```
1  /*
2      Thiago Issao Yasunaka           RA:103069
3      Otávio Hideki Gonçalves Kochi   RA:107635
4  */
5
6  const assert = require('assert')
7  const BRANCO = 'branco'
8  const CINZA = 'cinza'
9  const PRETO = 'preto'
10 const NUMBER_TESTS = [250, 500, 750, 1000, 1250, 1500, 1750, 2000]
11
12 const r = 0
13 const s = 1
14 const t = 2
15 const u = 3
16 const v = 4
17 const w = 5
18 const x = 6
19 const y = 7
20
21 const a = 0
22 const b = 1
23 const c = 2
24 const d = 3
25 const e = 4
26 const f = 5
27 const g = 6
28 const h = 7
29 const i = 8
30 const j = 9
31
32 const diametro = (G) => {
33     const s = Math.round(Math.random() * (G.length - 1))
34     const a = bfs(G, s).indexOf(bfs(G,s).reduce((number1, number2) =>
   Math.max(number1, number2)))
35     const b = bfs(G,a).reduce((number1, number2) => Math.max(number1,
   number2))
36     return b
37 }
38
39 const teste_diametro = () => {
40     const G = [
41         [b,c],                      //       a
42         [a],                        //     /   \
43         [a]                         //   b       c
44     ]
45     const G2 = [
46         [a]                         //       a
47     ]
48     const G3 = [
49         [b,c],                       //          a
50         [a,e,d],                     //         /  \
51         [a,f],                       //        b    c
52         [b],                         //       / \     \
53         [b],                         //      e   d     f
54         [c,g],                       //                 \
55         [f],                         //                  g
56     ]
57     const G4 = [                             //            a
58         [b,g],                               //           / \
```

```
        [a,c,d],
        [b,e],
        [b,f],
        [c],
        [d],
        [a,h],
        [g,i],
        [h,j],
        [i]
    ]
    const dis = diametro(G)
    const dis2 = diametro(G2)
    const dis3 = diametro(G3)
    const dis4 = diametro(G4)

    assert (dis == 2)
    assert (dis2 == 0)
    assert (dis3 == 5)
    assert (dis4 == 7)
}


const bfs = (G, s) => {
    let grafo = Object.assign({}, G)
    Object.keys(grafo).forEach( index => {
        let adj = G[index]
        grafo[index] = {
            'd': Number.POSITIVE_INFINITY,
            'pi': null,
            'cor': BRANCO,
            'adj':  adj
        }
    })
    grafo[s].d = 0
    grafo[s].pi = null
    grafo[s].cor = CINZA
    let fila = [] //Inicialização da fila
    fila.push(grafo[s])
    indexU = 0
    while (indexU != G.length) {
        let u = fila[indexU]
        indexU++
        u.adj.forEach( v => {
            if(grafo[v].cor === BRANCO) {
                grafo[v].d = u.d + 1
                grafo[v].pi = G.indexOf(u.adj)
                grafo[v].cor = CINZA
                fila.push(grafo[v])
            }
        })
        u.cor = PRETO
    }
    return Object.keys(grafo).map(index => grafo[index].d)
}

const teste_bfs = () => {
    const G = [
        [s,v],
        [r,w],
        [u,w,x,y],
```

```
//          b   g
//         / \   \
//        c   d   h
//       /     \   \
//      e       f   i
//                   \
//                    j
```

```
        [t,x,y],
        [r],
        [s,t,x],
        [w,t,u,y],
        [u,x]
    ]
    const G2 = [
        [s,v],                  //       r
        [r,t,u],                //      / \
        [w,s],                  //     v   s
        [s],                    //        / \
        [r],                    //       t   u
        [t]                     //      /
    ]                           //     w
    let d = bfs(G, 1)
    let d1 = bfs(G2, 0)

    assert (d[r] == 1)
    assert (d[s] == 0)
    assert (d[t] == 2)
    assert (d[u] == 3)
    assert (d[v] == 2)
    assert (d[w] == 1)
    assert (d[x] == 2)
    assert (d[y] == 3)

    assert(d1[r] == 0)
    assert(d1[s] == 1)
    assert(d1[t] == 2)
    assert(d1[u] == 2)
    assert(d1[v] == 1)
    assert(d1[w] == 3)

}

const numero_arestas = (G) => {
    let contador = 0;
    if(G.length > 1)
        for(let i = 0; i < G.length; i ++){
            for(let j = 0; j < G[i].length; j ++){
                if(G[i][j] <= i)
                    contador++;
            }
        }
    return contador;
}

const teste_aresta = () => {
    const G4 = [
        [b,c],                  //       a
        [a],                    //      /   \
        [a]                     //   b       c
    ]
    const G = [
        [a]
    ]
    const G2 = [
        [s,v],                  //       r
        [r,t,u],                //      / \
        [w,s],                  //     v   s
```

```
        [s],                        //       / \
        [r],                        //      t   u
        [t]                         //     /
    ]                               //    w
    const G3 = [
        [b,c],                          //          a
        [a,e,d],                        //         /  \
        [a,f],                          //        b    c
        [b],                            //       / \     \
        [b],                            //      e   d     f
        [c,g],                          //                 \
        [f],                            //                  g
    ]
    const are1 = numero_arestas(G)
    const are2 = numero_arestas(G2)
    const are3 = numero_arestas(G3)
    const are4 = numero_arestas(G4)
    assert(are1 == 0)
    assert(are2 == 5)
    assert(are3 == 6)
    assert(are4 == 2)
}

const random_tree_random_walk = n => {
    let grafo = Object.assign({}, Array.apply(null, Array(n)))
    Object.keys(grafo).forEach((index) => {
        grafo[index] = {visitado: false, adj:Array()}
    })
    let u = Math.round(Math.random() * (n - 1))
    grafo[u].visitado = true
    let arestas = 0
    while( arestas < n - 1){
        let v = Math.round(Math.random() * (n -1))
        if(!grafo[v].visitado){
            grafo[v].adj.push(u)
            grafo[u].adj.push(v)
            grafo[v].visitado = true
            arestas++
        }
        u = v
    }
    return Object.keys(grafo).map(index => grafo[index].adj)
}

const eh_arvore = G => {
    const arestas = numero_arestas(G)
    if(arestas != (G.length - 1)){
        return false
    }
    const distancias = bfs(G, 0)
    for(let i = 0; i < G.length; i ++){
        if(distancias[i] === Number.POSITIVE_INFINITY)
            return false
    }
    return true
}

//AQUI COMECA PRIM
const extract_min = (Q, G) => {
    let menor = Number.POSITIVE_INFINITY
```

```
    let index = -1
    let vertice
    Q.forEach( (v, i) => {
        if(G[v].chave < menor ){
            menor = G[v].chave
            index = i
            vertice = v
        }
    })
    Q.splice(index, 1)
    return vertice
}

const mst_prim = (G, w, r) => {
    Object.keys(G).forEach(u => {
        G[u] = {
            adj: G[u],
            chave: Number.POSITIVE_INFINITY,
            pi: null,
            pertenceQ: true
        }
    })
    G[r].chave = 0
    let Q = Object.keys(G)
    let A = []
    for(let i = 0; i<G.length; i++) A[i] = []
    while(Q.length != 0){
        const u = extract_min(Q, G)
        G[u].pertenceQ = false
        if(G[u].pi != null){
            A[G[u].pi].push(Number(u))
            A[u].push(Number(G[u].pi))
        }
        G[u].adj.forEach((v, index) => {
            if(G[v].pertenceQ === true && w[u][index] < G[v].chave){
                G[v].pi = u
                G[v].chave = w[u][index]
            }
        })
    }
    return A
}

const createEdges = (current, n) => {
    let e = Array()
    for(let i=0; i < n; i++)
        if(i != current) e.push(i)
    return e
}

const createWeights = (end) => {
    let w = Array()
    for(let i=0; i < end - 1; i++)
        w.push(Math.random())
    return w
}

const random_tree_prim = n => {
    const G = Array(n).fill(null)
    G.forEach((element,i) => G[i] = createEdges(i, n))
```

```
299     const w = Array(n).fill(null)
300     w.forEach((element, i) => w[i] = createWeights(n))
301     let u = Math.round(Math.random() * (n - 1))
302     return mst_prim(G, w, u)
303 }
304 
305 
306 const teste_mst_prim = () => {
307     const G = [
308         [b, h],         //a
309         [a, c, h],      //b
310         [b, d, i ,f],   //c
311         [c, e, f],      //d
312         [d, f],         //e
313         [c, d, e, g],   //f
314         [f, i, h],      //g
315         [a, b, i, g],   //h
316         [c, g, h],      //i
317     ]
318 
319     const w = [
320         [4, 8],
321         [4, 8, 11],
322         [8, 7, 2, 4],
323         [7, 9, 14],
324         [9, 10],
325         [4, 14, 10, 2],
326         [2, 6, 1],
327         [8, 11, 7, 1],
328         [2, 6, 7]
329     ]
330     const t = mst_prim(G, w, Math.round(Math.random()*(w.length - 1)))
331     assert(t[0].includes(1) || (t[0].includes(1) && t[0].includes(0)))
332     assert((t[1].includes(2) && t[1].includes(0)) || t[1].includes(0))
333     assert((t[2].includes(1) && t[2].includes(3) && t[2].includes(5) &&
t[2].includes(8)) || (t[2].includes(3) && t[2].includes(8)))
334     assert(t[3].includes(2) && t[3].includes(4))
335     assert(t[4].includes(3))
336     assert(t[5].includes(6) || (t[5].includes(2) && t[5].includes(6)))
337     assert((t[6].includes(5) && t[6].includes(7) && t[6].includes(8)) ||
(t[6].includes(5) && t[6].includes(7)))
338     assert((t[7].includes(0) && t[7].includes(6)) || t[7].includes(6))
339     assert((t[3].includes(2) && t[7].includes(6)) || t[3].includes(2))
340 }
341 
342 
343 // AQUI COMECA O KRUSKAL
344 const make_set = (v, adj) => {
345     return {
346         E: adj,
347         rank: 0,
348         p: v
349     }
350 }
351 
352 const union = (x, y, G) => {
353     link(find_set(G, x), find_set(G, y), G)
354 }
355 
356 const find_set = (G,v) => {
```

```
357     if(v != G[v].p) G[v].p = find_set(G, G[v].p)
358     return G[v].p
359 }
360 
361 const link = (x, y, G) => {
362     if(G[x].rank  > G[y].rank){
363         G[y].p = x
364     } else{
365         G[x].p = y
366         if(G[x].rank == G[y].rank){
367             G[y].rank += 1
368         }
369     }
370 }
371 
372 const mst_kruskal = (G, w) => {
373     let A = Array(G.length)
374     for(let i = 0; i<G.length; i++){
375         A[i] = []
376     }
377     Object.keys(G).forEach( v => G[v] = make_set(v, G[v]))
378     let arestaOrdenada = []
379     Object.keys(G).forEach(u => {
380         G[u].E.forEach(v => {
381             u = parseInt(u)
382             if(u > v){
383                 arestaOrdenada.push([w[u][G[u].E.indexOf(v)], u, v])
384             }
385         })
386     })
387     arestaOrdenada.sort((a,b) => {
388         return a[0] - b[0]
389     })
390     for(let i = 0; i < arestaOrdenada.length; i ++){
391         if(find_set(G, arestaOrdenada[i][1]) != find_set(G, arestaOrdenada[i]
    [2])){
392             A[[arestaOrdenada[i][1]]].push(arestaOrdenada[i][2])
393             A[[arestaOrdenada[i][2]]].push(arestaOrdenada[i][1])
394             union(arestaOrdenada[i][1], arestaOrdenada[i][2], G)
395         }
396     }
397     return A
398 }
399 
400 const teste_mst_kruskal = () => {
401 
402     const G = [
403         [b, h],          //a
404         [a, c, h],       //b
405         [b, d, i ,f],    //c
406         [c, e, f],       //d
407         [d, f],          //e
408         [c, d, e, g],    //f
409         [f, i, h],       //g
410         [a, b, i, g],    //h
411         [c, g, h],       //i
412     ]
413 
414     const w = [
415         [4, 8],
```

```
        [4, 8, 11],
        [8, 7, 2, 4],
        [7, 9, 14],
        [9, 10],
        [4, 14, 10, 2],
        [2, 6, 1],
        [8, 11, 7, 1],
        [2, 6, 7]
    ]
    const A = mst_kruskal(G, w)
        assert (A[0][0] == 1)
        assert (A[1][0] == 0)
        assert (A[1][1] == 2)
        assert (A[2][0] == 8)
        assert (A[2][1] == 5)
        assert (A[2][2] == 3)
        assert (A[2][3] == 1)
        assert (A[3][0] == 2)
        assert (A[3][1] == 4)
        assert (A[4][0] == 3)
        assert (A[5][0] == 6)
        assert (A[5][1] == 2)
        assert (A[6][0] == 7)
        assert (A[6][1] == 5)
        assert (A[7][0] == 6)
        assert (A[8][0] == 2)
}

const random_tree_kruskal = n => {
    const G = Array(n).fill(null)
    G.forEach((element,i) => G[i] = createEdges(i, n))

    const w = Array(n).fill(null)
    w.forEach((element, i) => w[i] = createWeights(n))

    return mst_kruskal(G, w)
}

const teste_arvore = randomTree => {
    const n = NUMBER_TESTS
    n.forEach(number => {
        let soma_diametro = 0
        for(let i=0; i<500; i++) {
            let G = randomTree(number)
            assert(eh_arvore(G))
            soma_diametro = soma_diametro + diametro(G)
        }
        let media = soma_diametro/500
        console.log(number + ' ' + media)
    })
}
/* TESTES */
console.log("Rodando Walk")
teste_arvore(random_tree_random_walk)
console.log("Rodando Prim")
teste_arvore(random_tree_prim)
console.log("Rodando Kruskal")
teste_arvore(random_tree_kruskal)

// teste_bfs()
```

```
476 // teste_diametro()
477 // teste_aresta()
478 // teste_mst_kruskal()
479 // teste_mst_prim()
```