Sugestões de resolução:

# 2018.01 – época normal

1.

["carro","auto","auto","autocarro","carro","carro","autocarro","auto"]

["carro","auto","auto1","autocarro","carro1","carro2","autocarro1","auto2"]

Map<String, Integer> dicionario = new HashMap<>();

List<String> resultado = new ArrayLista<>();

Iterar sobre a lista original

for (String palavra : palavras)

Procurar cada string no dicionário

dicionario.containsKey(palavra)

Se ainda não existir no dicionário:

- adicioná-la à lista resultado;

- adicioná-la ao dicionário com o valor 1

Se existir no dicionário:

- adicioná-la à lista resultado concatenada com o valor que possui no dicionário

- incrementar o seu valor no dicionário

dicionario.put(palavra, dicionario.get(palavra) + 1);

# 2018-02 recurso

List<String> maisSinonimos(Map<String,List<String>>mapSyn,Integer i)

Da mesma forma que no exercício anterior vamos precisar de um Map para ir armazenando as palavras que vão surgindo e registar o seu número.

Map<String, Integer> dicionario = new HashMap<>();

Map<String, List<String>> mapa = new HashMap<>();

mapa.put("alto", new ArrayList<>(

Arrays.asList("crescido", "comprido", "subido")

));

mapa.put("baixo", new ArrayList<>(

Arrays.asList("rasteiro", "fraco", "pequeno")

));

mapa.put("criança", new ArrayList<>(

Arrays.asList("infante", "jovem", "pequeno")

));

mapa.put("longo", new ArrayList<>(

Arrays.asList("demorado", "comprido", "lento")

));

mapa.put("vagaroso", new ArrayList<>(

Arrays.asList("lento", "demorado", "arrastado", "pequeno")

));

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **String** | **List<String>** | | | |
| **alto** | crescido | comprido | subido |  |
| **baixo** | rasteiro | fraco | pequeno |  |
| **criança** | infante | pequeno |  |  |
| **longo** | demorado | comprido | lento |  |
| **vagaroso** | lento | demorado | arrastado | pequeno |

Aqui precisamos de iterar sobre os elementos do Map inicial

E para cada um deles iterar sobre os elementos da lista de sinónimos:

Algo como:

static List<String> maisSinonimos(Map<String, List<String>> mapSyn, Integer i) {

for (Object palavra : mapSyn.keySet().toArray()) {

// Ex: palavra = ‘vagaroso

for (String sinonimo : mapSyn.get((String) palavra)) {

// sinónimos da palavra ‘vagaroso’: ‘lento’,’demorado’,’arrastado’,’pequeno’

// ….. aqui ir construindo o Map dicionario

}

}

// iterar sorbe o Map dicionario e criar uma lista resultados

}

Podemos usar um dicionario Map do tipo do exercicio anterior para ir registando cada sinonimo que surge e quantas vezes surge. Exactamente o mesmo que fizemos no exercício anterior.

No final percorrer este Map e adicionae a uma List aquelas palavras com frequência superior a ‘i’.

# 2018-09 recurso

List<Integer> abcissas = Arrays.asList(7, 1, 2, 3, 4, 5, 6, 10, 8, 10, 9);

List<Integer> ordenadas = Arrays.asList(1, 1, 2, 3, 4, 5, 6, 10, 8, 9, 10);

|  |  |
| --- | --- |
| abcissas | ordenadas |
| **x** | **y** |
| 7 | 1 |
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |
| 4 | 4 |
| 5 | 5 |
| 6 | 6 |
| 10 | 10 |
| 8 | 8 |
| 10 | 9 |
| 9 | 10 |

Aqui trata-se de verificar a distância entre todos os pares de ponto e ver qual a maior.

Para problemas do tipo, qual o maior, ou qual o menor, precisamos sempre de uma variável onde vamos guardando o maior (ou menor) valor encontrado até ao momento.

A inicialização desta variável é importante: se procuramos “o maior” devemos inicializá-la com o menor valor possível; se caso contrário procuramos “o menor” devemos inicializá-la com o maior valor possível.

Nota sobre este problema: a distância entre o pontos (a e b) é a mesma entre os pontos (b ,a) 😉

Isto quer dizer que precisamos de calcular a distância entre um dado ponto da lista os pontos seguintes (não os anteriores).

Algo como:

for (int i = 0; i < abcissas.size() - 1; i++) {

Integer x1 = abcissas.get(i);

Integer y1 = ordenadas.get(i);

for (int j = i + 1; j < abcissas.size(); j++) {

Integer x2 = abcissas.get(j);

Integer y2 = ordenadas.get(j);

// calcular a distância entre (x1,y1) e (x2,y2)

// e guardá.la caso seja a maior encontrada até ao momento

}

}

Pertunta: qual a razão dos dois pedaços de código assinalados?

> /\*\*  
>  \*  
>  \* @author david  
>  \*/  
> public class ExplicacoesInes {  
>   
>     public static boolean elementoFecho(String tag) {  
>         return tag.startsWith("</");  
>     }  
>   
>     public static boolean validaTags(String[] tags) {  
>   
>         Stack pilha = new Stack();  
>   
>         for (String tag : tags) {  
>   
>             if (pilha.isEmpty()) {  
>                 if (elementoFecho(tag)) {  
>                     return false;  
>                 } else {  
>                     pilha.push(tag);  
>                 }  
>             } else {  
>                 String head = (String) pilha.peek();  
>                 if (elementoFecho(tag)) {  
>                     if (tag.replace("/", "").equals(head)) {  
>                         pilha.pop();  
>                     } else {  
>                         return false;  
>                     }  
>                 } else {  
>                     pilha.push(tag);  
>                 }  
>             }  
>         }  
>   
>         return pilha.isEmpty();  
>     }  
>   
>     public static int mistery(int[] a) {  
>         int max = a[1] - a[0];  
>         for (int j = 2; j < a.length; j++) {  
>             for (int i = 0; i < j; i++) {  
>                 if (a[j] - a[i] > max) {  
>                     max = a[j] - a[i];  
>                 }  
>             }  
>         }  
>         return max;  
>     }  
>   
>     public static void main(String args[]) {  
>   
>         //String[] tags = new String[]{  
>         //    "<h1>", "</h1>", "<p>", "</p>", "</a>"};  
>         //System.out.println(validaTags(tags));  
>         //int[] vals = new int[]{3, 1, 4, 1, 5, 9, 2};  
>         //System.out.println(mistery(vals));  
>         //testComparable();  
>         /\*Map<String, List<String>> mapa = new HashMap<>();  
>         mapa.put("alto", new ArrayList<>(  
>                 Arrays.asList("crescido", "comprido", "subido")  
>         ));  
>         mapa.put("baixo", new ArrayList<>(  
>                 Arrays.asList("rasteiro", "fraco", "pequeno")  
>         ));  
>         mapa.put("criança", new ArrayList<>(  
>                 Arrays.asList("infante", "jovem", "pequeno")  
>         ));  
>         mapa.put("longo", new ArrayList<>(  
>                 Arrays.asList("demorado", "comprido", "lento")  
>         ));  
>         mapa.put("vagaroso", new ArrayList<>(  
>                 Arrays.asList("lento", "demorado", "arrastado", "pequeno")  
>         ));  
>   
>         for (String palavra : maisSinonimos(mapa, 2)) {  
>             System.out.println(palavra);  
>         }\*/  
>         List<Integer> abcissas = Arrays.asList(7, 1, 2, 3, 4, 5, 6, 10, 8, 10, 9);  
>         List<Integer> ordenadas = Arrays.asList(1, 1, 2, 3, 4, 5, 6, 10, 8, 9, 10);  
>   
>         System.out.println(EucliDistPoints(abcissas, ordenadas) - Math.sqrt(Math.pow(9, 2) \* 2));  
>     }  
>   
>     static Double distEuclid(Integer x1, Integer y1, Integer x2, Integer y2) {  
>         return Math.sqrt(Math.pow(x2 - x1, 2) + Math.pow(y2 - y1, 2));  
>     }  
>   
>     static public double EucliDistPoints(List<Integer> abcissas, List<Integer> ordenadas) {  
>         Double maxDist = Double.MIN\_VALUE;  
>         for (int i = 0; i < abcissas.size() - 1; i++) {  
>             Integer x1 = abcissas.get(i);  
>             Integer y1 = ordenadas.get(i);  
>             for (int j = i + 1; j < abcissas.size(); j++) {  
>                 Integer x2 = abcissas.get(j);  
>                 Integer y2 = ordenadas.get(j);  
>                 Double d = distEuclid(x1, y1, x2, y2);  
>                 if (d > maxDist) {  
>                     maxDist = d;  
>                 }  
>             }  
>         }  
>         return maxDist;  
>     }  
>   
>     static List<String> maisSinonimos(Map<String, List<String>> mapSyn, Integer i) {  
>   
>         Map<String, Integer> dicionario = new HashMap<>();  
>   
>         for (Object palavra : mapSyn.keySet().toArray()) {  
>             for (String sinonimo : mapSyn.get((String) palavra)) {  
>                 if (!dicionario.containsKey(sinonimo)) {  
>                     dicionario.put(sinonimo, 1);  
>                 } else {  
>                     dicionario.put(sinonimo, dicionario.get(sinonimo) + 1);  
>                 }  
>             }  
>         }  
>   
>         List<String> resultado = new ArrayList<>();  
>         for (Object sinonimo : dicionario.keySet().toArray()) {  
>             if (i < dicionario.get(sinonimo)) {  
>                 resultado.add((String) sinonimo);  
>             }  
>         }  
>         return resultado;  
>   
>     }  
>   
>     static void testComparable() {  
>   
>         System.out.println("\n\ntestComparable");  
>   
>         List<MyComparable> l = new ArrayList<>();  
>         l.add(new MyComparable(2, "A"));  
>         l.add(new MyComparable(2, "C"));  
>         l.add(new MyComparable(2, "B"));  
>         l.add(new MyComparable(1, "Z"));  
>         l.add(new MyComparable(1, "F"));  
>         l.add(new MyComparable(1, "A"));  
>   
>         System.out.println("= Sort =======");  
>   
>         // for tradicional  
>         //for (MyComparable item : l) {  
>         //    System.out.printf("%s\n", item.toString());  
>         //}  
>         // for com iterador  
>         for (Iterator<MyComparable> it = l.iterator(); it.hasNext();) {  
>             MyComparable item = it.next();  
>             System.out.printf("%s\n", item.toString());  
>         }  
>   
>         Collections.sort(l);  
>   
>         System.out.println("==============");  
>   
>         // solução funcional  
>         l.forEach((item) -> {  
>             System.out.printf("%s\n", item.toString());  
>         });  
>   
>     }  
>   
> }  
>

SETEMBRO 2019

//Exercicio 1

public static Map<Character, Double> meanWordss(String[] text) {

Map<Character, Double> map1 = new HashMap<>();

Map<Character, Double> map2 = new HashMap<>();

for (int i = 0; i < text.length; i++) {

Character c = text[i].charAt(0);

if (!map1.containsKey(c)) {

map1.put(c, 1.0);

map2.put(c, (double) text[i].length());

} else {

map1.put(c, map1.get(c) + 1);

map2.put(c, map2.get(c) + (double) text[i].length());

}

}

for (Character c1 : map1.keySet()) {

map1.put(c1, map2.get(c1) / map1.get(c1));

}

return map1;

}

//Exercicio 1 (outra resolução)

public static Map<Character, Double> meanWords(String[] text) {

Map<Character, Pair<Integer, Integer>> tabela = new HashMap<>();

for (String palavra : text) {

Character primeiro = palavra.toCharArray()[0];

if (!tabela.containsKey(primeiro)) {

tabela.put(primeiro, new Pair<Integer, Integer>(palavra.length(), 1));

// System.out.println(palavra.length()+" "+palavra);

} else {

Pair<Integer, Integer> valor = tabela.get(primeiro);

// System.out.println(palavra.length()+" "+palavra);

//System.out.println(valor);

tabela.put(primeiro, new Pair<Integer, Integer>(valor.getKey() + palavra.length(), valor.getValue() + 1));

}

}

Map<Character, Double> letras = new HashMap<>();

for (int i = 0; i < tabela.size(); i++) {

Pair<Integer, Integer> valor = tabela.get((char) tabela.keySet().toArray()[i]);

letras.put((char) tabela.keySet().toArray()[i], (0.0 + valor.getKey()) / valor.getValue());

}

return letras;

}

//Exercicio 2

// Complexidade O(n) - Deterministico

//[2,5,6,9] - faz a inversao [9,6,5,2]

public List<T> enigma(List<T> q) {

if (q.isEmpty()) {

return q;

}

T data = q.remove(0);

q = enigma(q);

q.add(data);

return q;

}

//Exercicio 3

// BST - Binary Search Tree AVL -Balanced Tree

public boolean isBalanced(Node node) {

int lh;

/\* for height of left subtree \*/

int rh;

/\* for height of right subtree \*/

/\* If tree is empty then return true \*/

if (node == null) {

return true;

}

/\* Get the height of left and right sub trees \*/

lh = height(node.left);

rh = height(node.right);

if (Math.abs(lh - rh) <= 1

&& isBalanced(node.left)

&& isBalanced(node.right)) {

return true;

}

/\* If we reach here then tree is not height-balanced \*/

return false;

}

// //Exercicio 4

public static <V, E> Map<V, List<V>> vertsnotReachable(Graph<V, E> g) {

Map<V, List<V>> map = new HashMap<>();

Iterator<Vertex<V, E>> allVerts;

allVerts = g.vertices().iterator();

while (allVerts.hasNext()) {

Vertex<V, E> v = allVerts.next();

Deque<Vertex<V, E>> a = GraphAlgorithms.BreadthFirstSearch(g, v);

Deque<Vertex<V, E>> list = new ArrayDeque<Vertex<V, E>>();

Iterator<Vertex<V, E>> allVerts1;

allVerts1 = g.vertices().iterator();

while (allVerts1.hasNext()) {

Deque<Vertex<V, E>> list1 = new ArrayDeque<Vertex<V, E>>();

Vertex<V, E> v1 = allVerts1.next();

if (!a.contains(v1)) {

list1.add(v1);

}

map.put((V) v, (List) list1);

}

}

return map;

}

// Exercicio 5

public int getIndElem(Integer value) {

for (int i = 0; i < heap.size(); i++) {

if (heap.get(i).getValue().equals(value)) {

return i;

} else {

return -1;

}

}

return 0;

RECURSO 2019

//Exercicio 1

private static double sumAllElements(List<Double> doubles) {

double soma = 0;

for (Double d : doubles) {

soma += d;

}

return soma;

}

public static Double packing(Double capac, List<Double> pesos, Map<Integer, LinkedList<Double>> paletes) {

int cont = 1;

for (int i = 0; i < pesos.size(); i++) {

for (int p = 0; p < pesos.size(); p++) {

if (!paletes.containsKey(cont)) {

paletes.put(cont, new LinkedList<>());

}

if (sumAllElements(paletes.get(cont)) + pesos.get(p) <= capac) {

//paletes.put(cont, paletes.get(cont).add(p));

paletes.get(cont).add(pesos.get(p));

pesos.remove(p);

p--;

}

}

i--;

cont++;

}

Double mean=0.0;

for(Integer i:paletes.keySet()){

mean+=sumAllElements(paletes.get(i));

}

mean=mean/paletes.size();

System.out.println(mean);

System.out.println(paletes);

return mean;

}

//Exercicio 2

public static int mistery(String tt, String pp) {

for (int i = 0; i <= tt.length() - pp.length(); i++) {

int j = 0;

while (j < pp.length() && tt.charAt(i + j) == pp.charAt(j)) {

j++;

}

if (j == pp.length()) {

return i;

}

}

return -1;

}

//Exerccio 3

public ArrayList<Integer> imprimirInversaTree() {

ArrayList<Integer> tree = new ArrayList<>();

printInversaTree(this.root(), this.height(), 0, tree);

return tree;

}

private void printInversaTree(Node<Integer> node, int height, int currentL, ArrayList<Integer> lst) {

if (node == null) {

return;

}

printInversaTree(node.getLeft(), height - 1, currentL, lst);

printInversaTree(node.getRight(), height - 1, currentL, lst);

lst.add(node.getElement());

}

// Exercicio 5

public static <K, V extends Comparable<V>> int NumbElemsLastLevel(HeapPriorityQueue<K, V> heap) {

int cont = 0;

ArrayList<V> heapArray = new ArrayList<>();

for (int i = 0; i < heap.size(); i++) {

// heapArray.add(heap[i]);

}

return 0;

}

NORMAL 2019

//Exercicio 1

public static boolean elementoFecho(String tag) {

return tag.startsWith("</");

}

public static boolean validaTags(String[] tags) {

Stack pilha = new Stack();

for (String tag : tags) {

if (pilha.isEmpty()) {

if (elementoFecho(tag)) {

return false;

} else {

pilha.push(tag);

}

} else {

String head = (String) pilha.peek();

if (elementoFecho(tag)) {

if (tag.replace("/", "").equals(head)) {

pilha.pop();

} else {

return false;

}

} else {

pilha.push(tag);

}

}

}

return pilha.isEmpty();

}

//Exercicio 2

public static int mistery(int[] a) {

int max = a[1] - a[0];

for (int j = 2; j < a.length; j++) {

for (int i = 0; i < j; i++) {

if (a[j] - a[i] > max) {

max = a[j] - a[i];

}

}

}

return max;

}

// static Double distEuclid(Integer x1, Integer y1, Integer x2, Integer y2) {

// return Math.sqrt(Math.pow(x2 - x1, 2) + Math.pow(y2 - y1, 2));

// }

//

// static public double EucliDistPoints(List<Integer> abcissas, List<Integer> ordenadas) {

// Double maxDist = Double.MIN\_VALUE;

// for (int i = 0; i < abcissas.size() - 1; i++) {

// Integer x1 = abcissas.get(i);

// Integer y1 = ordenadas.get(i);

// for (int j = i + 1; j < abcissas.size(); j++) {

// Integer x2 = abcissas.get(j);

// Integer y2 = ordenadas.get(j);

// Double d = distEuclid(x1, y1, x2, y2);

// if (d > maxDist) {

// maxDist = d;

// }

// }

// }

// return maxDist;

// }

// static void testComparable() {

//

// System.out.println("\n\ntestComparable");

//

// List<MyComparable> l = new ArrayList<>();

// l.add(new MyComparable(2, "A"));

// l.add(new MyComparable(2, "C"));

// l.add(new MyComparable(2, "B"));

// l.add(new MyComparable(1, "Z"));

// l.add(new MyComparable(1, "F"));

// l.add(new MyComparable(1, "A"));

//

// System.out.println("= Sort =======");

//

// // for tradicional

// //for (MyComparable item : l) {

// // System.out.printf("%s\n", item.toString());

// //}

// // for com iterador

// for (Iterator<MyComparable> it = l.iterator(); it.hasNext();) {

// MyComparable item = it.next();

// System.out.printf("%s\n", item.toString());

// }

//

// Collections.sort(l);

//

// System.out.println("==============");

//

// // solução funcional

// l.forEach((item) -> {

// System.out.printf("%s\n", item.toString());

// });

//

// }

}

SETEMBRO 2018

RECURSO 2018

Map<String, List<String>> mapa = new HashMap<>();

mapa.put("alto", new ArrayList<>(Arrays.asList("crescido", "comprido", "subido")));

mapa.put("baixo", new ArrayList<>(Arrays.asList("rasteiro", "fraco", "pequeno")));

mapa.put("criança", new ArrayList<>(Arrays.asList("infante", "jovem", "pequeno")));

mapa.put("longo", new ArrayList<>(Arrays.asList("demorado", "comprido", "lento")));

mapa.put("vagaroso", new ArrayList<>(Arrays.asList("lento", "demorado", "arrastado", "pequeno")));

List<String> lista = maisSinonimos(mapa, 1);

System.out.println(lista);

List<String> lista2 = maisSinonimos2(mapa, 2);

System.out.println(lista2);

}

//Exercicio 1

public static List<String> maisSinonimos(Map<String, List<String>> mapSyn, Integer i) {

Map<String, Integer> dicionario = new HashMap<>();

List<String> result = new ArrayList<>();

for (int chaveIndice = 0; chaveIndice < mapSyn.size(); chaveIndice++) {

String palavra = (String) mapSyn.keySet().toArray()[chaveIndice];

for (int indice = 0; indice < mapSyn.get(palavra).size(); indice++) {

String chavestr = mapSyn.get(palavra).get(indice);

if (!dicionario.containsKey(chavestr)) {

dicionario.put(chavestr, 1);

} else {

dicionario.put(chavestr, dicionario.get(chavestr) + 1);

}

}

}

for (int j = 0; j < dicionario.size(); j++) {

String palavra = (String) dicionario.keySet().toArray()[j];

if (dicionario.get(palavra) > i) {

result.add(palavra);

}

}

return result;

}

static List<String> maisSinonimos2(Map<String, List<String>> mapSyn, Integer i) {

Map<String, Integer> dicionario = new HashMap<>();

for (String palavra : mapSyn.keySet()) {

for (String sinonimo : mapSyn.get(palavra)) {

if (!dicionario.containsKey(sinonimo)) {

dicionario.put(sinonimo, 1);

} else {

dicionario.put(sinonimo, dicionario.get(sinonimo) + 1);

}

}

}

List<String> resultado = new ArrayList<>();

for (String sinonimo : dicionario.keySet()) {

if (i < dicionario.get(sinonimo)) {

resultado.add((String) sinonimo);

}

}

return resultado;

}

NORMAL 2018

public static void main (String [] args){

List<String> exer1 = new ArrayList<>();

List<String> resul = new ArrayList<>();

exer1.add("carro");

exer1.add("auto");

exer1.add("auto");

exer1.add("autocarro");

exer1.add("carro");

exer1.add("carro");

exer1.add("autocarro");

exer1.add("auto");

resul = renomear(exer1);

System.out.println(resul);

int x = mistery(10);

System.out.println(x);

}

//Exercicio 1

public static List<String> renomear (List<String> lst){

Map<String,Integer> dicionario = new HashMap<>();

List<String> resultado = new ArrayList<>();

for (int i =0; i<lst.size();i++){

if (!dicionario.containsKey(lst.get(i))){

resultado.add(lst.get(i));

dicionario.put(lst.get(i), 0);

}else{

int cont = dicionario.get(lst.get(i))+1;

resultado.add(lst.get(i)+" "+ cont);

dicionario.put(lst.get(i),cont);

}

}

return resultado;

}

//Exercicio 2

public static int mistery(int n) {

int i;

int j = 0;

int res = 0;

for (i = n / 2; j <= n; i++) {

for (j = 2; j <= n; j = j \* 2) {

res = res + n / 2;

return res;

}

}

return 0;

}