Analizador Sintáctico

Estructuras de Datos y Algoritmos I

Tomás Maiza - M-7116/1

1. Funcionamiento del programa

El programa recibe como entrada tres archivos .txt (diccionario, frases de entrada y salida). Se almacena el diccionario en un árbol con 26 hijos (uno para cada letra del abecedario). Luego se lee cada frase del archivo de entrada y se obtiene la cadena espaciada buscando palabras maximizadas en el árbol que almacena el diccionario. Una vez que se obtiene una cadena espaciada, se imprime (junto a los errores encontrados) en el archivo de salida.

2. Estructuras de Datos

Utilicé un **trie** para almacenar el diccionario ya que permite recuperar palabras de forma que la cantidad de candidatos se va limitando a medida que se recorre la cadena buscada (las palabras candidatas tienen todas el mismo prefijo). De esta forma se evita recorrer palabras más cortas que la buscada que no comparta el prefijo con ella.

La estructura del trie tiene la siguiente forma:

```
typedef struct _GTNodo {
  char dato;
  struct _GTNodo** hijos;
  EstadoDeAceptacion estado;
} GTNodo;

typedef GTNodo* GTree;
```

donde *dato* es el caracter a almacenar, el array de punteros a nodos *hijos* almacena la dirección de 26 nodos (uno por cada letra, todos inicializados en NULL) y *estado* indica si el estado del nodo es de aceptación, es decir que en ese caracter finaliza una palabra que se halla en el diccionario, o si no lo es y por lo tanto en ese nodo no finaliza ninguna palabra

almacenada.

A la hora de leer una cadena y pasar a analizarla, para almacenar la cadena espaciada y los errores encontrados uticé una estructura Salida que almacena un array de caracteres *cadenaFinal* que guarda la cadena espaciada resultante, el entero *indiceCadena* que almacena el índice posterior al último lugar del array donde se insertó un caracter válido y la cola *errores* que almacena los errores que se hallan a medida que se analiza la cadena.

```
typedef struct {
  String cadenaFinal;
  int indiceCadena;
  Cola errores;
} Salida;
```

3. Algoritmo

La función *main* recibe como entrada tres archivos de texto: el diccionario, las frases a espaciar y la salida con las frases analizadas. Una función se encarga de abrir los tres archivos y luego se pasa a ejecutar el analizador sintáctico.

Primero se crea un GTree (trie) que almacenará el diccionario. Para ello, cuando se lee una palabra se almacena cada letra en un nodo del árbol con el siguiente criterio:

- La letra inicial de una palabra se almacena en el nodo hijo correspondiente a la raiz del árbol. Si la posición del array correspondiente a dicho caracter es NULL se llena la información del nodo, sino simplemente nos movemos a dicho nodo.

- En una misma palabra cada letra es hija de la anterior, es decir que los ancestros del caracter *i* de una palabra son los caracteres *1, 2, ..., i-1*, donde el caracter *i-1* es su padre. Así las palabras quedan almacenadas como una rama del árbol y cada prefijo existente en el diccionario se almacena una única vez.

- Si el caracter leído es un salto de línea significa que se terminó de leer una palabra, por lo cual el estado del nodo actual (que almacena el último caracter de la palabra) pasa a ser de aceptación.

Una vez leido y almacenado, se cierra el archivo .txt correspondiente al diccionario.

Luego se procede a leer la entrada de frases: se lee una palabra del archivo caracter por caracter aplicando *tolower* a cada uno y se almacena en un String. Si la cadena leída es vacía no se hace nada (pasamos a la siguiente línea de la entrada). En caso contrario pasamos a analizar la cadena leída. Inicializamos la estructura **Salida** explicada en la sección anterior y comenzamos a leer la cadena sin espacios. Tenemos una variable entera **indice** que indica el índice del caracter a leer en la cadena leída y otra variable entera **indiceInicioPalabra** que nos indica el índice en la cadena leída del caracter a partir del cual buscaremos la palabra

más larga posible que lo tenga como inicial. La variable comienza en el valor 0 y la función terminará de ejecutarse cuando la misma supere el máximo índice que almacena la cadena (cuando alcance su tamaño). El algoritmo funciona de la siguiente manera:

1. Buscamos el caracter leido en el **indice** actual de la cadena leída en el diccionario: si el nodo actual del diccionario (la raiz si estamos buscando el inicio de una palabra) tiene por hijo en la posición correspondiente al caracter leído un nodo no NULL significa que el prefijo existe en el diccionario y por lo tanto puede haber una palabra válida. En dicho caso retornamos ese nodo, si el hijo fuera NULL retornamos NULL. Ahora se abren dos posibles casos:

- Si el nodo no es NULL almacenamos el caracter leído en la cadena espaciada y si el nodo es de aceptación almacenamos el índice del caracter en la cadena. Luego incrementamos la variable *indice* y repetimos el bucle.
- Si el nodo es NULL nos fijamos si previamente encontramos un estado de aceptación, es decir si en algún caracter anterior de la cadena se leyó una palabra válida. De ser así volvemos al índice consecutivo al de aceptación y colocamos un espacio en la cadena espaciada (ya que se leyó un palabra válida en algún momento del bucle) y ahora **indiceInicioPalabra** toma el valor del caracter posterior al de aceptación, esto es, ahora vamos a buscar un palabra que comience con el primer caracter que aún no pertenece a ninguna palabra ni es un error. Si nunca se encontró un estado de aceptación entonces ninguna palabra comienza con el prefijo leido y por lo tanto el caracter inicial es un error, y lo almacenamos en la cola de errores. En este caso **indiceInicioPalabra** se incrementará en un unidad (la letra siguiente al error). Luego se repite el bucle.

2. Una vez finalizado el bucle se terminó la lectura de la cadena leída y procedemos a imprimir la cadena espaciada y la cola de errores en el archivo de salida. Luego repetimos para la siguiente línea de la entrada y finalizamos cuando lleguemos al EOF.

Después de leer todas las frases de la entrada se cierran los archivos y se libera la memoria del diccionario.