Trabajo Práctico Final
Estructuras de datos y Algoritmos I

PARSER - ANALIZADOR SINTÁCTICO

Estudiante: Belardo, Luciano Manuel (B-6434/3)

RESUMEN:

El programa consta de un parser/analizador sintáctico simple que lee líneas de un archivo de texto y en base a un diccionario pasado como argumento y las reescribe correctamente en un tercer archivo.

CONVENCIONES/DECISIONES:

1. El ejecutable toma, en orden: un archivo diccionario, un archivo de entrada y un archivo/path de salida.

`./ejecutable ./diccionario.txt ./entrada.txt ./salida.txt`

2. El archivo diccionario debe contener únicamente palabras que:
   - Contengan únicamente letras de la A-Z. (26 letras posibles)
   - No contengan Ñ, Á, É, Í, Ó, Ú, Ü (Ni otros caracteres especiales).
   - No contengan carácteres especiales: -, ‘ ’ (espacio), ‘,’, etc.

Todas las palabras deben estar separadas una de la otra por un salto de línea “\n”. Además, el diccionario desprecia las mayúsculas y minúsculas, por lo que “HOLA”, “hola”, “Hola”, etc. son todas válidas y equivalentes. Si hay un error de formato termina la ejecución del programa

VÁLIDO	INVÁLIDO	INVÁLIDO	INVÁLIDO
Deposite dolares dolar quien recibira esperanza PERa es espera INLAW ESPANA ALFAROMEO FeRrArI	Deposite dólares dólar quién recibirá esperanza PERa es espera INLAW ESPAÑA ALFAROMEO ferrari	Deposite, dolares, dolar, quien, recibira esperanza PERa es espera IN-LAW ESPANIA ALFA-ROMEO FERRaRI	Deposite dolares dolar quien recibira esperanza PERa es espera IN LAW ESPANA ALFA ROMEO FeRrArI

3. El archivo de entrada puede contener cualquier texto y el programa corregirá las líneas por separado, dejando únicamente las palabras pertenecientes al diccionario contenidas en el texto, espaciadas entre sí.

4. El analizador: (ejemplos teniendo en cuenta el dic. válido de ejemplo)

- Borra caracteres erróneos y agrega espacios.

ENTRADA: QUIENXRECIBIRAWDOLARESY

SALIDA: QUIEN RECIBIRA DOLARES (Hubo errores) (Espacios agregados 2) (Caracteres borrados: 'X', 'W', 'Y')

- Maximiza las palabras, sin predecir a futuro.

ENTRADA: dolaresperanza

SALIDA: dolares pera (Hubo errores) (Espacios agregados 1) (Caracteres borrados: 'n', 'z', 'a')

NO OCURRE:

dolar esperanza

dolar es pera

dolar espera

- No ignora errores para ver si existe una palabra cortada.

ENTRADA: dOlAr esperXanza.

SALIDA: dOlAr es (Hubo errores) (Caracteres borrados: 'p', 'e', 'r', 'x', 'a', 'n', 'z', 'a', '.')

NO OCURRE:

dOlAr esperanza

dOlAr es pera

dolar espera

ENTRADA: quien depositWe dolares

SALIDA: quien dolares (Hubo errores) (Caracteres borrados: 'd', 'e', 'p', 'o', 's', 'i', 't', 'W', 'e', ' ')

NO OCURRE:

quien deposite dolares

- No ignora espacios para ver si existe una palabra cortada.

ENTRADA: es pera
SALIDA: es pera
NO OCURRE: espera

ENTRADA: espa na
SALIDA: es (Hubo errores) ...
NO OCURRE: espana

- No diferencia Ñ, Á, É, Í, Ó, Ú, Ü, etc. (SON ERRORES).

ENTRADA: españa

SALIDA: es (Hubo errores) (Caracteres borrados: 'p', 'a', '?', '?', 'a')

NO OCURRE:

espana

españa

ESTRUCTURAS USADAS:

DICCIONARIO:

El diccionario se almacena en un Árbol (Trie) de forma tal que una palabra pertenece al mismo si:

1. El índice respectivo de la primera letra de la palabra representa un subárbol (hijo) no vacío del árbol raíz.
2. El índice respectivo de cada letra subsiguiente de la palabra representa un subárbol (hijo) no vacío de la letra previa.
3. El nodo representativo de la última letra de la palabra contiene el validador "esFinDePalabra" en 1 (Verdadero).

Por lo tanto, como consecuencia de estas reglas, si un nodo no posee hijos entonces no hay palabras con el prefijo respectivo en el diccionario, y ese prefijo es palabra (excepto en el caso de la raíz). Sin embargo, es posible que existan palabras que contengan dicho prefijo, por lo que no es condición suficiente para que un nodo sea palabra que no tenga hijos (para ello existe el validador "esFinDePalabra").

Asimismo, si un nodo posee hijos, obligatoriamente existe al menos una palabra con el prefijo respectivo.

El diccionario/árbol se define de la siguiente forma:

```
#define LARGO_ALFABETO 26 // Cantidad de hijos por nodo
/**
* Tipo de dato que representa cada nodo de un Trie.
*/
typedef struct _ANodo {
  struct _ANodo ** hijo; // hijo[LARGO_ALFABETO]
  int esFinDePalabra; // Indicador de si este nodo es el fin de una palabra.
} ANodo;
/**
* Tipo de dato que representa al diccionario (es un Trie).
*/
typedef ANodo * Diccionario;
```

En donde cada nodo posee un subárbol de *LARGO_ALFABETO* hijos y el validador esFinDePalabra.

Véase que cada nodo no posee ninguna variable de tipo Dato, por lo que no es posible saber desde el propio nodo de qué letra se proviene. Sin embargo esto no es un inconveniente, ya que en este programa la existencia o no de una palabra queda determinada por la existencia subsecuente de cada uno de los nodos hijos, y no por un valor almacenado.

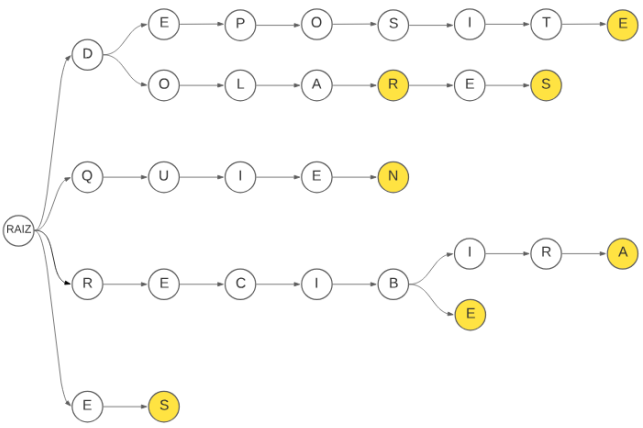
Para ello, cada posible letra del alfabeto tiene asignada por una biyección un índice representativo previamente establecido.

```
/*
* Devuelve la traducción / biyección del caracter pasado. Si es inválido
* retorna -1.
  0: A, 1: B, 2: C, 3: D, 4: E, 5: F, 6: G, 7: H, 8: I, 9: J, 10:K, 11:L, 12:M,
  13:N, 14:O, 15:P, 16:Q, 17:R, 18:S, 19:T, 20:U, 21:V, 22:W, 23:X, 24:Y, 25:Z
*/
int validar_y_traducir_caracter(char letra) {
  if (!isalpha(letra))
      return -1;

  return toupper(letra) - 'A'; //((int) letra) - 65;
}
```

Por ende, una letra es hija de un nodo si existe un nodo almacenado en la variable `nodo.hijo[indice]` en donde el índice es el número asignado a la letra en la función `validar_y_traducir_caracter`.

Por ejemplo, si el diccionario es: [DOLAR, DOLARES, DEPOSITO, QUIEN, RECIBE, RECIBIRA, ES], entonces el arbol se vería graficamente de la siguiente manera, en donde cada flecha representa la variable nodo.hijo[indice(char)] y los nodos amarillo representan los nodos cuya variable esFinDePalabra = 1.
Véase que como no existe una palabra que comience por "N", entonces nodo.hijo[indice("N")] = NULL (no existe) siendo indice la función antedicha.



A la hora de crear el diccionario, simplemente se lee caracter por caracter y se van agregando los nodos respectivos al índice de la letra leída (o desplazándose sobre ellos si el nodo ya está creado) y validando el último, para luego volver a comenzar desde la raíz para la siguiente palabra.

LÍNEA DE SALIDA

Cada línea de salida es una estructura definida de la siguiente manera:

```
/*
  * Tipo de dato que contiene el la linea de entrada corregida, la cantidad de
  * ' ' agregados y todos los caracteres eliminados.
*/
typedef struct _LineaDeSalida {
  char * resultado;
  int contadorEspaciosAgregados;
  Cola errores;
} LineaDeSalida;
```

En donde cada una contiene tres variables: La línea de entrada corregida (el resultado), el contador de espacios agregados y los errores.

La línea resultado es una simple string, el contador de espacios es un entero y por último los errores están definidos a través de una Cola, en donde cada error que se encuentra se va almacenando a medida que se lee la línea original, para luego mostrarlos en el orden de aparición.

La colas utiliza una implementación de listas genéricas, por lo que estas también son genéricas, por lo que también se definieron las funciones pertinentes al dato requerido en este caso (letras, char) en caracter.h

ALGORITMO

El programa toma el path de los tres archivos y los abre. Para luego guardarlos en una estructura.

```
/**
* Tipo de dato contenedor de los archivos pasados como argumento.
*/
typedef struct _Archivos {
FILE * diccionario, * entrada, * salida;
} Archivos;
```

Esta es llamada por la función parser, que se encarga de realizar el algoritmo.

```
Archivos * archivos = archivos_abrir(argv);
parser(archivos);
```

parser se encarga de crear el diccionario y luego va ejecutando el analizador sobre cada línea del archivo de entrada leída,

```
// Mientras la linea no sea vacía
while ((lineaEntrada = obtener_linea_de_archivo(LARGO_MAXIMO,
archivos->entrada))) {
  // La analiza y corrige, retornandola corregida y con sus errores
  LineaDeSalida lineaSalida = parser_analizar_linea(diccionario, lineaEntrada);
```

para luego imprimir el resultado en el archivo de salida

```
  // Imprime en archivo el resultado
  lineaSalida = imprimir_en_archivo(lineaSalida, archivos->salida);
```

mientras haya líneas en el archivo.

EL ANALIZADOR

parser_analizar_linea toma el diccionario y la línea de entrada y la corrige.

En principio, la idea para reconocer palabras dentro de una frase es que, se va leyendo letra por letra y, a medida que se van leyendo las letras también existe un nodoActual que se va desplazando desde la raíz a medida que las letras van apareciendo.

Para el analizador, no existe explícitamente el término “palabras válidas”, sino que, frente a la validez del nodoActual, imprime lo almacenado en el buffer (temp). Por ejemplo, si el diccionario es [dolar, dolarizacion], y la entrada es “dolarizacion” el analizador imprimirá respectivamente

“dólar” y luego adyacentemente “izacion”, haciendo ver en pantalla que imprimió “dolarizacion”, pero nunca almacenando realmente la palabra completa “dolarización”, sino el prefijo/sufijo válido en el momento oportuno.

Para funcionar, el algoritmo posee las siguientes variables (además del diccionario):

- **entrada**: la línea de entrada.

- **i**: es el índice leyendosé de la línea de entrada.

- **temp**: variable de almacenamiento temporal, donde se van a ir almacenando las letras consecuentemente mientras estas sean válidas en una posible palabra o sufijo de palabra.

```
// Variable donde se almacenan las letras mientras sean válidas
char temp[strlen(lineaEntrada)];
vaciar_cadena(temp);
```

- **continuar**: indica si se llegó al final de la línea o debo seguir.

```
// Indicador de si la el caracter es EOL
int continuar = 1;
```

- **resultado**: va almacenando la línea con las palabras válidas y espaciadas.
- **errores**: cola donde se van almacenando los errores a medida que ocurren. Estos se producen si se un carácter no pertenece ni al alfabeto o ni a una subrama válida del nodoActual.
- **espaciosAgregados**: es un contador de los espacios agregados por el analizador, desde el punto de vista del usuario. Este se incrementa en 1 cada vez que el analizador coloca un espacio, y se resta en uno cada vez que el usuario colocó correctamente un espacio por su cuenta (Por ello a nivel usuario, puesto que a nivel algoritmo todos los espacios los colocó el analizador).

```
LineaDeSalida lineaSalida = lineaDeSalida_crear(strlen(lineaEntrada));
```

- **bandera**: Es un indicador que dice a donde retomar la lectura en caso encontrar una falla. Es un *checkpoint*. Si temp está vacío, entonces si hay un error, el error es la propia letra, por lo que no existe necesariamente bandera (-1) y simplemente se guarda el error y se continúa. Por otro lado, si se está comenzando a leer una posible palabra, entonces la bandera va a almacenar el índice actual, ya que, en caso de fallar en el proceso de intentar buscar una palabra en el subárbol iniciado por el carácter leído, voy a querer retomar la lectura desde el carácter siguiente al inicial (entrada[bandera]). Véase que si hay una falla en el proceso, el carácter que provocó el error va a ser el primero de temp.

Además, es un indicador de si temp está vacío. Podría decirse que es equivalente decir que si bandera == -1 => temp es vacío.

Cada vez que se encuentra una "subpalabra", este se desplaza hacia el índice actual, puesto que efectivamente no hubo errores, pero si los puede haber en la posterior lectura.

```
// 'Checkpoint' que indica a donde regresar en caso de error
int bandera = -1;
```

- **espacioValido**: es un indicador que dice si es válido colocar un espacio en el lugar actual. Sirve para saber si el usuario colocó los espacios válidos donde y cuando correspondía, y devolver con mayor certeza los errores habidos. Se vuelve verdadero siempre luego de escribir una palabra en resultado, y se vuelve falso en el momento en el que aparece un espacio en la línea de entrada. Véase que solamente sirve para manejar errores, puesto que para el analizador no es más que un caracter inválido pues no pertenece al alfabeto. Todos los espacios resultantes en *resultado* son introducidos por el analizador.

```
// Indicador de si es válido poner un espacio según el contexto
int espacioValido = -1;
```

- **seguir**: es un indicador que dice si ya se encontró una "subpalabra" válida en la rama actual del árbol (y por lo tanto se transcribió en *resultado*), pero se continúa analizando esta subrama en caso de que aparezca una "extensión" de la palabra. Esto sirve para saber que, en caso de haber un error, el mismo provino de intentar "extender" una "subpalabra" y fallar en el intento, pero no necesariamente hubo un error a nivel usuario.

  Por ejemplo: Diccionario = [mes, mesas, azul]

  Entrada = mesazul

Luego de encontrar válida la subpalabra "mes", el analizador va a intentar extender esta palabra. Será válida la "a", pero inválida la "z" (en esta subrama, pues "mesaz" no existe). Por lo que la letra "z" sería un error, pero como proviene del intento de extender una palabra, no lo es necesariamente. Efectivamente, luego el analizador continua desde "a" con el diccionario "reiniciado" y encuentra la palabra "azul", por lo que devuelve:

Salida: mes azul

También sirve para saber si debo poner un espacio antes de escribir la subpalabra encontrada en caso de ser válida

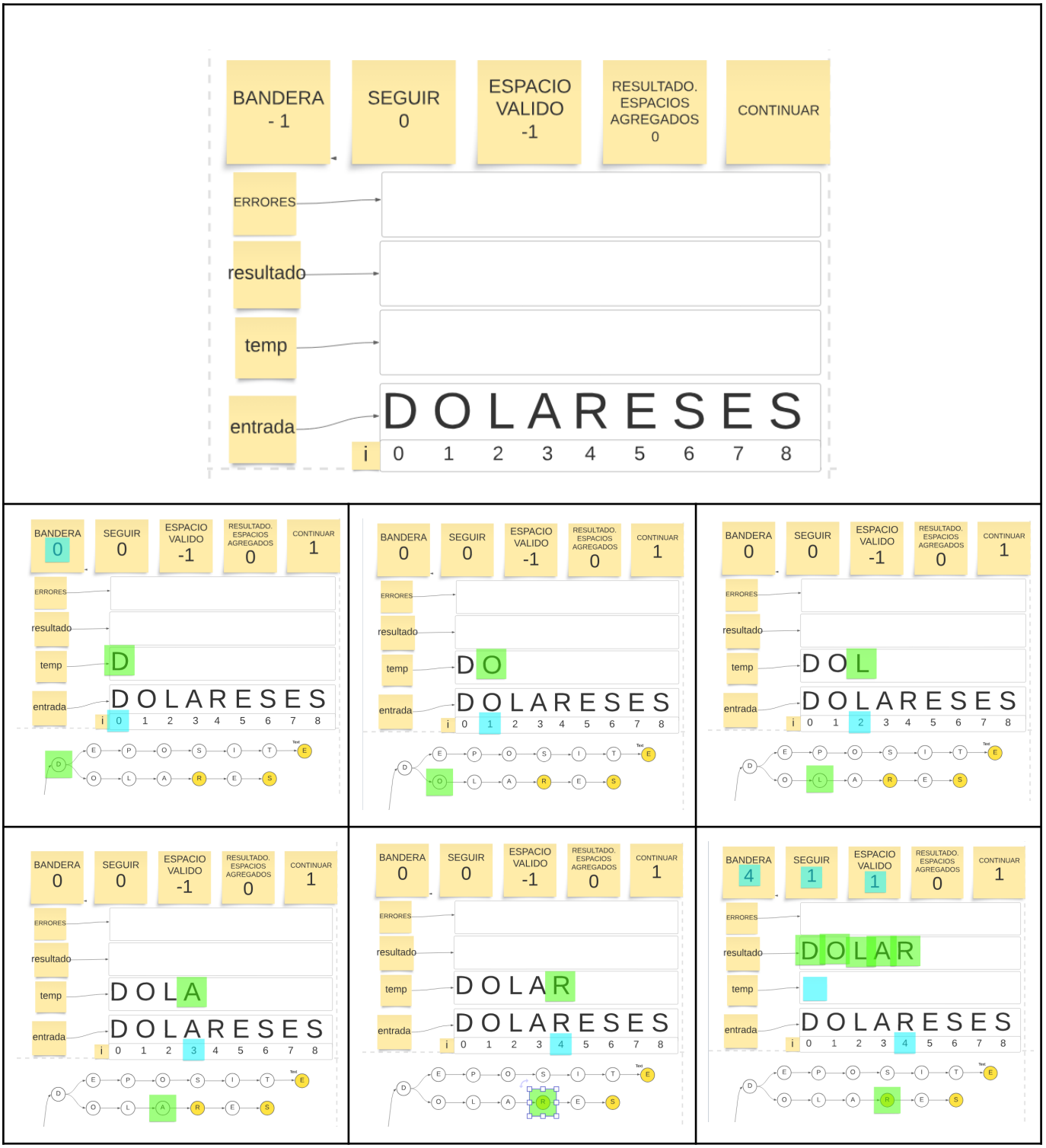
```
// Indicador de si se continua leyendo en la subrama de una subpalabra válida
int seguir = 0;
```

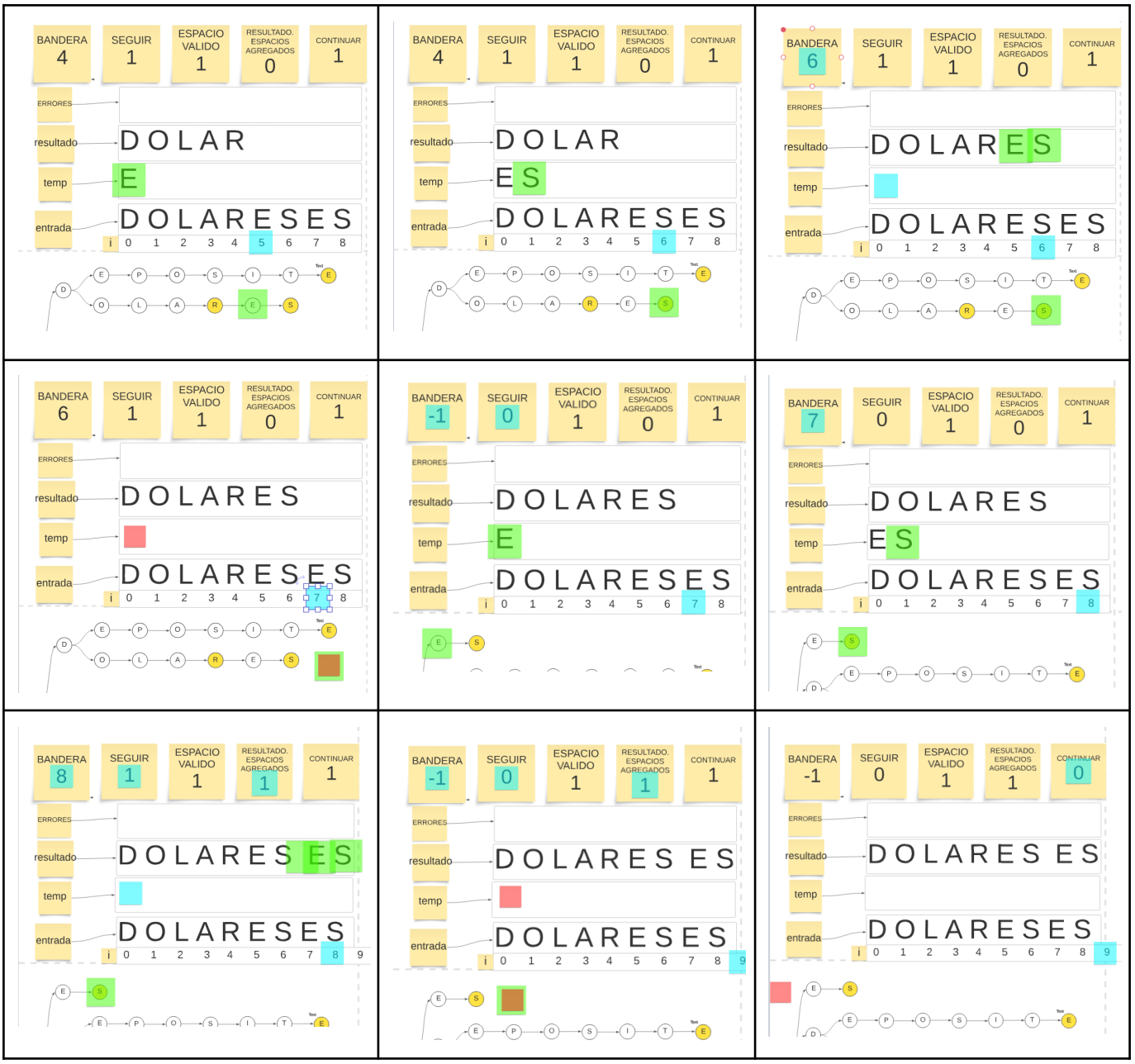
- **nodoActual**: es la variable que representa la subrama sobre la que se está trabajando actualmente. Al comenzar una palabra desde 0 es la raiz del diccionario, y se va desplazando a medida que encuentra letras válidas.

```
Diccionario nodoActual = diccionario;
```

Ejemplos:

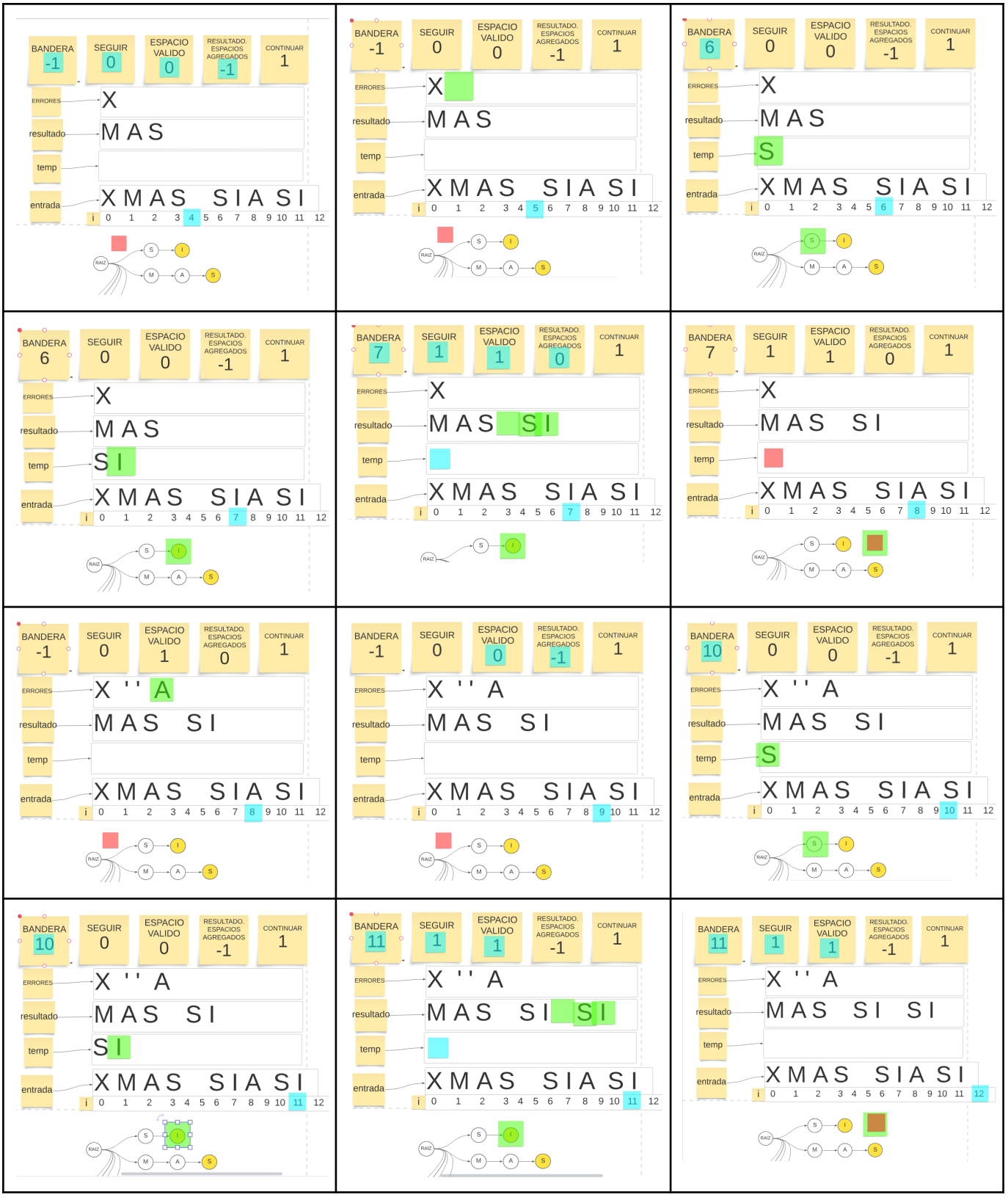
Suponiendo el mismo árbol usado antes de ejemplo, si la línea de entrada es: “DOLARESES” el programa haría lo siguiente:

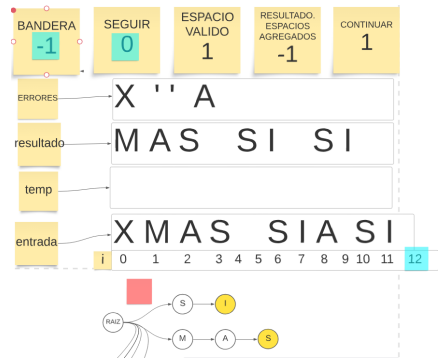
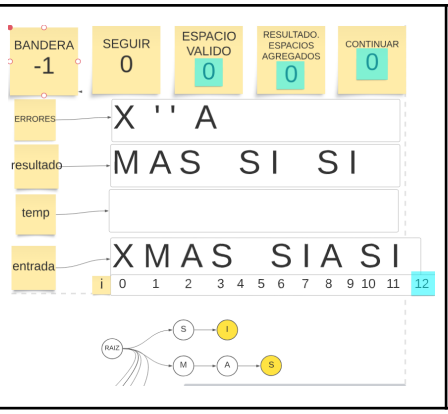




RESULTADO: “DOLARES ES”
ERRORES “”
ESPACIOS AGG: 1

Otro ejemplo: Supongamos que el diccionario contiene: [mas, si]
Y proporcionamos “XMAS SIA SI” como entrada


BANDERA -1 SEGUIR 0 ESPACIO VALIDO 0 RESULTADO. ESPACIOS AGREGADOS -1 CONTINUAR 1
ERRORES X
resultado M A S
temp
entrada X M A S S I A S I
i 0 1 2 3 4 5 6 7 8 9 10 11 12
RAIZ S I M A S
BANDERA -1 SEGUIR 0 ESPACIO VALIDO 0 RESULTADO. ESPACIOS AGREGADOS -1 CONTINUAR 1
ERRORES X
resultado M A S
temp
entrada X M A S S I A S I
i 0 1 2 3 4 5 6 7 8 9 10 11 12
RAIZ S I M A S
BANDERA 6 SEGUIR 0 ESPACIO VALIDO 0 RESULTADO. ESPACIOS AGREGADOS -1 CONTINUAR 1
ERRORES X
resultado M A S
temp S
entrada X M A S S I A S I
i 0 1 2 3 4 5 6 7 8 9 10 11 12
RAIZ S I M A S
BANDERA 6 SEGUIR 0 ESPACIO VALIDO 0 RESULTADO. ESPACIOS AGREGADOS -1 CONTINUAR 1
ERRORES X
resultado M A S
temp S I
entrada X M A S S I A S I
i 0 1 2 3 4 5 6 7 8 9 10 11 12
RAIZ S I M A S
BANDERA 7 SEGUIR 1 ESPACIO VALIDO 1 RESULTADO. ESPACIOS AGREGADOS 0 CONTINUAR 1
ERRORES X
resultado M A S S I
temp
entrada X M A S S I A S I
i 0 1 2 3 4 5 6 7 8 9 10 11 12
RAIZ S I M A S
BANDERA 7 SEGUIR 1 ESPACIO VALIDO 1 RESULTADO. ESPACIOS AGREGADOS 0 CONTINUAR 1
ERRORES X
resultado M A S S I
temp
entrada X M A S S I A S I
i 0 1 2 3 4 5 6 7 8 9 10 11 12
RAIZ S I M A S
BANDERA -1 SEGUIR 0 ESPACIO VALIDO 1 RESULTADO. ESPACIOS AGREGADOS 0 CONTINUAR 1
ERRORES X ' ' A
resultado M A S S I
temp
entrada X M A S S I A S I
i 0 1 2 3 4 5 6 7 8 9 10 11 12
RAIZ S I M A S
BANDERA -1 SEGUIR 0 ESPACIO VALIDO 0 RESULTADO. ESPACIOS AGREGADOS -1 CONTINUAR 1
ERRORES X ' ' A
resultado M A S S I
temp
entrada X M A S S I A S I
i 0 1 2 3 4 5 6 7 8 9 10 11 12
RAIZ S I M A S
BANDERA 10 SEGUIR 0 ESPACIO VALIDO 0 RESULTADO. ESPACIOS AGREGADOS -1 CONTINUAR 1
ERRORES X ' ' A
resultado M A S S I
temp S
entrada X M A S S I A S I
i 0 1 2 3 4 5 6 7 8 9 10 11 12
RAIZ S I M A S
BANDERA 10 SEGUIR 0 ESPACIO VALIDO 0 RESULTADO. ESPACIOS AGREGADOS -1 CONTINUAR 1
ERRORES X ' ' A
resultado M A S S I
temp S I
entrada X M A S S I A S I
i 0 1 2 3 4 5 6 7 8 9 10 11 12
RAIZ S I M A S
BANDERA 11 SEGUIR 1 ESPACIO VALIDO 1 RESULTADO. ESPACIOS AGREGADOS -1 CONTINUAR 1
ERRORES X ' ' A
resultado M A S S I S I
temp
entrada X M A S S I A S I
i 0 1 2 3 4 5 6 7 8 9 10 11 12
RAIZ S I M A S
BANDERA 11 SEGUIR 1 ESPACIO VALIDO 1 RESULTADO. ESPACIOS AGREGADOS -1 CONTINUAR 1
ERRORES X ' ' A
resultado M A S S I S I
temp
entrada X M A S S I A S I
i 0 1 2 3 4 5 6 7 8 9 10 11 12
RAIZ S I M A S

		RESULTADO: “MAS SI SI” ERRORES: “X, , A” ESPACIOS AGG: 0

En resumen: el analizador va leyendo línea por línea e intenta llegar lo más lejos posible en las ramas del árbol. A medida que encuentra una subpalabra, la escribe en el resultado ya que si o si la subcadena es correcta, e intenta extenderla. Si lo consigue, la extiende, y si no lo consigue, reinicia desde el último carácter impreso en *resultado* y vuelve a buscar otra palabra desde 0. Si en el proceso general no encuentra nunca una subpalabra, escribe el carácter erróneo en *errores* y continúa el análisis desde el consiguiente al erróneo. Antes de escribir cada prefijo de palabra, coloca un “ ” en *resultado*.