explicação:
http://mathandmultimedia.com/2012/06/02/determining-primes-through-
square-root/
        for (let i = 3; i < Math.sqrt(n) + 1; i++) {
            if (n % i == 0) {
                console.log("Não é primo");
                console.log("Numero de iterações = " + cont);
                return false;
            }
            i++; //não precisa testar cada um, pode ir de 2 em 2
        }
        console.log("É primo");
        console.log("Numero de iterações = " + cont);
```

```
        return true;
    }
}

let n = prompt("Digite um número para eu checar se é primo: ");

console.log(n);
ehprimo(n);

console.log("----------EX2----------");

/*
* Palíndromo
* Palíndromo, do grego palin (novo) e dromo (percurso), é toda palavra ou frase que quando lida ao contrário,
* desconsiderando espaços e pontuações, possui o mesmo sentido. Ex.: “asa”, “ovo”, “A base do teto desaba”.
* Escreva uma função que receba uma string como parâmetro e retorne true caso o valor dessa string seja um palíndromo ou false,
* caso contrário.

A lógica é
    ou copiar -> inverter -> comparar com o original
    ou comparar cada letra com a posição (final-ela)
        asasa
        12345

        1 com 5, 2 com 4, 3 com 3

Como é JS, sabia que tinha algum metodo que invertia a string, e tem, porém eu encontrei em

https: //www.freecodecamp.org/news/two-ways-to-check-for-palindromes-in-javascript-64fea8191fd7/

que se a string passada for uma frase com espaços e letras maiusculas e minusculas
como "A man, a plan, a canal. Panama" é preciso tratar esses casos. Estão tratados como no link.

*/


let str = prompt("Digite uma string para checar se é um palindromo: ");

function checapalindromo(str) {
    // Step 1. Lowercase the string and use the RegExp to remove unwanted characters from it
    var re = /[\W_]/g; // para tirar tudo o que não for alfanumérico
```

```
    var lowRegStr = str.toLowerCase().replace(re, '');
    // tira o que não for alfanumerico e coloca em minusculo

    var reverseStr = lowRegStr.split('').reverse().join('');
    // cria um array com os caracteres separados em cada espaço, inverte
esse array, e depois junta em uma string denovo.

    return reverseStr === lowRegStr; // retorna true se forem iguais
}


if (checapalindromo(str)) {
    console.log("------------");
    console.log("é palindromo")
    console.log("------------");
} else {
    console.log("----------------");
    console.log("NAO é palindromo")
    console.log("----------------");
}


console.log("----------EX3----------");

/*
* Movimento do bot
* Existe um bot localizado em um par de coordenadas inteiras, (x, y). Ele
pode ser movido para um outro par de coordenadas.
* Embora o bot possa se mover quantas vezes quiser, ele só pode fazer os
dois tipos de movimentos a seguir:
* 1. Da posição (x, y) para a posição (x + y, y).
* 2. Da posição (x, y) para a posição (x, x + y).
* Por exemplo, se o bot começa em (1, 1), ele pode fazer a seguinte
sequência de movimentos: (1, 1) → (1, 2) → (3, 2) → (5, 2).
* Observe que o movimento sempre será para cima ou para a direita.
*
* Escreva uma função que, dadas as coordenadas iniciais e finais,
determine se o bot pode alcançar as coordenadas finais
* de acordo com as regras de movimento.
* Descrição da função:
* A função deve retornar true se o bot puder atingir seu objetivo, caso
contrário, retorne false.
* A função tem o(s) seguinte(s) parâmetro(s):
*
* x1: valor inteiro, coordenada x inicial
* y1: valor inteiro, coordenada y inicial
* x2: valor inteiro, coordenada x final
* y2: valor inteiro, coordenada y final
```

```
Após quebrar a cabeça (meu racicinio antigo está abaixo) vi que ha uma
solução muito muito mais elegante que a minha antiga
https://www.geeksforgeeks.org/check-possible-move-given-coordinate-
desired-coordinate/

Tentanto entender a solução vi que esse problema está relacionado com o
algoritmo de euclides
(para uma explicação visual: https: //www.youtube.com/watch?v=jb1VMGrxPWg
)

Então no final das contas, dadas as regras do robozinho, se os maximos
divisores comuns das coordenadas iniciais e finais
não forem iguais, não tem jeito dele chegar lá.

*/

/**
 * Algoritmo de euclides não otimizado(para uma explicação visual: https:
//www.youtube.com/watch?v=h86RzlyHfUE )
 */
function mdc(i, j) {
    if (i == j)
    return i; //se os numeros forem iguais, o MDC é um deles mesmo.

    if (i > j)
    return mdc(i - j, j); //o maximo divisor comum entre quaisquer dois
naturais é no minimo 1, a função sempre tem retorno.
    return mdc(i, j - i);
}

function checacaminho(x, y, a, b) {

    if (a < x || b < y) {
        return false;
    }

    return (mdc(x, y) == mdc(a, b));
}


let x = prompt("Informe x1: ");
let y = prompt("Informe y1: ");
let a = prompt("Informe x2: ");
let b = prompt("Informe y2: ");

if (checacaminho(x, y, a, b))
console.log("Consigo chegar la");
else
```

```
console.log("Não consigo chegar lá...");

/*

////////////////////////////////Raciocinio
Antigo////////////////////////////////

* "Escreva uma função que, dadas as coordenadas iniciais e finais,
* determine se o bot pode alcançar as coordenadas finais
* de acordo com as regras de movimento.""

No exercicio fala em ir para a direita ou ir para cima. Porém
teoricamente esse exercício é analogo a caminhar em uma
arvore binária. Por tanto ao invés dos movimentos direita e subir eu
nomeei os métodos como descer e subir esquerda
e descer e subir direita para pensar numa arvore.

A idéia é ir formando uma arvore e parar assim que conseguir chegar em
x2, y2 ou quando tudo for tentado
(isso acontece quando ele tenta "subir acima do nó raiz", ou seja, ele
"volta" o nó inicial, o que não pode e ai indica
que é impossivel chegar em x2, y2).

Algo tipo assim

1,1 <- digamos que serja a posicao inicial.
| \
|  \
|   \
|    \
1,2   2,1 <- digamos que queira vir para cá. Ela começa de 1,2, ai 1,3,
ai volta para 1,2, ai vai para 3,2, volta para 1,1, depois para em 2,1
| \
|  \
|   \
1,3   3,2 <- por exemplo aqui ele chegaria por dDir, checaria que não tem
filhos, e faria sDir.

*/

/*

var x1 = 1;
var y1 = 1;
var x2 = 1;
var y2 = 2;

function posicao(x1, y1) {
```

```
    this.x = x1;
    this.y = y1;
    this.tentativa = 0;
    this.descipela = 0;

    this.dDir = function () {
        // console.log("antes "+ this.x);
        this.x += this.y;
        // this.x = this.x + y;
        // console.log("depois "+ this.x);
    };

    this.dEsq = function () {
        this.y += this.x;
    };

    this.sEsq = function () {
        this.x -= this.y;
    };

    this.sDir = function () {
        this.y -= this.x;
    };

    //para ele lembrar se ja passou pelo nodo
    this.setTentativa = function (t) {
        this.tentativa = t;
    };

    //para ele lembrar de onde veio
    this.setDescipela = function (d) {
        this.tentativa = d;
    };
};

let pos = new posicao(x1, y1);

 */

/**
 * a cada nivel que desce na arvore marcara se ja foi para a esquerda
 * a cada volta muda o valor
 * chega está false (vai para a esquerda)
 * se voltar marca true e vai para a direita
 * no filho está false, vai para a esquerda
 * e assim até chegar numa folha onde ou termina a execução ou vai
subindo trocando tudo de novo de true para false até chegar na raiz
 * onde recomeça o processo agora na parte direita da arvore
 */
```

```
/*
// let movimento = [1]; //1- desça esquerda 2- desça direita 3- suba
direita
//A cada nivel ele registra o movimento que deve tomar.
let nivel = 0; //começa no nivel 0
// let descipela = [0]; //1- esquerda 2- direita

if ((x1 === 0 && x2 !== 0) && (y1 === 0 && y2 !== 0)) {
    return false; //caso especial. se o inicio for 0,0 e o destino não
for o próprio 0,0, nem precisa começar.
}


let count = 0;

console.log("Tentei:");

while (true) {

    count++;
    if (count > 15) {
        break;
    }
    console.log("Passo" + count + " x= " + pos.x + " y= " + pos.y);

    if ((pos.x < x1) || (pos.y < y1)) {
        console.log("NAO da para chegar...")
        break;
        // return false; //Não da para chegar de jeito nenhum (arvore
totalmente percorrida)
    }
    if ((pos.x === x2) && (pos.y === y2)) {
        console.log("Cheguei!")
        break;
        // return true; //chegou no destino
    }
    // movimento[nivel + 1] = 1; //cria a marcação de um novo nivel para
descer. A direção padrao do novo nivel é sempre 1

    //como nao está no destino, checa se não pode descer mais. Se não
puder, sobe
    //Esse "não poder descer" é estar em um local proibido, ele passou
das coordenadas e, ja que não pode voltar, tenta outra vez.
    if ((pos.x > x2) || (pos.y > y2) || (pos.tentativa == 2)) {
        if (pos.descipela === 1) { // ou seja, ele desceu pela esquerda,
1 é o padrão para um novo "nó"
            console.log("Subi pela direita")
```

```
            pos.sDir(); //e sobe pela direita
            // nivel--;
            pos.setTentativa(1); //para na proxima iteração ele descer
pela direita
            console.log("ccccccc nivel= " + nivel)

        } else if (descipela[nivel - 1] === 2) {
            console.log("Subi pela esquerda")
            pos.sEsq();
            // nivel--;
            pos.setTentativa(2); //Todas as tentativas para o nodo
        }
    } else if (pos.tentativa === 0) { //se puder, desce para a esquerda.
        console.log("desci esquerda");
        pos.descipela = 1; //para ele lembrar de onde desceu naquele
nivel
        // nivel++;
        movimento = false; //para ele lembrar que desceu pela esquerda
        pos.dEsq(); //ou seja, se a "plaquinha" estiver false, vá para a
esquerda. Se não, direita.
        console.log("eeeeeee nivel= " + nivel)
    } else if (pos.tentativa === 1) {
        console.log("desci direita")
        pos.descipela = 2; //para ele lembrar de onde desceu naquele
nivel
        // nivel++;
        pos.dDir();
        console.log("fffffff nivel= " + nivel)
    }


}

 */
```