Modelos y Conceptos de Programación

Gustavo Gutiérrez

23 de noviembre de 2006

1. Lineamientos

Este proyecto constituye el 20 % de la nota del curso *Modelos y conceptos de programación*. Debe ser entregada a más tardar[1] el día 11 de Diciembre de 2006 a las 12:00 m. La entrega se debe hacer de manera física en un sobre que contenga un documento y el programa en algún medio de almacenamiento.

2. Descripción del problema

Existen diferentes estructuras para modelar diccionarios de palabras, entre estas se cuenta con *árboles alfabéticos*, donde cada nodo del árbol tiene un hijo por cada letra del alfabeto. En este trabajo se debe implementar esta estructura de datos y proveer un conjunto de operaciones que permitan acceso concurrente utilizando las técnicas vistas en clase.

2.1. Árboles alfabéticos

Un árbol alfabético es una estructura de datos que permite almacenar palabras de un lenguaje particular. Para el propósito del presente trabajo solamente se consideraran 26 letras, sin embargo, durante la explicación del proyecto solamente se utilizará el alfabeto $\{a, e, s, t, n\}$. Un árbol alfabético esta compuesto por nodos que tienen un conjunto de hijos, y cada hijo es a su vez un nodo. Para este caso, el número máximo

[1] Se penalizará con 10 % sobre la nota por cada día de retraso.

de hijos es el número de letras del alfabeto. Cada nodo del árbol tiene una bandera que indica si el nodo representa o no la terminación de una palabra.

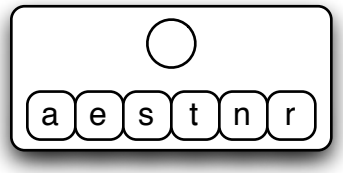

Figura 1: Nodo de un árbol alfabético (suponiendo alfabeto simplificado), si el circulo está coloreado quiere decir que el nodo representa el fin de una palabra.

Las palabras son almacenadas (recursivamente) en el árbol mediante las conexiones de cada nodo con sus hijos. Si la palabra a insertar tiene longitud cero, el nodo es marcado para representar el final de la palabra. De cualquier otra forma, se considera la adición de la primera letra y recursivamente se adiciona el resto de la palabra. Si al insertar la palabra uno de los nodos no existe, el procedimiento de inserción los debe crear.

La Figura 2 muestra la inserción paso a paso de cada una de las letras de la palabra *estar* a partir de un árbol vacío. Por su parte, la Figura 3 muestra el árbol resultante de las inserción de las palabras: *estar, estaran, es, esta, esa, tener.*

Algunas de las propiedades de un árbol alfabético son:

- Sí los hijos de cada nodo están ordenados alfabéticamente es sencillo imprimir recursivamente todas las palabras del árbol alfabéticamente.
- Encontrar las palabras que tengan un prefijo común es sencillo debido a que están agrupadas en un sub-árbol.

2.2. Operaciones a implementar

A continuación se enumeran las operaciones (en forma de funciones) que deben ser implementadas. Estas operaciones pueden ser funciones o métodos de clases. En cualquier caso se debe justificar la decisión.

- {NewAlphaTree ?AT}. Esta función debe retornar un árbol alfabético vacío.

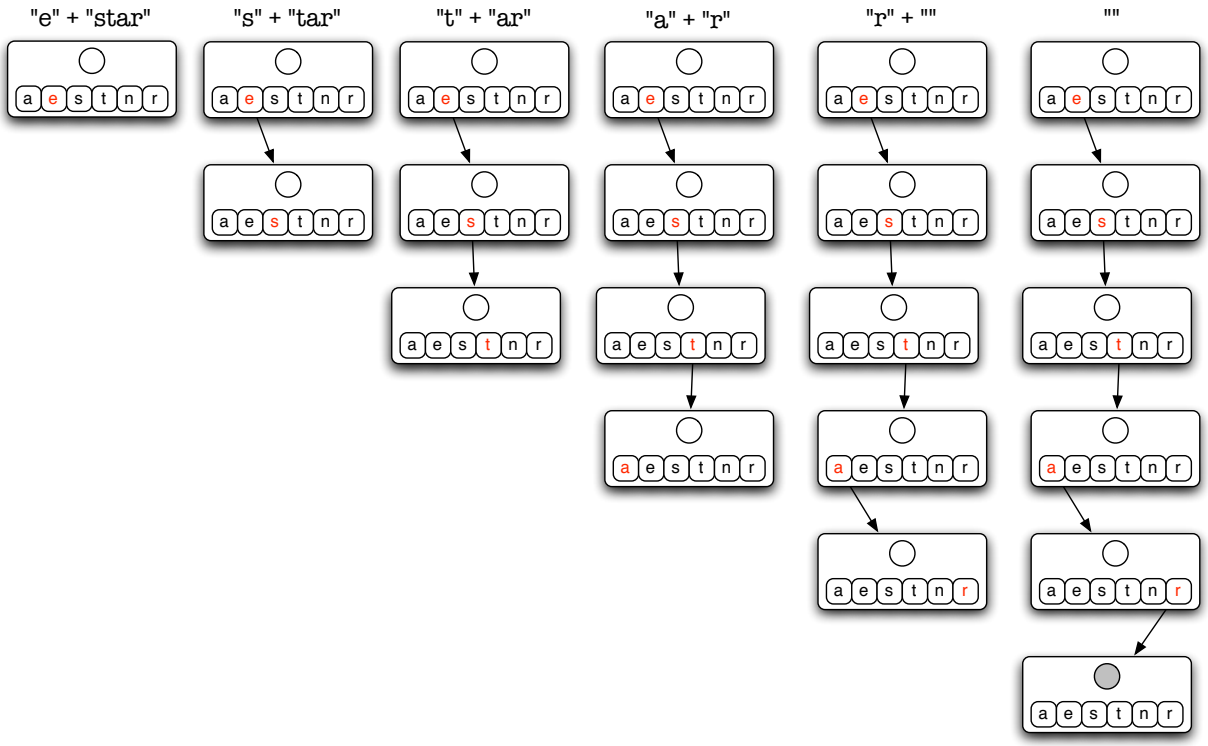

Figura 2: Inserción de la palabra *star*

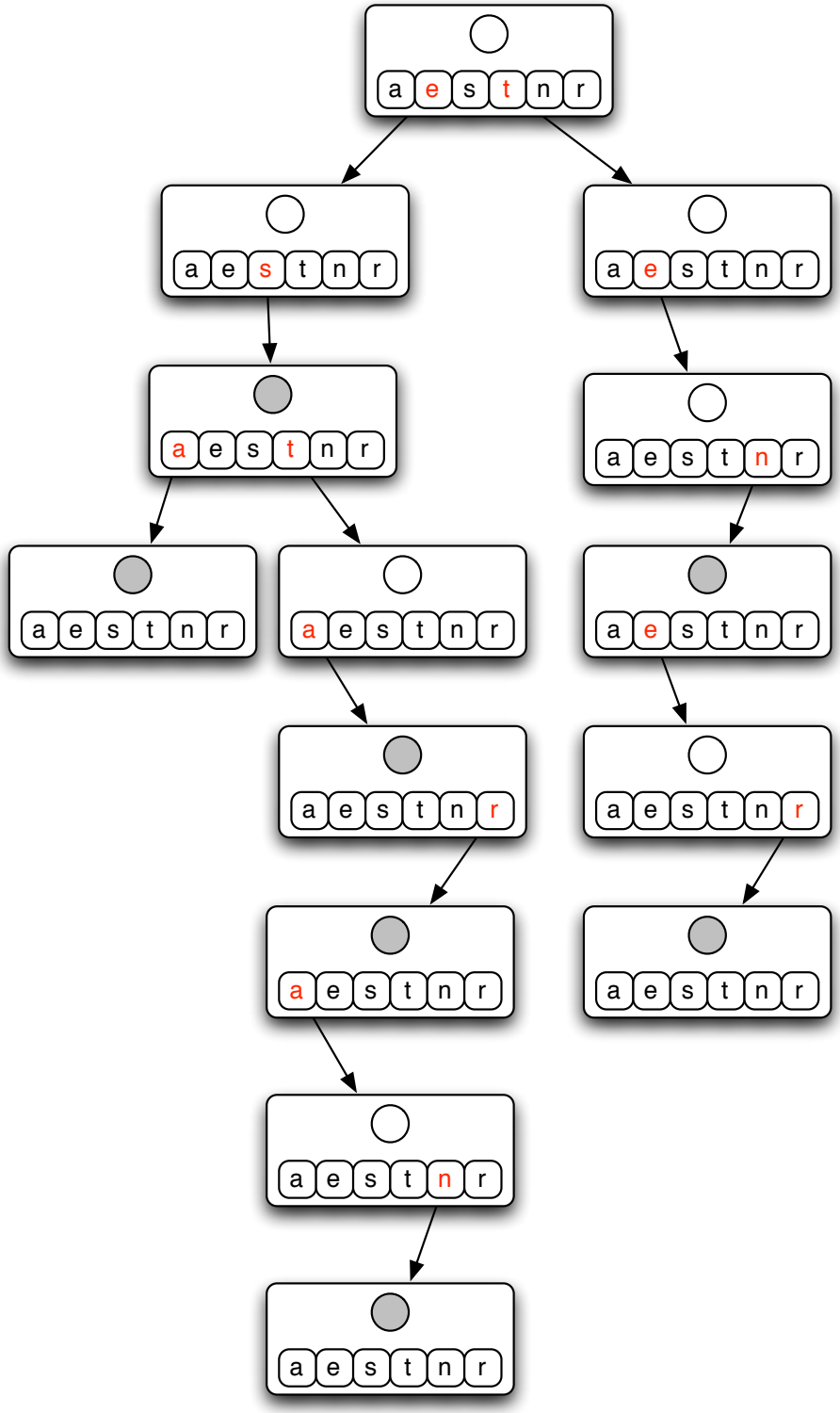

Figura 3: Árbol alfabético con las palabras: *estar, estaran, esta, es, esa, tener, ten.*

- {AddWord AT W ?B}. Adiciona la palabra W al árbol AT si esta no existe y retorna **true**. De lo contrario retorna **false**.
- {IsWord AT W ?B}. Retorna **true** si la palabra W se encuentra en el árbol AT de lo contrario, retorna **false**.
- {DeleteWord AT W ?B}. Borra la palabra W del árbol AT y retorna **true**. Sí la palabra no se encuentra en el árbol, esta función retorna **false**.
- {Print AT}. Imprime en pantalla todo el contenido del diccionario ordenado alfabéticamente.
- {PrintPrefix AT Pr}. Imprime todos las palabras del árbol AT que tengan como prefijo Pr.
- {Interrupt AT T}. Hace que la estructura no sea accesible durante un tiempo T.

2.3. Agentes concurrentes

Sobre el diccionario pueden actuar concurrentemente diferentes agentes de manera concurrente. Los conceptos vistos durante la última parte del curso (concurrencia de estado compartido) deben ser aplicados para garantizar un correcto funcionamiento de la estructura. Recuerde que el objetivo no es eliminar la concurrencia sino permitir tanta concurrencia como sea posible.

Para lograr una interacción interesante de los agentes, cada agente debe tener asociado un retardo de operación y una lista de acciones a realizar. Estas acciones deben comenzar después de haber ocurrido el retardo y no desde su creación.

La forma de modelar los agentes concurrentes debe ser a través de transacciones. Esto debe involucrar la ejecución *justa*, es decir, debe haber un orden en la ejecución de las transacciones que sea coherente con el tiempo en que cada una de ellas fue creada.

2.4. Interfaz gráfica

El programa presentado debe contar con una interfaz gráfica para facilitar su uso. Esta interfaz debe permitir:

- Visualización en todo momento del estado de la estructura que almacena las palabras.
- Visualización o diferenciación de los nodos que en cada momento están siendo accedidos por agentes concurrentes.
- Visualización de los nodos que están siendo accedidos por los diferentes agentes.
- Identificación de los diferentes tipos de agentes que pueden actuar sobre la estructura.
- Creación de nuevos agentes que puedan ejecutar una o más operaciones sobre la estructura.
- Creación aleatoria de agentes de diferentes tipos que realicen operaciones sobre la estructura.

Es necesario contar con una interfaz gráfica para visualizar la actuación de los agentes sobre la estructura. Esta parte es de gran importancia y omitirla implica una gran dificultad al realizar las pruebas durante la sustentación del trabajo.

3. Criterios de calificación

Los siguientes aspectos serán tenidos en cuenta a la hora de la calificación:

Correctitud: Significa que su implementación debe prevenir interferencia entre agentes que actuen concurrentemente. Además se deben garantizar las propiedades de *safety*: en ningún momento el programa o sus estructuras deben alcanzar estados inválidos y *liveness*: todos los agentes que se ejecutaron deben alcanzar su terminación. El documento presentado debe argumentar por qué estas propiedades se mantienen con la implementación propuesta.

Concurrencia: Se debe garantizar acceso concurrente a las estructuras de datos del programa tanto como sea posible sin ir en contra de la correctitud. En el documento presentado se debe argumentar como se logra el grado de concurrencia de la implementación y debe justificar por qué no es posible proveer una mayor concurrencia de las operaciones.

Claridad: Tanto el documento como el código fuente del programa deben ser claros y fáciles de entender. El código debe estar bien documentado y se deben utilizar nombres significativos para los procedimientos. El reporte debe describir

claramente las decisiones tomadas y una justificación de por qué estas llevan a una implementación correcta y concurrente.