Informe TP Final
Diccionario Buscador

Miranda, Nicolás - Blanco, Santiago

Diccionario

- *cargarArchivos*: lee N archivos, donde N es menor o igual al número de archivos txt en la carpeta del proyecto. El texto de estos archivos se utiliza para crear el diccionario.
- *insertarTermino*: inserta un nuevo nuevo término en el arreglo.
- *archivoToDic*: abre todos los archivos con texto y lee todas las palabras en estos y los convierte en términos, los ingresa al arreglo de términos uno a uno.
- *arregloToDic*: abre el archivo "diccionario.bin" y carga la información del arreglo de términos en dicho archivo.

Buscador

- *diccionarioToBuscador*: Función principal de la operación. Lee los términos del archivo diccionario y los envía al buscador.
- *cargaBuscador*: Función intermedia que determina si debe insertar un nodo nuevo al buscador o si debe agregar una nueva ocurrencia.
- *insertaPalabra*: Inserta un nodo nuevo en el buscador ordenado de manera alfabética.
- *insertaOcurrencia*: Agrega una nueva ocurrencia de un término a su nodo, primero ordenando por orden de id de documento y luego por posición en el documento.
- *encuentra*: Busca un nodo en el árbol buscador. Si lo encuentra lo devuelve y si no devuelve NULL.
- *creaNodoBuscador*: Crea un nuevo nodo para el árbol buscador.
- *creaNodoOcurrencia*: Crea un nuevo nodo para la lista de ocurrencias.
- *inorder*: Muestra los datos del buscador ordenados por orden alfabético.

1. Busca apariciones de un término

- *buscarPostearAparicionesPalabra*: busca una palabra en el árbol recibida por parámetro e informa por pantalla todas las ocurrencias de la palabra, indicando la/s pos y el/los idDoc donde se encontró.

2. Busca apariciones de un término en diferentes documentos

- *cargarArregloIds*: permite que el usuario indique los archivos a buscar una palabra, mediante los ids.

- *existePalabraEnDoc*: busca una palabra en el doc mediante su id, retorna 1 si existe y 0 si no existe.
- *buscarAparicionesAND*: comprueba que exista la palabra en todos los docs que se pasaron en el arreglo de ids, llamando a *existePalabraEnDoc*. Si en alguno de los docs pasados por el arreglo de ids no existe, no se cumple el AND.

3. Busca múltiples términos en un documento

- *ingresarTerminosUsuario*: permite al usuario ingresar N términos a buscar en un archivo (buscador mediante).
- *buscarTerminosMismoDocumento*: recibe los N términos ingresados por usuario y busca las ocurrencias de esos términos en un doc.

4. Busca una frase

- *buscaFrase*: Función principal de la operación. Recibe la frase que se desea buscar, llama a las demás funciones e imprime el resultado.
- *separaFrase*: Toma la frase y la separa en palabras únicas, las cuales son ingresadas a una lista simple mediante el uso de *palabraToLista* y *creaNodoPalabra*.
- *existeFrase*: Toma la lista de palabras y comienza a revisar el buscador hasta encontrar la cabecera. Luego, comienza a comparar cada término de la frase para ver si están en el mismo documento y posición siguiente del término anterior mediante *comparaPalabras*.
- *comparaPalabras*: Devuelve un flag dependiendo de la situación de la comparación de los términos de la frase. Comienza a comparar cada término con su siguiente, revisando si están en el mismo documento y si son continuos (0: continúa buscando, 1: frase encontrada, 2: uno de los datos no se encuentra).

5. Busca la palabra con más frecuencia en un documento

- *frecuenciaPorDoc*: Función principal de la operación. Recibe el id del documento del cual se quiere saber la frase con más frecuencia e imprime un resultado.
- *maxfrecuencia*: Recorre todo el árbol buscando las palabras que pertenecen al documento pedido, y las compara entre sí guardando la frecuencia más alta y el término que corresponde en una estructura *frecuenciaArchivo*.
- *cuentaID*: Cuenta todas las ocurrencias de una palabra en un documento especificado.

6. Sugiere palabras similares

- *sugerirSimilares*: Función principal de la operación Recibe el término a comparar y llama a la función *buscasimilar*.

- *buscasimilar*: La función recorre todo el buscador y utiliza la función de Levenshtein para calcular la distancia del término en cada nodo. Si se encuentra a una distancia 0<d<=3, lo imprime como sugerencia.
- *buscaSimilarMenorDistancia*: utiliza el mismo algoritmo que la función *buscasimilar*, con la diferencia de que guarda la palabra con menor distancia, es decir, la más parecida al que el usuario buscó.
- *Levenshtein*: Toma dos términos y los compara entre sí, devolviendo un número de cuantas correcciones hay que realizar para que ambos términos sean iguales.
- *minimo*: devuelve el valor mínimo entre dos valores.