

ENTREGA 2 - PROYECTO

**LUIS FELIPE AYALA URQUIZA
NICOLÁS ESTEBAN BAYONA ORDOÑEZ
JUAN SEBASTIÁN HERRERA GUAITERO**



Pontificia Universidad
JAVERIANA
Bogotá

**ANDREA DEL PILAR RUEDA OLARTE
PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERÍA
ESTRUCTURAS DE DATOS
BOGOTÁ D.C
22 DE ABRIL DE 2021**

Índice

1. Descripción del programa
 - 1.1. Descripción de entradas
 - 1.2. Descripción de salidas
 - 1.3. Descripción de condiciones
 - 1.4. Descripción de condiciones para los procesos auxiliares

2. TAD'S
 - 2.1. TAD Palabra
 - 2.1.1. Conjunto mínimo de datos
 - 2.1.2. Comportamiento (Operaciones)
 - 2.2. TAD Diccionario
 - 2.2.1. Conjunto mínimo de datos
 - 2.2.2. Comportamiento (Operaciones)
 - 2.3. TAD NodoClass
 - 2.3.1. Conjunto mínimo de datos
 - 2.3.2. Comportamiento (Operaciones)
 - 2.4. TAD TreeClass
 - 2.4.1. Conjunto mínimo de datos
 - 2.4.2. Comportamiento (Operaciones)
 - 2.5. TAD Tree
 - 2.5.1. Conjunto mínimo de datos
 - 2.5.2. Comportamiento (Operaciones)
 - 2.6. TAD ScrabbleClass
 - 2.6.1. Conjunto mínimo de datos
 - 2.6.2. Comportamiento (Operaciones)

3. Diagramas y dibujos

4. Diagrama de relación de TAD's

Descripción del funcionamiento del programa

1. Inicia la ejecución del programa
2. El sistema muestra un carácter “\$”
3. El sistema espera a que el usuario ingrese una instrucción
4. El sistema espera a que el usuario finalice de ingresar el comando que desea ejecutar y pulse la tecla enter
5. El sistema analiza la entrada y decide a qué función debería dirigirse la entrada (separada por el carácter ‘ ‘ como comando y argumento(opcional))
6. El repite indefinidamente las instrucciones (2-5) hasta que el usuario ingrese la palabra “salir”, allí el sistema decide que debe de llamar al método “exit” y termina la ejecución del programa.

Descripción de entradas:

Las entradas deben coincidir con el formato [comando] [argumento] para que el sistema funcione bien. No obstante, el sistema está diseñado teniendo en consideración casos en los que esto no necesariamente se cumpla, para cumplir con este fin, el sistema tiene varios mecanismos para asegurar su correcto funcionamiento en todo momento.

Descripción de salidas:

El sistema retorna las salidas de los diferentes comandos que puede ejecutar, para la entrega 2 serían:

1. inicializar [diccionario.txt]
POSIBLES SALIDAS
 - (Diccionario ya inicializado): El diccionario ya ha sido iniciado.
 - (Archivo no existe): El archivo no existe o no puede ser leído.
 - (Resultado exitoso): El diccionario se ha iniciado correctamente.
2. iniciar_inverso [diccionario.txt]
POSIBLES SALIDAS:
 - (Diccionario ya inicializado): El diccionario inverso ya ha sido iniciado.
 - (Archivo no existe): El archivo diccionario no existe o no puede ser leído.
 - (Resultado exitoso): El diccionario se ha iniciado correctamente.
3. puntaje [palabra]
POSIBLES SALIDAS:
 - (Palabra no existe): La palabra no existe en el diccionario.
 - (Letras inválidas): La palabra contiene símbolos inválidos.
 - (Resultado exitoso): La palabra tiene un puntaje de [puntaje2]

4. salir

5. iniciar_arbol[diccionario.txt]

POSIBLES SALIDAS

- (Árbol ya inicializado) El árbol del diccionario ya ha sido inicializado.
- (Archivo no existe): El archivo diccionario.txt no existe o no puede ser leído.
- (Resultado exitoso): El diccionario se ha iniciado correctamente.

6. iniciar_arbol_inverso [diccionario.txt]

POSIBLES SALIDAS

- (Árbol ya inicializado): El diccionario ya ha sido iniciado.
- (Archivo no existe): El archivo no existe o no puede ser leído.
- (Resultado exitoso): El diccionario se ha iniciado correctamente.

7. palabras_por_prefijo [prefijo]

POSIBLES SALIDAS

- (Prefijo inválido): Prefijo *prefijo* no pudo encontrarse en el diccionario
- (Resultado exitoso): Las palabras que inician con este *prefijo* son:

8. palabras_por_sufijo [sufijo]

POSIBLES SALIDAS

- (Sufijo inválido) Sufijo *sufijo* no pudo encontrarse en el diccionario .
- (Resultado exitoso): Las palabras que terminan con este *sufijo* son:

→ Si el usuario ingresa un comando diferente, pueden ocurrir dos cosas:

1. Que el comando indicado sea uno de los comandos que hasta el momento no se han implementado, en cuyo caso se mostrará la ayuda del comando especificado.
2. Que el usuario haya escrito algo mal, en cuyo caso se mostrará un mensaje de comando inválido.

Descripción de condiciones:

Para el procedimiento principal no debería de cumplirse ninguna condición específica, puesto que el sistema verifica primero que pueda asegurar su correcto funcionamiento con la entrada ingresada por el usuario para poder otorgar una respuesta correcta para la petición que se realiza.

Descripción de condiciones para los procesos auxiliares:

Para los procedimientos auxiliares no debe de cumplirse ninguna condición específica pues, la función “decide” se encarga de verificar que los argumentos puedan llegar sin complicaciones a las funciones necesarias.

TAD Palabra

Conjunto mínimo de datos:

- word, cadena de caracteres, representa la palabra almacenada por este objeto de tipo Palabra.

Comportamiento (Operaciones):

- check_character(line), verifica que todas las letras de la palabra sean alfabéticas, retorna el resultado de esta operación.
- inverse_characters(line), invierte los caracteres de una palabra, retorna el resultado de esta operación.
- Palabra(word, order), define la manera en la que se crea un objeto de tipo Palabra.
- getWord(), retorna la palabra almacenada por el objeto de tipo Palabra.
- invertOrder(), invierte la palabra almacenada por este objeto de tipo Palabra, retorna el resultado de realizar esta operación.
- suffixInWord(suffix), retorna si una palabra comienza con un prefijo indicado.
- prefixInWord(prefix), retorna si una palabra termina con un sufijo indicado.

TAD Diccionario

Conjunto mínimo de datos:

- lista_palabras, lista de objetos de tipo Palabra, representa todas las palabras que hay en el diccionario.
- file_name, cadena de caracteres, representa el nombre del archivo del cual se leyeron las palabras que contiene el diccionario.

Comportamiento (Operaciones):

- Diccionario(), define la manera en la que se inicializa un objeto de tipo Diccionario.
- to_string(), retorna una cadena de caracteres con todas las palabras contenidas en el diccionario.
- setFile_name(archive_name), actualiza el valor del campo file_name con la data pasada como parámetro.
- add_word(word), añade una palabra a la lista de palabras del diccionario.
- checkWord(word), busca si una palabra específica existe en el diccionario, retorna el resultado de esta operación.
- clear(), limpia la lista de palabras (lista_palabras).
- size(), retorna si la lista de palabras (lista_palabras) está vacía.
- getFile_name(), retorna el nombre del archivo usado para llenar el diccionario (file_name)
- wordsByPrefix(prefix), retorna una lista con todas las palabras que contienen un prefijo específico en el diccionario.

- wordsBySuffix(suffix), retorna una lista con todas las palabras que contienen un sufijo específico en el diccionario.

TAD NodoClass

Conjunto mínimo de datos:

- data, carácter, representa el carácter almacenado por el nodo.
- end, representa si el nodo es o no un carácter final de una palabra.
- children, vector de apuntadores a NodoClass, representa los hijos de un nodo.

Comportamiento (Operaciones):

- NodoClass(data, end), define la manera en la que se inicializa un objeto de tipo NodoClass.
- getData(), retorna el carácter almacenado por el objeto de tipo NodoClass.
- addChild(child, end), añade un nuevo hijo al objeto del tipo NodoClass, inicializado con la data pasada como parámetro.
- childExists(data), busca si existe un hijo que contenga la data pasada como parámetro, retorna una referencia al objeto resultado del proceso anterior.
- getChildren(), retorna un vector con las referencias a los hijos del objeto de tipo NodoClass.
- isEnd(), retorna si un objeto de tipo NodoClass es un carácter final de una palabra.
- print(level=0), retorna una impresión de forma recursiva del objeto de tipo NodoClass.
- childWords(before, palabras), retorna todas las palabras que se pueden formar a partir de un objeto de tipo NodoClass.
- setEnd(possible), actualiza el campo end con la data pasada como parámetro.

TAD TreeClass

Conjunto mínimo de datos:

- head, apuntador a objeto de tipo NodoClass, representa la cabeza del árbol

Comportamiento (Operaciones):

- TreeClass(head_data, end), define la manera en la que se inicializa un objeto de tipo TreeClass.
- addChild(data), añade un nodo con la data pasada como parámetro.
- getHead(), retorna una referencia a la cabeza del árbol.
- printTree(), retorna una impresión del árbol.
- wordsByPrefix(prefix), retorna una lista con todas las palabras que contienen un prefijo específico en el diccionario.
- wordsBySuffix(suffix), retorna una lista con todas las palabras que contienen un sufijo específico en el diccionario.
- wordEnd(word), actualiza el campo end de una palabra, retorna si esta ya estaba como final.

TAD Tree

Conjunto mínimo de datos:

- trees, vector de apuntadores a objetos de tipo TreeClass, representa todos los árboles que se tienen en el juego de Scrabble.
- file_name, cadena de caracteres, representa el nombre del archivo del cual se leyeron las palabras que contiene el diccionario.

Comportamiento (Operaciones):

- treeExist(data), retorna si existe un árbol que contenga como cabeza el primer carácter de la data pasada como parámetro.
- addTree(data, end = false), añade un árbol con la data pasada como parámetro.
- addDataToTree(data), añade una data al árbol que se necesita, en caso de no existir, lo crea.
- getFile_name(), retorna el nombre del archivo usado para inicializar los árboles.
- setFile_Name(File_name), actualiza el nombre del archivo usado para inicializar los árboles.
- printTree(), retorna una impresión del árbol.
- wordsByPrefix(prefix), busca el árbol al que la palabra corresponde y llama a la función wordsByPrefix del objeto de tipo TreeClass.
- wordsBySuffix(suffix), busca el árbol al que la palabra corresponde y llama a la función wordsBySuffix del objeto de tipo TreeClass.

TAD ScrabbleClass

Conjunto mínimo de datos:

- dictionary, objeto de tipo Diccionario, representa el diccionario que se tiene inicializado en el juego de Scrabble.
- inverse_dictionary, objeto de tipo Diccionario, representa el diccionario en orden inverso que se tiene inicializado en el juego de Scrabble.
- tree, objeto de tipo Tree, representa un contenedor con todos los árboles que se tienen inicializados en el juego de Scrabble.
- inverse_tree, objeto de tipo Tree, representa un contenedor con todos los árboles en orden invertido que se tienen inicializados en el juego de Scrabble.

Comportamiento (Operaciones):

- getDictionary(), retorna el diccionario que se tiene en el juego de Scrabble.
- getInverse_dictionary(), retorna el diccionario en orden inverso que se tiene en el juego de Scrabble.
- getTree(), retorna el árbol que se tiene en el juego de Scrabble.
- getInverseTree(), retorna el árbol que se tiene en orden inverso en el juego de Scrabble.
- start (diccionario.txt), inicializa el sistema a partir del archivo diccionario.txt, que contiene un diccionario de palabras aceptadas en el idioma inglés (idioma original del juego). El comando debe almacenar las palabras del archivo de forma que sea fácil recuperarlas posteriormente. Las palabras deben ser verificadas para no

almacenar aquellas que incluyen símbolos inválidos (como guiones, números y signos de puntuación).

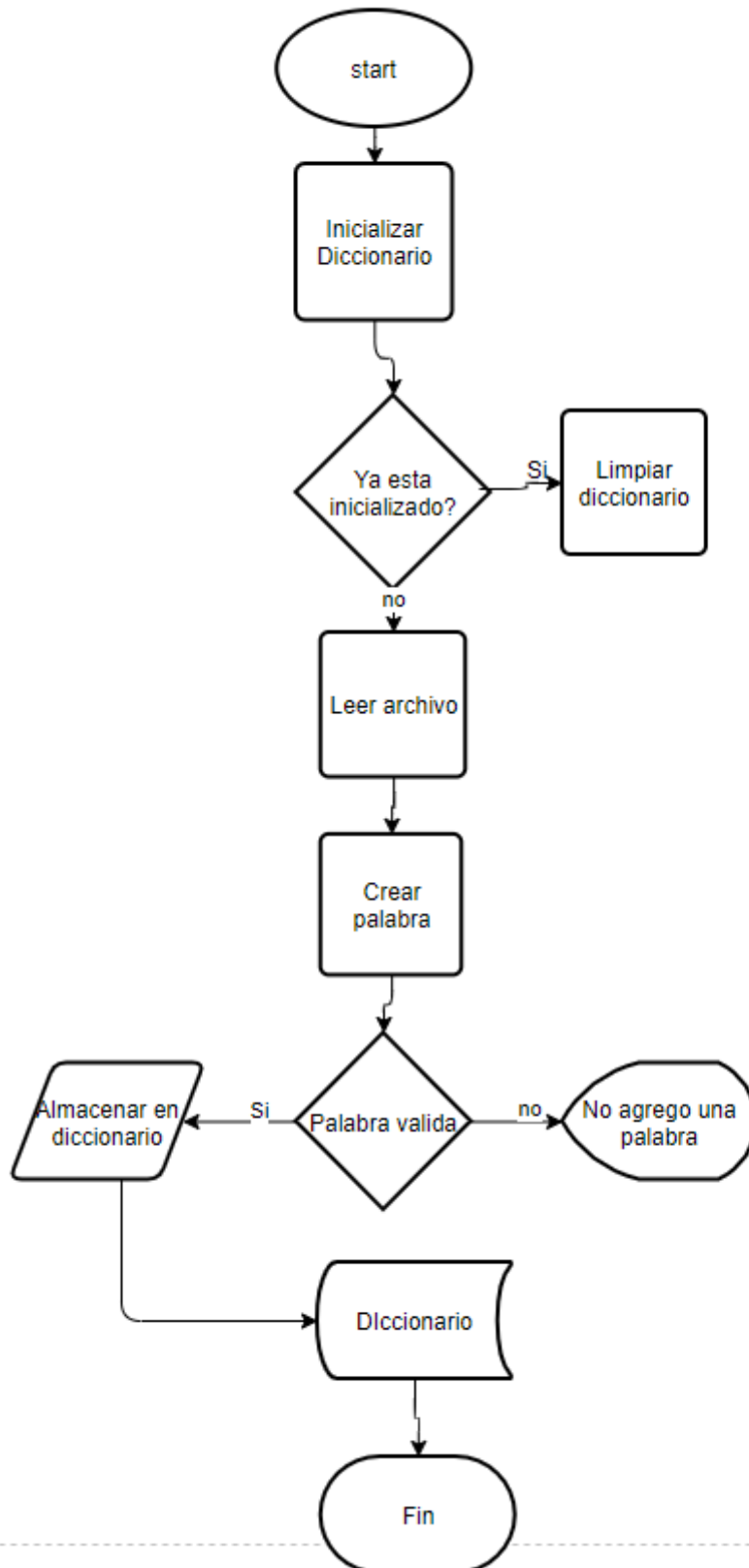
- `inverse_start` (diccionario.txt), inicializa el sistema a partir del archivo `diccionario.txt`, que contiene un diccionario de palabras aceptadas en el idioma inglés (idioma original del juego). A diferencia del comando `inicializar`, este comando almacena las palabras en sentido inverso (leídas de derecha a izquierda), teniendo en cuenta que sea fácil recuperarlas posteriormente. Las palabras también deben ser verificadas para no almacenar aquellas que incluyen símbolos inválidos (como guiones, números y signos de puntuación).
- `score` (palabra), el comando permite conocer la puntuación que puede obtenerse con una palabra dada, de acuerdo a la tabla de puntuación de cada letra presentada anteriormente. Sin embargo, el comando debe verificar que la palabra sea válida, es decir, que exista en el diccionario (tanto original como en sentido inverso), y que esté escrita con símbolos válidos.
- `exit` (), termina la ejecución de la aplicación.
- `start_tree` (diccionario.txt), inicializa el sistema a partir del archivo `diccionario.txt`, que contiene un diccionario de palabras aceptadas en el idioma inglés (idioma original del juego). A diferencia del comando `start`, este comando almacena las palabras en uno o más árboles de letras (como se considere conveniente). Las palabras deben ser verificadas para no almacenar aquellas que incluyen símbolos inválidos (como guiones, números y signos de puntuación).
- `start_inverse_tree` (diccionario.txt), inicializa el sistema a partir del archivo `diccionario.txt`, que contiene un diccionario de palabras aceptadas en el idioma inglés (idioma original del juego). A diferencia de los comandos `"start_inverse"` e `"start_tree"`, este comando almacena las palabras en uno o más árboles de letras, pero en sentido inverso (leídas de derecha a izquierda). Las palabras también deben ser verificadas para no almacenar aquellas que incluyen símbolos inválidos (como guiones, números y signos de puntuación).
- `words_by_prefix` (prefijo), dado un prefijo de pocas letras, el comando recorre el(los) árbol(es) de letras (construido(s) con el comando `"start_tree"`) para ubicar todas las palabras posibles a construir a partir de ese prefijo. A partir del recorrido, se presenta al usuario en pantalla todas las posibles palabras, la longitud de cada una y la puntuación que cada una puede obtener.
- `words_by_suffix` (sufijo), dado un sufijo de pocas letras, el comando recorre el(los) árbol(es) de letras (construido(s) con el comando `"start_inverse_tree"`) para ubicar todas las palabras posibles a construir que terminan con ese sufijo. A partir del recorrido, se presenta al usuario en pantalla todas las posibles palabras, la longitud de cada una y la puntuación que cada una puede obtener.
- `word_graph` (), con las palabras ya almacenadas en el diccionario, el comando construye un grafo de palabras, en donde cada palabra se conecta a las demás si y sólo si difieren en una única letra (con las demás letras iguales y en las mismas posiciones).
- `possible_words` (letras), dadas ciertas letras en una cadena de caracteres (sin importar su orden), el comando debe presentar en pantalla todas las posibles palabras válidas a construir, indicando la longitud de cada una y la puntuación que se puede obtener con cada una. En las letras de la cadena de caracteres, puede admitirse un único símbolo comodín (?), el cual representará una letra desconocida y permitirá generar mayores posibilidades de palabras a construir. Para este propósito,

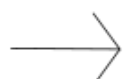
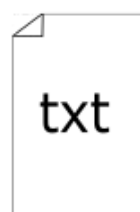
el comando debe hacer uso del grafo de palabras construido con el comando "word_graph".

- decide (comando), esta función se encarga de escoger a que función debería dirigirse la ejecución del programa, junto a los argumentos que necesita. me retorna el puntaje específico de esta palabra como número.
- find_in_dictionaries (palabra), me retorna si una palabra está en ambos diccionarios.
- sumScore (palabra), dada una palabra, busca la puntuación de esta, retorna el resultado de dicha operación.

Diagramas y dibujos

- start (diccionario.txt):



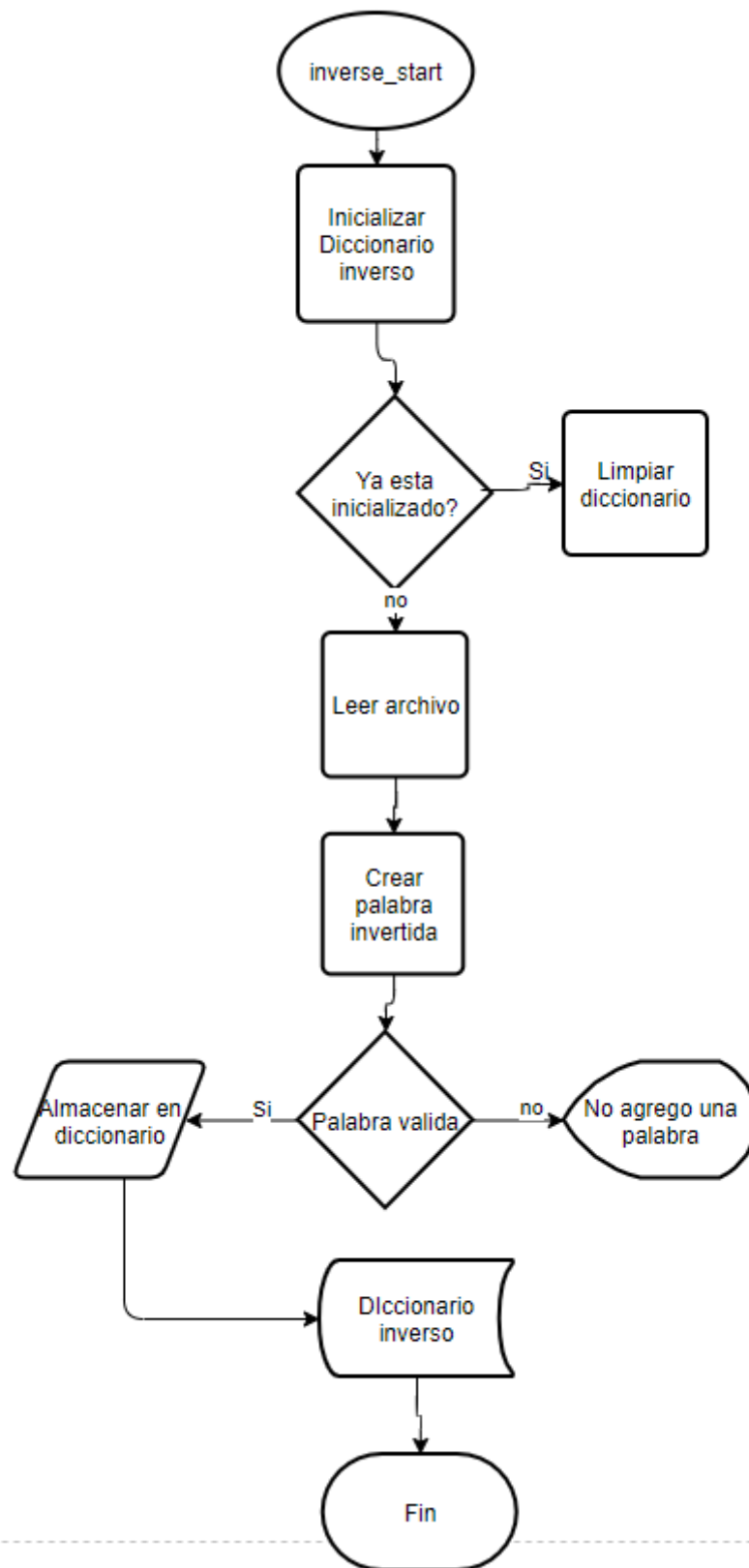


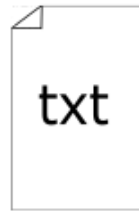
Palabra



Hola
Calle

- inverse_start (diccionario.txt):



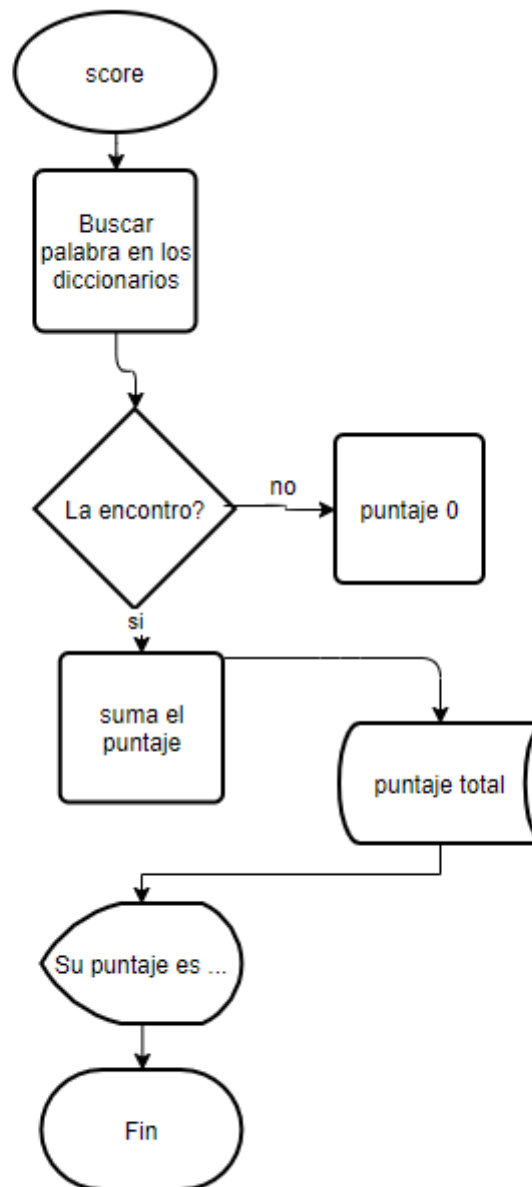


arbalaP



aloH
alleC

- score (palabra):

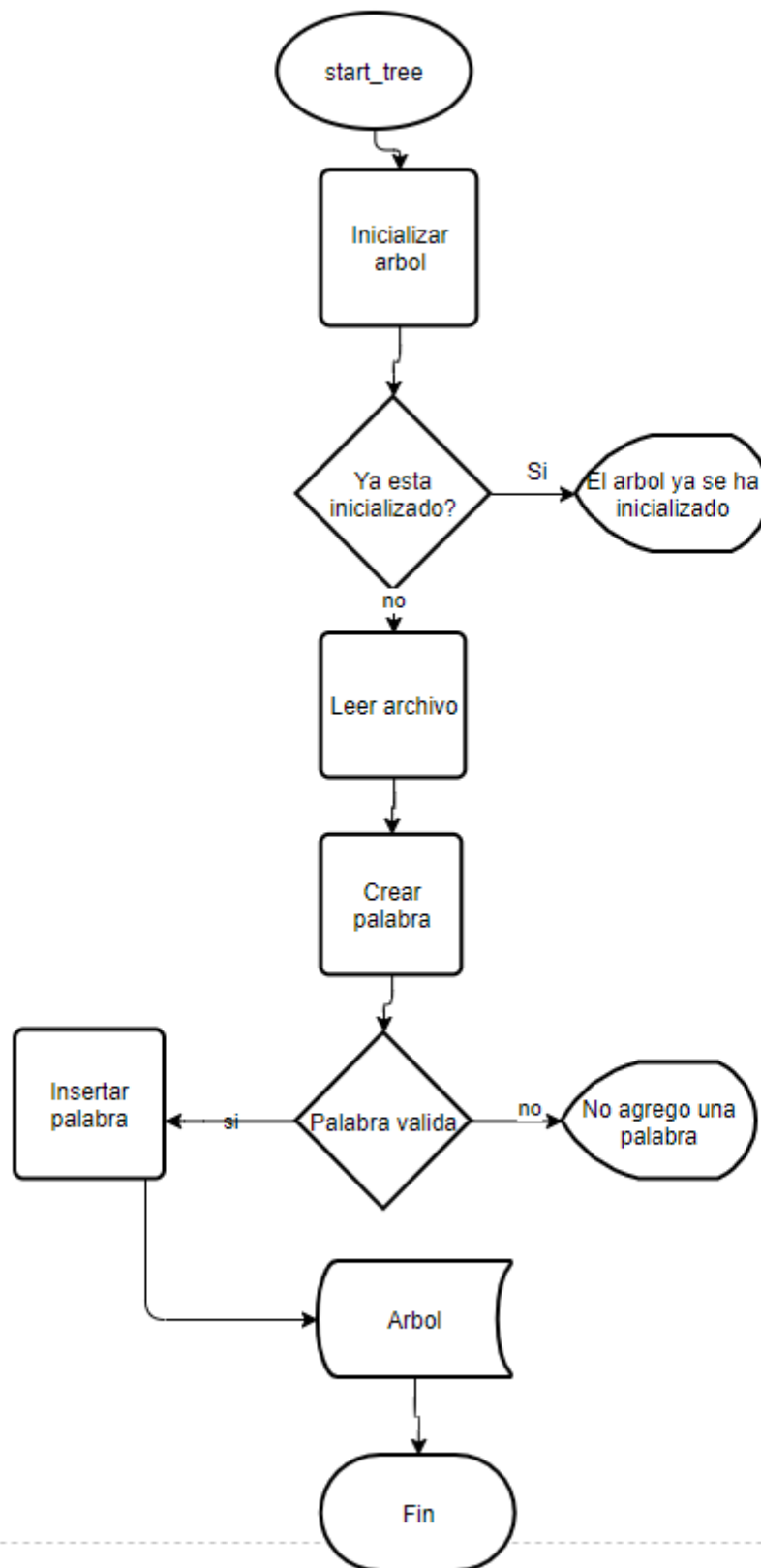


h - o - l - a

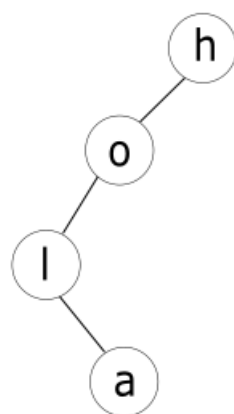
+4 +1 +1 +1

=7

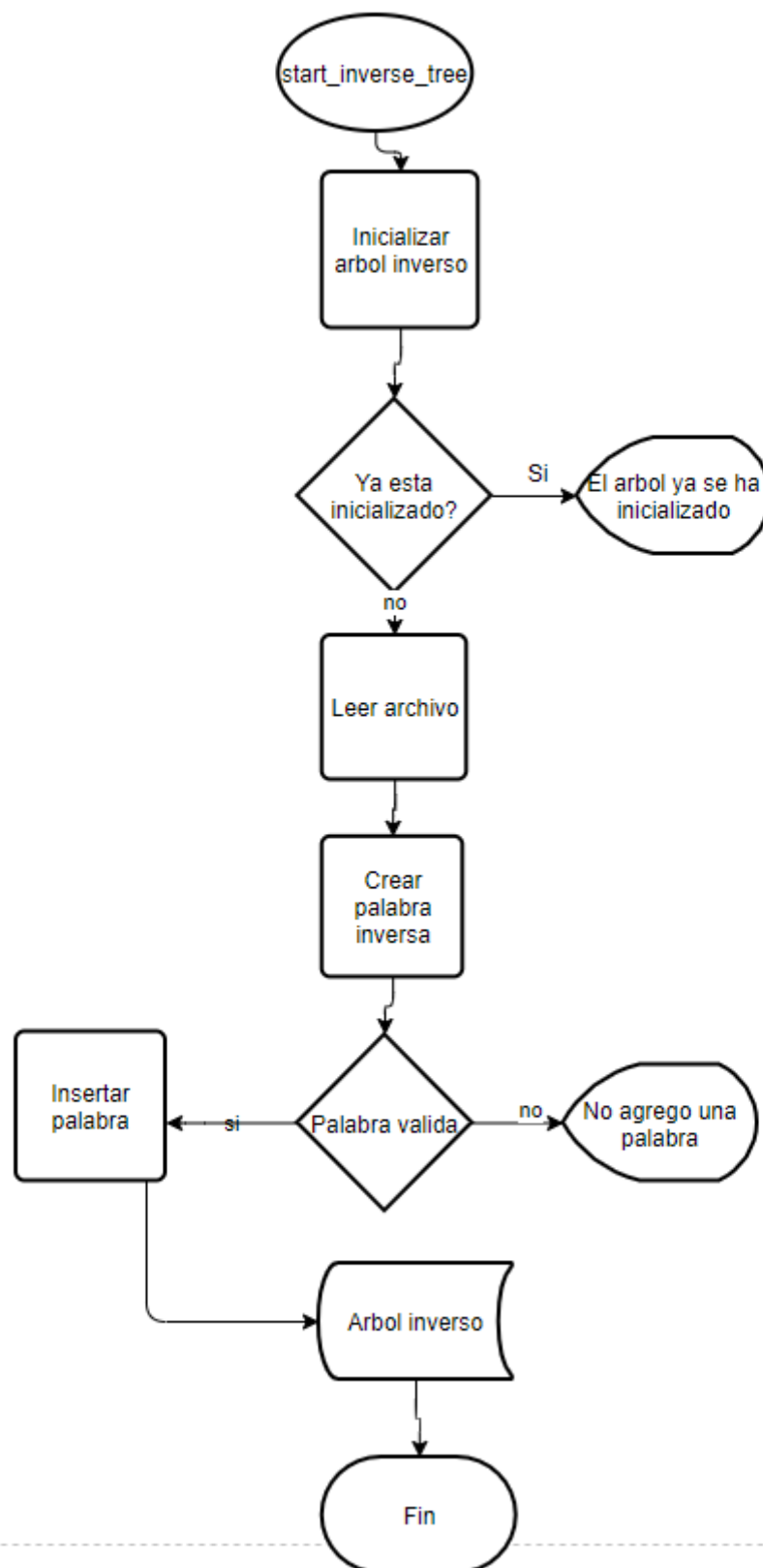
- start_tree (diccionario.txt)



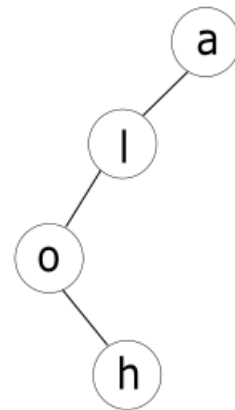
hola



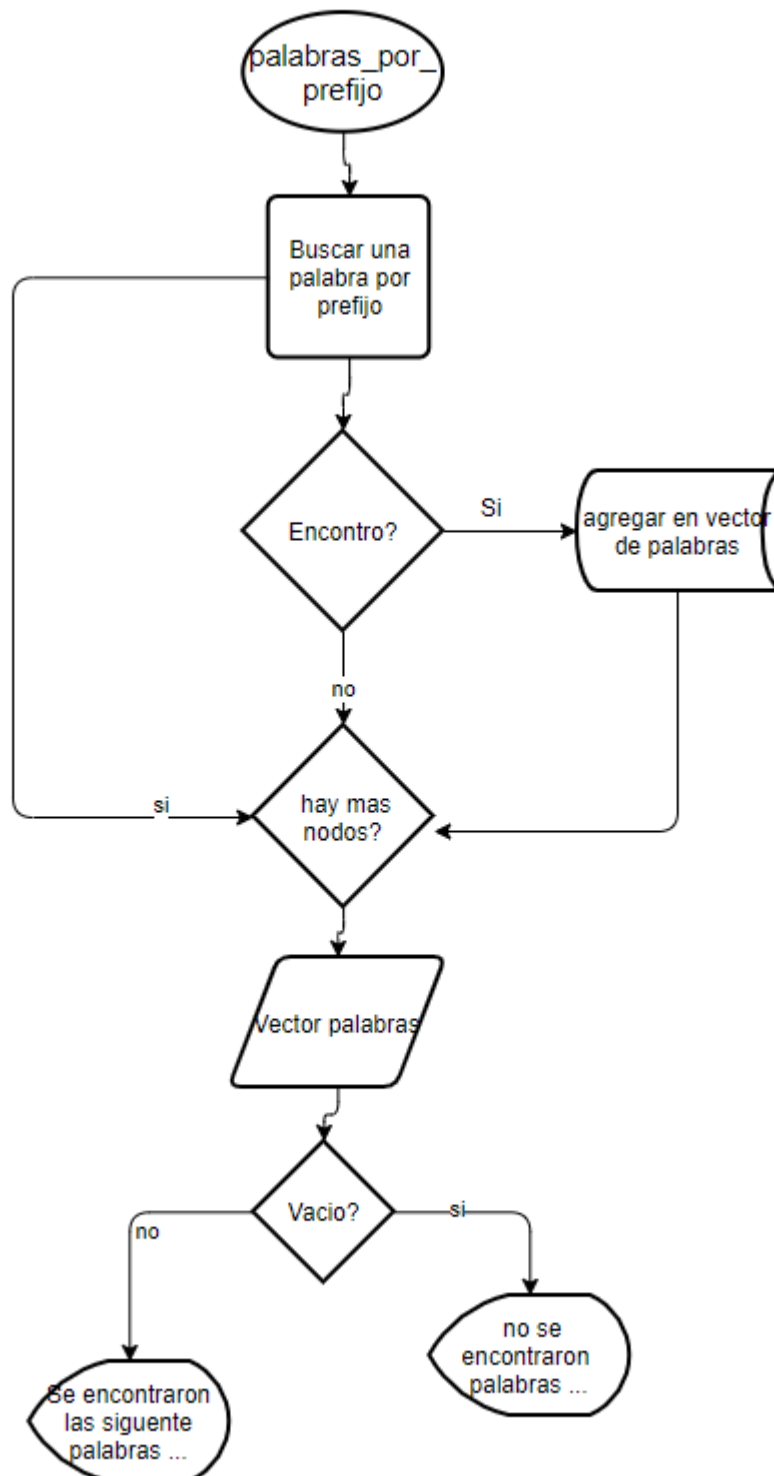
- start_inverse_tree (diccionario.txt)

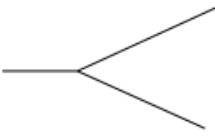


aloh

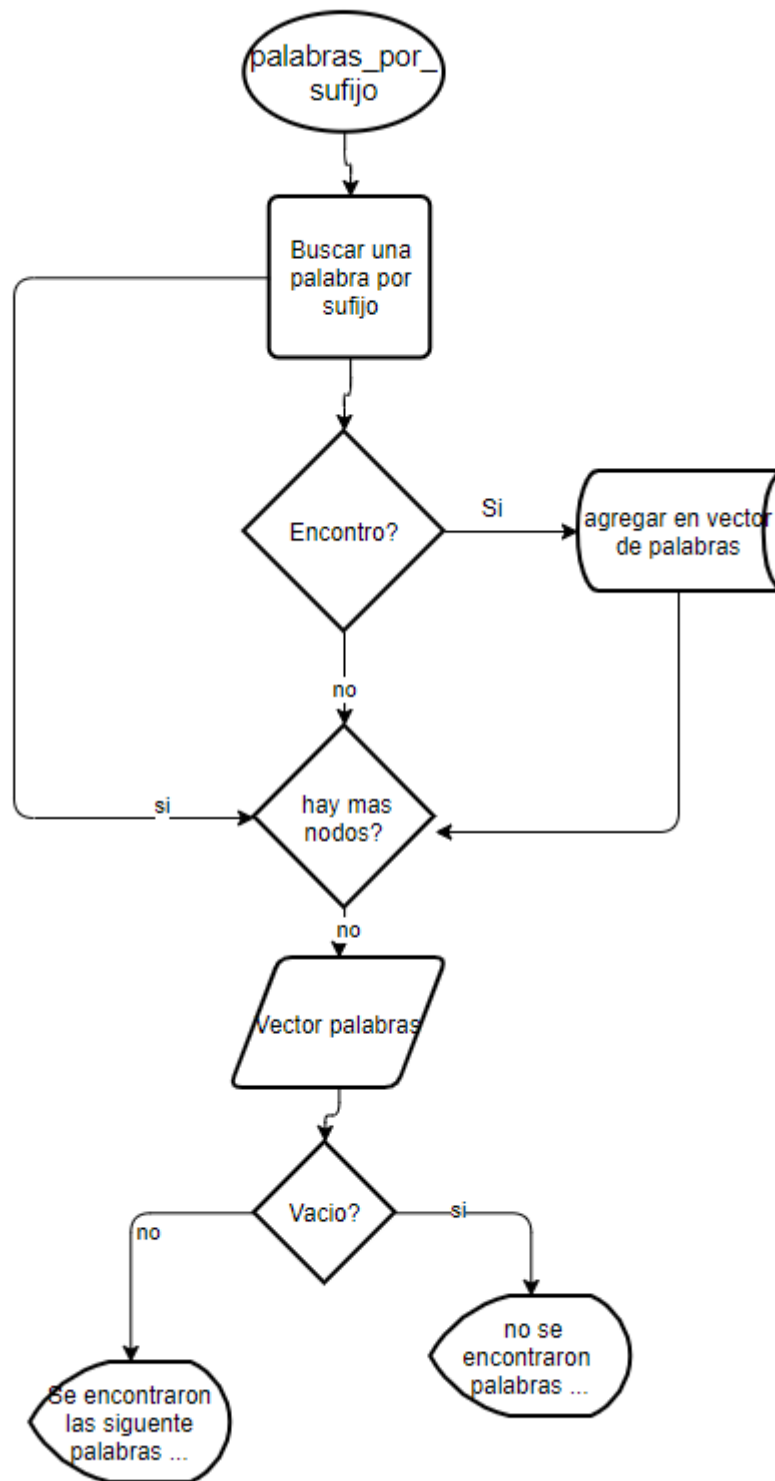


- words_by_prefix (prefijo)



→ zend  zendic
zendician

- words_by_suffix (sufijo)



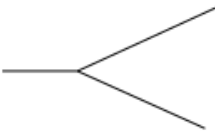
→ cer  Hacer
Ven**cer**

Diagrama de relación de TAD's

