Proyecto Final

Priscilla Chacón Conejo

David Prado Orozco

Jafet Valverde Villegas

Álvaro Villalobos Moreira

Tecnológico de Costa Rica

Notas de los autores

Priscilla Chacón Conejo, David Prado Orozco, Jafet Valverde Villegas, Álvaro Villalobos Moreira, Estructuras de Datos, Instituto Tecnológico de Costa Rica

La correspondencia de este trabajo debe ser dirigida a Mauricio Áviles

Estructuras de Datos, Instituto Tecnológico de Costa Rica, Avda. 9, Amón, San José.

Contenido

Introducción

Durante este documento se estará desarrollando el último proyecto de Estructuras de Datos, el cual se realiza con la intención de abarcar y poner en práctica toda la información vista durante el segundo periodo del semestre actual. El proyecto utilizará las clases vistas durante el periodo lectivo para poder realizar una implementación de un programa de consola en modo texto, que deberá cumplir con las siguientes consideraciones:

- Se debe de iniciar el programa imprimiendo un mensaje de bienvenida para el usuario en donde se le expondrá el propósito del programa. Posteriormente se le solicita al usuario ingresar el nombre del archivo con el que desee trabajar. En caso de que el archivo no exista o no sea posible abrirlo, se le presenta al usuario un mensaje de error y se termina el programa. De otro modo, el archivo se abrirá y se procesará línea por línea el texto.
- Cada línea será separada palabra por palabra. Las mayúsculas se sustituirán por minúsculas. Los espacios en blanco y signos de puntuación que separan las palabras no serán tomadas en cuenta como parte de estas. Las palabras deberán ser utilizadas en el idioma español y serán considerados los acentos.
- Cada palabra leída será insertada en el árbol Trie, junto con el número que corresponde a la línea donde aparece. Se tomará en cuenta de que las palabras pueden repetirse a lo largo del texto, por ello cada palabra mostrará una lista de los números de las líneas en las que aparece.
- Una vez cargada las palabras en el Trie se le presenta al usuario un pequeño menú con las siguientes funciones: Consultar por algún prefijo, buscar una palabra, consultar una palabra por cantidad de letras y palabras utilizadas.

- Por último, después que el programa haya terminado de ejecutar una acción, se regresará al menú principal en caso de que se desee realizar otra. De otro modo, el programa presentará una opción de salir del mismo.

Análisis de los problemas.

Problemas por resolver

1. La utilización de una clase que fuera de rápido acceso con la que se pudiera guardar las palabras y hacer uso de índices para facilitar la búsqueda de estas dentro del árbol.
2. Lograr la menor duración al comenzar el programa. Se pretende que al correr el programa tarde unos cuantos segundos en leer las líneas y guardarlas en el diccionario. Además, llenar palabra por palabra el Trie.
3. Abrir un archivo de texto. Se debe de asegurar que el archivo sea abierto de modo correcto porque si no puede ocasionar problemas a la hora de leer los datos que contiene.
4. Ignorar las palabras especiales. Se debe de corregir la apertura del archivo "Ignorar.txt" ya que cada vez que se comprueba una palabra que está dentro de este archivo, se abre, comprueba y cierra. Esto causa que el insertar en el trie se haga de forma lenta.
5. Poder leer acentos en el texto. Al ser un archivo txt se debe de buscar una manera de poder imprimir las palabras con acentos en codeblocks.

6. Crear un método con el que se pueda buscar una palabra específica en una determinada línea. Al ser textos grandes se debe de encontrar la forma de buscar de manera eficiente las palabras a lo largo del txt.
7. Se deberá trabajar con varios procedimientos para realizar un método donde encuentre las palabras y cuente cuantas veces se repite.
8. Realizar un método que cuente el número de letras en las palabras. Se tiene que lograr imprimir todas las palabras con una cierta cantidad de letras.
9. Mostrar las palabras más utilizadas. Para ello se debe de averiguar cuáles son las palabras que más se repiten en el texto.
10. Corregir la fuga de memoria. Al tener muchas palabras que insertar se presenta un error en la compilación del código por el gran uso de memoria.

Resolución de los problemas

1. Se utilizó la clase diccionario a la hora de insertar el texto; esto facilitó el hecho de poder guardar las líneas para poder trabajarlas de forma más eficiente, ya que en esta clase al insertar todos los datos los hace de forma ordenada. Por otro lado, para poder trabajar el programa, en general se utilizó la clase del Trie, ya que está clase facilitaba el uso de los índices para poder localizar con mayor facilidad el lugar donde se encontraba el dato que se buscaba.
2. Se modifico el método insert tanto en la clase del UnsortedArrayList, donde retornaba un error por llaves repetidas, como en el Trie, donde se suprimió el método checkexisting para las llaves, ya que las llaves se insertaron en orden. De este modo el código pudo insertar, de forma más rápida, los datos en las distintas estructuras.

3. Se lee el archivo txt y los datos que se encuentran se guardan en una arraylist. Esto para corregir el error de estar abriendo el archivo cada vez que se necesite el txt de palabras a ignorar para compararlas.
4. Para la apertura del archivo se utilizó la librería iostream, la cual contiene un comando(fstream) que facilita la apertura de los archivos txt.
5. Se utilizó el formato ANSI para la lectura de los acentos en formatos txt.
6. Se utilizó el método getMaches para verificar si la palabra se encuentra en el Trie y el getline para encontrar la línea en la que se encuentra la palabra.
7. Se utiliza un arraylist dentro del trieNode para almacenar las líneas donde se encuentra la palabra, este arraylist también indica la cantidad de veces que aparece al utilizar la función getsize ().
8. Se uso el método length () para comparar el número de letras que contenía cada palabra con el número introducido por el usuario junto con el getlineSize () para encontrar el número de líneas en las que las palabras fueron utilizadas.
9. Se buscan las palabras con más apariciones por medio de varias funciones las cuales obtienen de un arraylist el número de apariciones de todas las palabras en el trie, luego se acomodan las palabras de mayor cantidad a menor cantidad de apariciones y por último solo se muestra el top que decidió el usuario.

Conclusiones

1. Utilizar los insert como estaban implementados en las clases, daba fallos a la hora de ingresar los datos en las diferentes clases, ya que retornaba un error porque se generaban más nodos de los que se necesitaban.
2. El hacer uso de la función getline para insertar datos en el archivo da problemas porque solo permite insertar la primera palabra que encuentra en la oración que se digita, esto sucede porque al encontrar un espacio (" ") deja de insertar.
3. El método de LlenarTrie () presentaba complicaciones, ya que, cuando se necesitaba verificar que una palabra no estuviera dentro del archivo de palabras a ignorar, se abría y cerraba por cada revisión que se necesitaba hacer teniendo como consecuencia que se llenara el Trie de forma lenta.
4. Al implementar un arraylist que contenía la información de txt de palabras a ignorar se complicó la búsqueda de las palabras que contenía el archivo porque el contains revisaba todo el arraylist lo que ocasionaba que el proceso se volviera cansado y lento.
5. Prefixcount no funciona para ver la cantidad de repeticiones de una palabra que se usa en el trie dado que cuando se pone una palabra como prefijo esta podía ser un prefijo de una palabra mayor y así contarse como si la palabra apareciera repetidas veces.

Recomendaciones

1. Al agregar un método dentro del insert que verificara la existencia de una palabra dentro del Trie ayudó a evitar que se crearán más nodos de los necesario. Esto logró que el programa no se cayera y que se pudieran insertar las palabras de forma rápida y eficaz dentro del método.
2. Para poder resolver el problema de la función getline y que no se salga de la ejecución seguir insertando cuando encuentre un espacio se debe de hacer uso de la función ignore (), esta función extrae caracteres de la secuencia de entrada y los descarta, hasta que se hayan extraído n caracteres.
3. Se creó un arrayList el cual contenía la información del archivo de palabras a ignorar, así se evita tener que abrir y cerrar el archivo cada vez que se tuviera que verificar que la palabra no estuviera dentro de esa lista. Esto redujo, considerablemente, el tiempo de ejecución del programa.
4. Se utilizó la clase Trie en vez de un ArrayList, ya que, el Trie no recorre todas las palabras para verificar si estás existen en él o no. Esto colaboró con el hecho de que se pudiera reducir, notable, el tiempo de ejecución de los métodos donde se tenía que usar la lista de palabras a ignorar.
5. Se utilizaron las funciones getLine y getSize las cuales cuentan de manera exacta la cantidad de veces que se repite una palabra dentro del texto. Con este se pudo mejorar la precisión y la efectividad con la que se buscaban las palabras dentro del Trie y así poder retornar el dato exacto de la reiteración de estas.

Referencias

Áviles, M. (2020). *Proyecto Programado – Indización de texto con Tries.* San José.

Stackoverflow. (2014). *getline() No funciona si se uda después de algunas entradas*.

Stackoverflow. (Abril de 2018). *Crear, visualizar, buscar, modificar y eliminar un fichero txt.* Obtenido de https://es.stackoverflow.com/questions/48291/crear-visualizar-buscar-modificar-y-eliminar-un-fichero-txt