Cuarta Práctica Individual de ADDA

Memoria técnica del proyecto

Roberto Camino Bueno, grupo asignado número **3**

Firmado por Roberto Camino

Índice:

1. Código con la soluciones de los problemas.

- Problema 1, solución con PDR

Mi razonamiento es el siguiente: recorrer la lista 'números' dada en el enunciado con el índice 'posActual' hasta el final y para cada número voy a generar la alternativa True o False.
Será True cuando lo meta en la lista pequeña (lista 0) o será False en caso de ir a la lista grande (lista 1).
También usaré la propiedad 'sumal0' (suma de los elementos de la lista 0) y 'sumal1' (suma de los elementos de la lista 1) para condicionar que ambas listas sumen lo mismo y así al minimizar el tamaño de la lista pequeña, no quede vacía.

```
public class problema1_PDR implements ProblemaPDR<List<List<Integer>>, Boolean, problema1_PDR> {
                                    // ProblemaPDR<TipoSolucion,TipoAlter, clase>

    private static List<Integer> numeros; // lista dada

    Integer posActual; // indice en numeros
    Integer sumal0; //suma peque
    Integer sumal1; // suma grande

public problema1_PDR(int indice, int suma0, int suma1) {
    this.posActual = indice;
    this.sumal0 = suma0;
    this.sumal1 = suma1;
}

public static problema1_PDR create(int indice, int suma0, int suma1) {
    return new problema1_PDR(indice, suma0, suma1);
}

public static problema1_PDR create(List<Integer> ls_numeros) { //para inicializar el meto de arriba
    numeros = ls_numeros;
    return new problema1_PDR(0, 0, 0);

}

public Tipo getTipo() {
    return Tipo.Min;
}

public int size() {
    return numeros.size() - posActual;
}

public boolean esCasoBase() {
    return numeros.size() == posActual;
}

@Override
public Sp<Boolean> getSolucionParcialCasoBase() { //devuelve una solucion parcial si cumple la restriccion

    if (sumal0 != sumal1) {// si no son iguales descarta este subproblema
        return null;
    }
    return Sp.create(null, 0.0); //pero si son iguales es una solucion posible y crea el subprobelma (Boolean a null)
}
```

```java
public problema1_PDR getSubProblema(Boolean b) { // vamos creando subproblemas

    int posActualcopy = posActual; // 1º hacemos una copia para no machacar valores
    int sumal0copy = sumal0;
    int sumal1copy = sumal1;

    if (b == true) { // cuando la alternativa es true
        sumal0copy = sumal0copy + numeros.get(posActualcopy); // el nº entra en suma peque
    } else { //cuando  b == false
        sumal1copy = sumal1copy + numeros.get(posActualcopy); // el nº entra en suma grande
    }
    posActualcopy++;
    return new problema1_PDR(posActualcopy, sumal0copy, sumal1copy);
}

public Sp<Boolean> getSolucionParcialPorAlternativa(Boolean b, Sp<Boolean> ls) { // para la reconstruccion de alternativas
    int cont = 0;
    if (b == true) { // cuando b es true siginifica que he metido el elemento en la lista
        cont = 1;
    }
    return Sp.create(b, ls.propiedad + cont); // es para llevar la cuenta y poder minimizar
}

public List<Boolean> getAlternativas() { // como es boolean tiene 2 alternativas T/F
    List<Boolean> res = new ArrayList<>();
    res.add(true);
    res.add(false);

    return res;
}

@Override
public List<List<Integer>> getSolucionReconstruidaCasoBase(Sp<Boolean> sp) { // devuelve la estrucutra del dato
                                                                              // vacio porque luego se ira
                                                                              // llenando
    List<List<Integer>> res = new ArrayList<>();
    List<Integer> ls_0 = new ArrayList<>();
    List<Integer> ls_1 = new ArrayList<>();

    res.add(ls_0);
    res.add(ls_1);

    return res;
}

@Override
public List<List<Integer>> getSolucionReconstruidaCasoRecursivo(Sp<Boolean> alter_sp, List<List<Integer>> ls_solParcial) {
    // va reconstruyendo la solucion desde el final hasta el principio de la traza

    List<Integer> ls0 = ls_solParcial.get(0);
    List<Integer> ls1 = ls_solParcial.get(1);

    if (alter_sp.alternativa == true) {
        ls0.add(numeros.get(posActual)); // en lspeq entra el nº
    } else {
        ls1.add(numeros.get(posActual)); // en lsgrande entra el nº
    }
    return ls_solParcial;
}
```

- **Problema 1, solución con BT**

Mi razonamiento es el siguiente: recorrer la lista 'números' dada en el enunciado con el índice 'posActual' hasta el final y para cada número voy a generar la alternativa True o False.
Será True cuando lo meta en la lista pequeña (lista 0) o será False en caso de ir a la lista grande (lista 1). Para llevar actualizado los datos en los métodos avanza y retrocede voy a usar 'ls_0' y 'ls_1'.
También usaré la propiedad 'sumal0' (suma de los elementos de la lista 0) y 'sumal1' (suma de los elementos de la lista 1) para condicionar que ambas listas sumen lo mismo y así al minimizar el tamaño de la lista pequeña, no quede vacía.

```
public class estadoProblema1BT implements EstadoBT<List<List<Integer>>, Boolean, estadoProblema1BT> {

    private static List<Integer> numeros;
    // A diferencia de PDR, necesito aqui las listas:
    private List<Integer> ls_0 = new ArrayList<>();
    private List<Integer> ls_1 = new ArrayList<>();

    Integer sumal0; // suma peque
    Integer sumal1; // suma grande
    Integer posActual; // indice



private estadoProblema1BT(List<Integer> ls_02, List<Integer> ls_12, int indice, int suma0, int suma1) {
    this.ls_0 = ls_02;
    this.ls_1 = ls_12;
    this.posActual = indice;
    this.sumal0 = suma0;
    this.sumal1 = suma1;
}

public static estadoProblema1BT create(List<Integer> ls_0, List<Integer> ls_1, int indice, int suma0, int suma1) {
    return new estadoProblema1BT(ls_0, ls_1, indice, suma0, suma1);
}

public static estadoProblema1BT create(List<Integer> ls_numeros) { // para inicializar el create anterior
    numeros = ls_numeros;
    return new estadoProblema1BT(new ArrayList<>(), new ArrayList<>(), 0, 0, 0);

}

public estadoProblema1BT getEstadoInicial() {
    return estadoProblema1BT.create(ls_0, ls_1, 0, 0, 0);
}
```

```java
public estadoProblema1BT getEstadoInicial() {
    return estadoProblema1BT.create(ls_0, ls_1, 0, 0, 0);
}

public Tipo getTipo() { //problema de minimizar
    return EstadoBT.Tipo.Min;
}

public Double getObjetivo() { //Necesario para que mi ls_0 sea minimizada y ademas no quede vacía
    Integer suma_n = numeros.stream().mapToInt(i -> i.intValue()).sum();
    Double res = (double) suma_n;
    //Utilizo el siguiente 'if' para que ls_0 no quede vacio
    if (sumal0 == suma_n / 2 && sumal1 == suma_n / 2) { // tambien vale sumal0 == sumal1 pero es menos optimo
        res = (double) ls_0.size(); //minimiza
    }
    return res;
}

public int size() {
    return numeros.size() - posActual;
}

public boolean esCasoBase() {
    return numeros.size() == posActual;
}

public estadoProblema1BT avanza(Boolean a) {
    int posActualcopy = posActual; // 1º hacemos una copia para no machacar valores
    int sumal0copy = sumal0;
    int sumal1copy = sumal1;
    List<Integer> ls_0copy = ls_0;
    List<Integer> ls_1copy = ls_1;

    if (a == true) { // cuando la alternativa es true
        sumal0copy = sumal0copy + numeros.get(posActualcopy); // el nº se suma en lista peque
        ls_0copy.add(numeros.get(posActualcopy)); // y entra en ls peque
    } else { // cuando b == false
        sumal1copy = sumal1copy + numeros.get(posActualcopy); // el nº se suma en lista grande
        ls_1copy.add(numeros.get(posActualcopy)); // y entra en ls grande
    }
    posActualcopy++;
    return new estadoProblema1BT(ls_0copy, ls_1copy, posActualcopy, sumal0copy, sumal1copy);
}
```

```java
public estadoProblema1BT retrocede(Boolean a) { // al reves que avanza, tiene que retroceder
    int posActualcopy = posActual; // Recordemos 1º hacer una copia para no machacar valores
    int sumal0copy = sumal0;
    int sumal1copy = sumal1;
    List<Integer> ls_0copy = ls_0;
    List<Integer> ls_1copy = ls_1;

    posActualcopy--;
    if (a == true) {
        sumal0copy = sumal0copy - numeros.get(posActualcopy);
        ls_0copy.remove(numeros.get(posActualcopy));
    } else {
        sumal1copy = sumal1copy - numeros.get(posActualcopy);
        ls_1copy.remove(numeros.get(posActualcopy));

    }
    return new estadoProblema1BT(ls_0copy, ls_1copy, posActualcopy, sumal0copy, sumal1copy);
}

public List<Boolean> getAlternativas() { // Alternativas T/F
    List<Boolean> res = new ArrayList<>();
    res.add(true);
    res.add(false);

    return res;
}

public List<List<Integer>> getSolucion() { //Hace copias de las listas y las añade a la lista solucion
    List<List<Integer>> lsContenedor = new ArrayList<>();
    List<Integer> ls_0_new = new ArrayList<>();
    List<Integer> ls_1_new = new ArrayList<>();

    for (int i = 0; i < ls_0.size(); i++) {
        ls_0_new.add(ls_0.get(i));

    }
    for (int j = 0; j < ls_1.size(); j++) {
        ls_1_new.add(ls_1.get(j));
    }
    lsContenedor.add(ls_0_new);
    lsContenedor.add(ls_1_new);

    return lsContenedor;

}
```

- **Problema 1, solución con GV**

Mi razonamiento es el mismo que en PDR pero con los cambios que conlleva hacerlo mediante Grafos Virtuales.
He creado una clase para las aristas, 'Problema1Edge', que genera las aristas del grafo y minimiza. También he creado la clase para los vértices, 'Problema1Vertex' donde implemento los métodos que generan nuevos vértices del grafo a partir del vértice estado inicial donde 'posActual', 'sumal0' y 'sumal1' se inicializan a 0.
En la clase test he usado un Predicate en vez de estado final donde le indico cuando quiero parar de generar vecinos, sin heurística.

Explicación del error: La solución suma los elementos en la lista grande y minimiza la lista pequeña hasta dejarla vacía porque debería pasarle como condición de parada que el estado final o el predicate fuese " sumal0==suma de números/2 && sumal1==suma de números/2" pero entonces no recorrería la lista números hasta el final (y si le añado a ese Boolean que llegue hasta el final, me devuelve la misma solución que antes).

Clase para las aristas:

```
public class Problema1Edge extends SimpleEdge<Problema1Vertex> {

    public static Problema1Edge of(Problema1Vertex v1, Problema1Vertex v2, Boolean a) { //con Integer a seria weight = 1.0 * a
        Double weight = 0.0;
        if (a) {
            weight = 1.0; // si fuese maximizar pondria -1.0
        }
        return new Problema1Edge(v1, v2, weight, a);
    }

    public Boolean a;

    private Problema1Edge(Problema1Vertex c1, Problema1Vertex c2, double weight, Boolean a) {
        super(c1, c2, weight);
        this.a = a;
    }

    public Problema1Edge(Problema1Vertex c1, Problema1Vertex c2) {
        super(c1, c2);

    }

}
```

Clase para los vértices:

```
public class Problema1Vertex extends ActionVirtualVertex<Problema1Vertex, Problema1Edge, Boolean> {

    // Propiedades individuales -> prop. de cada vertice
    private static List<Integer> numeros; // lista dada
    private List<Integer> ls_0 = new ArrayList<>();
    private List<Integer> ls_1 = new ArrayList<>();

    Integer posActual; // indice en numeros
    Integer sumal0; // suma peque
    Integer sumal1; // suma grande

    public Problema1Vertex(int indice, int suma0, int suma1) {
        this.posActual = indice;
        this.sumal0 = suma0;
        this.sumal1 = suma1;
    }

 public static Problema1Vertex create(int indice, int suma0, int suma1) {
     return new Problema1Vertex(indice, suma0, suma1);
 }

 public static Problema1Vertex create(List<Integer> ls_numeros) { // para inicializar el create anterior
     numeros = ls_numeros;
     return new Problema1Vertex(0, 0, 0);

 }

 public boolean isValid() {
     return 0 <= posActual && posActual <= numeros.size();
 }

 protected Problema1Vertex neighbor(Boolean a) {     // Similar a getSubproblema
     int posActualcopy = posActual; // 1º hacemos una copia para no machacar valores
     int sumal0copy = sumal0;
     int sumal1copy = sumal1;

     if (a == true) { // cuando la alternativa es true
         sumal0copy = sumal0copy + numeros.get(posActualcopy); // el nº entra en suma peque
     } else { // cuando b == false
         sumal1copy = sumal1copy + numeros.get(posActualcopy); // el nº entra en suma grande
     }
     posActualcopy++;
     return new Problema1Vertex(posActualcopy, sumal0copy, sumal1copy);
 }

 protected List<Boolean> actions() { // similar a getAlternativas
     List<Boolean> res = new ArrayList<>();
     res.add(true);
     res.add(false);

     return res;
 }
```

```
protected Problema1Vertex getThis() { // opcional, vertice actual
    return this;
}

protected Problema1Edge getEdge(Boolean a) { // devuelve una arista entre mis vertice actual y el siguiente
    return Problema1Edge.of(this, neighbor(a), a);
}

// La finalidad de este metodo optativo es para convertir [T,F,T...] en List<List<Integer>> res
public static List<List<Integer>> getSolucion(List<Problema1Edge> sol) {
    List<List<Integer>> lsContenedor = new ArrayList<>();
    List<Integer> ls_0 = new ArrayList<>();
    List<Integer> ls_1 = new ArrayList<>();
    lsContenedor.add(ls_0);
    lsContenedor.add(ls_1);

    for (int i = 0; i < sol.size(); i++) {
        if (sol.get(i).a == true) { // Cuando el elemento de la lista es a=T
            lsContenedor.get(0).add(numeros.get(i));
        } else { // a=F
            lsContenedor.get(1).add(numeros.get(i));
        }
    }
    return lsContenedor;
}
```

- **Problema 4, solución con PD**

Mi razonamiento es el siguiente: recorrer el string que me dan en el enunciado al cual he llamado ‘cadena’ y devolver una lista con los Strings que son palíndromos a la cual he llamado ‘subcadenas’.
Entonces utilizo los índices ‘posActual’ y ‘posFinal’ para recorrer el String ‘cadena’ y poder ir guardando las subcadenas que son palíndromas gracias a la propiedad ‘str’ que acumula los caracteres de la propiedad char ‘c’. De manera que cada elemento char lo voy guardando en ‘str’ y cuando sea palíndromo lo meto en mi lista de subcadenas palíndromos.
También pretendo minimizar el tamaño de ‘str’ para tener los palíndromos del menor tamaño posible. Las alternativas son True cuando ‘str’ es palíndromo y False cuando no lo sea y deba acumularse.

```java
public class problema4PD implements ProblemaPD<List<String>, Boolean, problema4PD> {

    private static String cadena; // cadena dato de entrada
    List<String> subcadenas = new ArrayList<>(); // Solucion: lista de palindromos
    // indices [i, j) como problema de matrices; [0,n)
    Integer posActual; // i
    Integer posFinal; // j
    String str;
    char c;

    public problema4PD(List<String> subcadena, Integer i, Integer j, String s) {
        this.subcadenas = subcadena;
        this.posActual = i;
        this.posFinal = j;
        this.str = s;

    }

    public static problema4PD create(List<String> subcadena, Integer i, Integer j, String s) {
        return new problema4PD(subcadena, i, j, s);
    }

public static problema4PD create(String cadenaDato) { // para actualizar el create anterior
    cadena = cadenaDato;
    return new problema4PD(new ArrayList<>(), 0, cadena.length(), "");

}

@Override
public Tipo getTipo() {
    return Tipo.Min;
}

@Override
public int size() {
    return posFinal - posActual;
}

public boolean esCasoBase() {
    // return esPalindromo(cadena.subSequence(posActual, posFinal));
    return esPalindromo(str);
}

@Override
public Sp<Boolean> getSolucionParcialCasoBase() {
    return Sp.create(null, 0.0); // crea el subprobelma
}
```

```java
public problema4PD getSubProblema(Boolean a, int np) { //
    int posActualcopy = posActual;
    List<String> subcadenascopy = subcadenas; // Solucion: lista de palindromos

    c = cadena.charAt(posActual);
    char c_copy = c;
    StringBuilder str_build = new StringBuilder();
    str_build.append(c_copy);
    str = str_build.toString(); // ahora c es string str

    problema4PD r = null;
    if (a == true) {
        subcadenascopy.add(str);
        r = problema4PD.create(subcadenascopy, posActualcopy++, posFinal, "");
    } else {
        r = problema4PD.create(subcadenascopy, posActualcopy++, posFinal, str); // str se acumula
    }
    return r;
}

public Sp<Boolean> getSolucionParcialPorAlternativa(Boolean a, List<Sp<Boolean>> ls) {
    int cont = 0;
    if (a == true) { // cuando b es true siginifica que he metido el elemento en la lista
        cont = 1;
    }
    return Sp.create(a, ls.get(0).propiedad + cont); // es para llevar la cuenta y poder minimizar

}

public List<Boolean> getAlternativas() {
    List<Boolean> res = new ArrayList<>();
    res.add(true);
    res.add(false);

    return res;
}

public int getNumeroSubProblemas(Boolean a) {
    return a.equals(false) ? 1 : 2;

}

public List<String> getSolucionReconstruidaCasoBase(Sp<Boolean> sp) {
    List<String> res = new ArrayList<>();
    String s = "";

    res.add(s);

    return res;
}

public List<String> getSolucionReconstruidaCasoRecursivo(Sp<Boolean> sp, List<List<String>> ls) {

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

    if (sp.alternativa == true) {
        res = ls.get(0);
    } else {
        res = ls.get(1);
    }
    return res;
}
```

2. Volcado de pantalla con los resultados obtenidos en las pruebas realizadas.

- **Problema 1, solución con PDR**

```
public class test_problema1_PDR {
    public static void main(String[] args) {
        AlgoritmoPD.isRandomize = false;

        List<Integer> numeros = List.of(1, 3, 1, 1, 2, 5, 8, 10, 6, 11);

        problema1_PDR pdr = problema1_PDR.create(numeros);
        var alg = AlgoritmoPD.createPDR(pdr);
        alg.ejecuta();

        System.out.println("\n Problema 1 resuelto por PDR\n");
        System.out.println(" Lista dato: " + numeros + "\n");
        if (alg.getSolucion() == null) {
            System.out.println(" Sin solucion");
        } else {
            System.out.println(" La solucion es: " + alg.getSolucion());
            int suma10 = alg.getSolucion().get(0).stream().mapToInt(i -> i.intValue()).sum();
            int suma11 = alg.getSolucion().get(1).stream().mapToInt(i -> i.intValue()).sum();
            System.out.println("\n La suma de la lista pequeña es: " + suma10);
            System.out.println(" La suma de la lista grande es: " + suma11);

        }
    }
}
```

```
Problema 1 resuelto por PDR

Lista dato: [1, 3, 1, 1, 2, 5, 8, 10, 6, 11]

La solucion es: [[11, 10, 3], [6, 8, 5, 2, 1, 1, 1]]

La suma de la lista pequeña es: 24
La suma de la lista grande es: 24
```

- **Problema 1, solución con BT**

```java
public class testProblema1BT {

    public static void main(String[] args) {

        AlgoritmoBT.isRandomize = false;
        AlgoritmoBT.numeroDeSoluciones = 4;

        List<Integer> numeros = List.of(1, 3, 1, 1, 2, 5, 8, 10, 6, 11);

        estadoProblema1BT estadoInicial = estadoProblema1BT.create(numeros);
        var alg = AlgoritmoBT.create(estadoInicial);

        alg.ejecuta();
        System.out.println(" Problema 1 resuelto por BT\n");
        System.out.println(" Lista dato: " + numeros + "\n");
        if (alg.getSolucion() == null) {
            System.out.println(" Sin solucion");
        } else {
            System.out.println(" La solucion es: " + alg.getSolucion());
            int suma10 = alg.getSolucion().get(0).stream().mapToInt(i -> i.intValue()).sum();
            int suma11 = alg.getSolucion().get(1).stream().mapToInt(i -> i.intValue()).sum();
            System.out.println("\n La suma de la lista pequeña es: " + suma10);
            System.out.println(" La suma de la lista grande es: " + suma11);
        }
    }
}
```

```
Problema 1 resuelto por BT

Lista dato: [1, 3, 1, 1, 2, 5, 8, 10, 6, 11]

La solucion es: [[8, 10, 6], [3, 1, 1, 1, 2, 5, 11]]

La suma de la lista pequeña es: 24
La suma de la lista grande es: 24
```

- **Problema 1, solución con GV**

```
public static void main(String[] args) {
    List<Integer> numeros = List.of(1, 3, 1, 1, 2, 5, 8, 10, 6, 11);
    Integer suma_n = numeros.stream().mapToInt(i -> i.intValue()).sum();
    Problema1Vertex estadoInicial = Problema1Vertex.create(numeros);

    Problema1Vertex estadoFinal = Problema1Vertex.create(numeros.size(), suma_n / 2, suma_n / 2);

    System.out.println(" Problema 1 resuelto por GV\n");
    System.out.println(" Lista dato: " + numeros + "\n");
    Integer tam = numeros.size();
    Predicate<Problema1Vertex> objetivo = (Problema1Vertex v) -> v.posActual == tam;
    // Predicate<Problema1Vertex> objetivo = (Problema1Vertex v)-> v.sumal0==suma_n/2 && v.sumal1==suma_n/2;
    // PredicateHeuristic<Problema1Vertex> predicateHeuristic = (x,p)->x.voraz(p);
    AStarGraph<Problema1Vertex, Problema1Edge> graph = AStarSimpleVirtualGraph.of(x -> x.getEdgeWeight());
    AStarAlgorithm<Problema1Vertex, Problema1Edge> alg = AStarAlgorithm.of(graph, estadoInicial, objetivo, null); // heuristica
                                                                                                                   // null
    List<Problema1Edge> lista_aristas = alg.getPath().getEdgeList(); // devuelve lista [T,F,T...]
    System.out.println(lista_aristas);

    List<List<Integer>> lsContendor = Problema1Vertex.getSolucion(lista_aristas);
    System.out.println(" Solucion :" + lsContendor);
}
```

Traza del resultado:

```
 Problema 1 resuelto por GV

 Lista dato: [1, 3, 1, 1, 2, 5, 8, 10, 6, 11]

[( posActual=0, sumal0=0, sumal1=0]
, posActual=1, sumal0=0, sumal1=1]
), ( posActual=1, sumal0=0, sumal1=1]
, posActual=2, sumal0=0, sumal1=4]
), ( posActual=2, sumal0=0, sumal1=4]
, posActual=3, sumal0=0, sumal1=5]
), ( posActual=3, sumal0=0, sumal1=5]
, posActual=4, sumal0=0, sumal1=6]
), ( posActual=4, sumal0=0, sumal1=6]
, posActual=5, sumal0=0, sumal1=8]
), ( posActual=5, sumal0=0, sumal1=8]
, posActual=6, sumal0=0, sumal1=13]
), ( posActual=6, sumal0=0, sumal1=13]
, posActual=7, sumal0=0, sumal1=21]
), ( posActual=7, sumal0=0, sumal1=21]
, posActual=8, sumal0=0, sumal1=31]
), ( posActual=8, sumal0=0, sumal1=31]
, posActual=9, sumal0=0, sumal1=37]
), ( posActual=9, sumal0=0, sumal1=37]
, posActual=10, sumal0=0, sumal1=48]
)]
 Solucion :[[], [1, 3, 1, 1, 2, 5, 8, 10, 6, 11]]
```

- **Problema 4, solución con PD**

```java
public class testProblema4PD {
    public static void main(String[] args) {
        AlgoritmoPD.isRandomize = false;

        String cadena = "ababbbabbababa";

        problema4PD pd = problema4PD.create(cadena);
        var alg = AlgoritmoPD.createPD(pd);
        alg.ejecuta();

        System.out.println(" Problema 1 resuelto por PD\n");
        System.out.println(" Cadena dato: " + cadena + "\n");
        System.out.println();

        if (alg.getSolucion() == null) {
            System.out.println(" Sin solucion");
        } else {
            System.out.println(" Solucion: " + alg.getSolucion());
            System.out.println(" Cantidad de palindromos: " + alg.getSolucion().size());
        }
    }

}
```

```
Problema 1 resuelto por PD

Cadena dato: ababbbabbababa


Solucion: []
```