Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI5651 – Diseño de Algoritmos I
Septiembre-Diciembre 2022

Proyecto
¿Cómo era la cosa?

Llevados por los valientes *"estudiantes de diseño de algoritmos I"*, considerados héroes por toda la comunidad científica en Titan, la tecnología y el pensamiento avanzaron a pasos agigantados. Aquella civilización que pasó milenios intentando resolver un tablero de Sudoku, finalmente veía más allá de sus horizontes lúdicos. Con la solución del Sudoku vino una revolución industrial, seguido por maquinaria, computadores, sistemas de archivos y más. ¡Y pensar que apenas habían pasado algunas semanas!

Con tantos adelantos, se hizo común que los computistas (una nueva rama de científicos enloquecidos, encargados de transcribir sus ideas a un computador) llevaran un control muy estricto del avance de sus proyectos. Comenzaron, cómo es de esperarse de personas decentes y trabajadoras, dando nombres cada vez menos sinceros a las versiones de sus proyectos:

a) *Versión Final*

b) *Versión Definitiva*

c) *Versión Final, Ahora sí*

d) *Versión Última o me corto una #!@**

e) *Versión Mocho*

f) *Versión beta 0.1*

g) ...

Esta forma saludable de llevar versiones se hizo inmanejable con el tiempo, ocasionando un sinfín de confusiones y esperanzas infundadas. Es por esto que los *"estudiantes de diseño de algoritmos I"* decidieron volver a la acción y desarrollar una solución al problema. Llamaron a su solución: `CELV` (representando el acrónimo **¿Cómo Era La Cosa?**). Muchos les preguntaron por qué la última letra era un acrónimo para *"Cosa"*, pero los héroes se mantuvieron siempre discretos sobre cualquier parte del funcionamiento de su milagrosa y revolucionaria tecnología.

En pro de mejorar la situación de los computistas y con la intención de no frenar los avances tecnológicos durante mucho tiempo, el gobierno central de Titán les ha dado acceso libre a todas las computadoras del globo lunar, durante dos semanas, para que puedan realizar la implementación e instalación de `CELV`. ¿Ustedes qué creen? ¿Podrán lograrlo una vez más?

1 Arbol de sistema de archivos

Definimos un árbol de sistema de archivos como un árbol enraizado, donde cada nodo puede tener una cantidad arbitraria de hijos.

Cada nodo en el árbol puede representar **archivos** o **directorios**. Ambos deben tener un nombre, representado por una cadena de caracteres no vacía de tamaño arbitrario. Además de esto, los archivos (que no deben tener hijos en el árbol) deben tener un contenido, también representado por una cadena de caracteres arbitraria (posiblemente vacía).

Un árbol de sistema de archivos debe ser dinámico, capaz de soportar las operaciones que se describen a continuación (siempre a partir del nodo del árbol sobre el cual fue invocado):

- `crear_dir(nombre)`: Crea un directorio llamado `nombre`.

  Debe reportar un error si ya existe otro elemento (archivo o directorio) con el mismo nombre.

- `crear_archivo(nombre)`: Crea un archivo llamado `nombre`, cuyo contenido será vacío.

  Debe reportar un error si ya existe otro elemento (archivo o directorio) con el mismo nombre.

- `eliminar(nombre)`: Elimina un archivo o directorio llamado `nombre`.

  En el caso de directorios, además elimina todo lo que está contenido en su interior (tanto archivos como directorios, de forma recursiva).

  Debe reportar un error si no existe un elemento (archivo o directorio) con el nombre provisto.

- `leer(nombre)`: Obtiene el contenido del archivo llamado `nombre` en una cadena de caracteres.

  Debe reportar un error si el archivo no existe.

- `escribir(nombre, contenido)`: Asigna `contenido` como contenido al archivo llamado `nombre`.

  Debe reportar un error si el archivo no existe.

- `ir(nombre)`: Navega al directorio llamado `nombre`, hijo del nodo actual.

  Debe reportar un error si el directorio no existe (incluso si existe un archivo con el nombre deseado).

- `ir()`: Navega al directorio padre del nodo actual.

  Debe reportar un error si el directorio no tiene está contendido en otro directorio (es la raíz del árbol).

2 ¿Cómo Era La Cosa?

Si es de interés llevar un historial de versiones para un árbol de archivos, debe de estar disponible el sistema CELV: ¿Cómo Era La Cosa?

Es por esto que debemos agregar algunas operaciones más al árbol de sistema de archivos, como soporte para este nuevo sistema.

2.1 Inicialización

Para inicializar el sistema, debe proveerse una operación:

`celv_iniciar()`

Esta operación prepara el árbol, desde el nodo en donde fue invocado, para el manejo de versiones.

Esta operación debe reportar un error si algún descendiente o ancestro del nodo sobre el que se invoca (incluyéndolo a sí mismo) ya tiene una inicialización. Esto es, no vamos a permitir que existan dos raíces de inicialización tal que una sea ancestro de la otra.

A partir de este punto, toda operación de CELV que sea realizada sobre algún nodo descendiente del nodo donde la inicialización fue hecha, refiere a este manejador de versiones particular.

Es importante notar que un mismo árbol de sistema de archivos puede tener más de una inicialización activa, siempre y cuando los nodos involucrados no tengan relación entre sí (que no sea uno ancestro del otro).

Si el nodo raíz de inicialización es eliminado, el manejador de versiones es eliminado junto con él, por lo que la información de versiones se pierde. Si algún nodo descendiente es eliminado, pero la raíz de inicialización se mantiene intacta, las versiones deben ser preservadas.

2.2 Historia

Para obtener la historia completa de modificaciones, debe proveerse una operación:

`celv_historia()`

Cuando es invocado sobre un nodo, tal que alguno de sus ancestros esté inicializado en CELV, debe realizar un reporte de todas las modificaciones que han sido realizadas sobre el árbol de archivos.

Cada modificación debe ser reportada con la siguiente información:

- Identificador de la versión (autogenerada por la operación realizada)
- Operación realizada
- Argumentos de la operación realizada
- Identificador de la versión anterior a la actual (versión origen)

Si algún nombre o contenido supera los veintitrés (23) caracteres, debe resumirse tomando los primeros diez (10) caracteres, seguido de tres puntos y luego los últimos diez (10) caracteres.

Por ejemplo:

- Texto original: ‘‘`supercalifragilisticoespialidoso`’’
- Texto resumido: ‘‘`supercalif...spialidoso`’’

2.3 Regresar a una versión anterior

Para ir a un estado particular del árbol de sistema de archivos, debe proveerse una operación:

```
celv_vamos(version)
```

Donde `version` es un identificador válido para la versión a la que queremos ir.

Debe reportar un error si no existe tal `version`.

Cuando es invocado sobre un nodo, tal que alguno de sus ancestros esté inicializado en CELV, debemos quedar en el mismo directorio en el que se estaba al invocar la operación. Si el directorio no existe en la versión destino, debemos navegar hasta el ancestro más cercano que sí exista.

Si se realizan modificaciones sobre una versión que no es la más reciente, de todas formas se crearán versiones nuevas (es por esto que en la historia se reporta una versión origen).

2.4 ¡Fusión!

Para fusionar el contenido de dos versiones diferentes, debe proveerse una operación:

```
celv_fusion (versionA, versionB)
```

Donde `versionA` y `versionB` son identificadores válidos para las versiones a fusionar.

Debe reportar un error si no existen `versionA` o `versionB`, así como si ambos identificadores corresponden a la misma version.

Cuando es invocado sobre un nodo, tal que alguno de sus ancestros esté inicializado en CELV, debemos quedar en el mismo directorio en el que se estaba al invocar la operación. Si el directorio no existe en la versión destino, debemos navegar hasta el ancestro más cercano que sí exista.

Al fusionar dos versiones, resultará una nueva versión con la siguiente características:

- Los elementos comunes se mantienen.
- Si un directorio existe en una versión y en la otra no, la fusión tendrá ese directorio.
- Si un archivo existe en una versión y en la otra no, la fusión tendrá ese archivo.
- Si un mismo archivo existe en ambas versiones, pero con contenido diferente, debe reportarse un *conflicto de fusión* y el contenido del archivo en la fusión tendrá un desglose del mismo.

Al ver la historia, usando `celv_historia()`, las fusiones deben incluir ambas versiones involucradas como su versión de origen.

2.4.1 Conflictos de fusión

Si un archivo presenta conflictos de fusión, su contenido debe reflejar los puntos en donde ambas versiones difieren. Para esto, se usará una seguidilla de tres corchetes (`[[[]]]`) para enmarcar las secciones diferentes de la primera versión y tres llaves (`{{{}}}`) para las de la segunda versión.

Por ejemplo:

- En la primera versión: ‘‘`aabbcc`’’
- En la segunda versión: ‘‘`abaccdd`’’
- En la versión fusionada: ‘‘`a[[[a]]]b[[[b]]]{{{a}}}cc{{{dd}}}`’’’

Los cambios a realizar entre una versión a otra deben ser los mínimos posibles, considerando inserciones, eliminaciones y modificaciones (similar a la distancia de edición entre dos cadenas de caracteres).

2.5 Espejo al mundo real

Para probar su sistema, debe proveerse una operación:

```
celv_importar(directorio)
```

Donde `directorio` es una dirección (*path*) del sistema de archivos real del usuario. Esta operación debe poblar un árbol de sistema de archivos con una copia del contenido de ese directorio y todos sus sub–directorios, tomando en cuenta únicamente directorios y archivos (no considerar otro elementos, como enlaces simbólicos).

Debe reportar un error si `directorio` no existe en el sistema de archivos del usuario o no tiene permisos suficientes para acceder a él.

Es importante notar, que esta es la **única** operación que tiene permitido hacer una llamada al sistema. El resto de las operaciones sobre el árbol de sistema de archivos deben ser hechas en memoria y sin realizar llamadas al sistema más allá de impresiones al terminal.

3 Entrega

Todos los proyectos deben estar en un repositorio privado en Github, cuyos colaboradores sean los miembro del equipo y el profesor del curso.

- Usuario de Github de Ricardo Monascal: rmonascal

La entrega será el día Jueves, 15 de Diciembre (semana 10), a las 11:59pm VET. A la hora de la entrega, se hará `pull` del repositorio y se corregirá lo que exista ahí hasta ese punto.

3.1 Implementación

Su entrega debe incluir la implementación completa para el árbol de sistema de archivos y el manejador de versiones CELV.

Además, debe incluir un cliente que permita interactuar con esta estructura de datos por medio de un menú de opciones.

3.2 Informe

Su proyecto debe incluir un manual de uso y una explicación detallada sobre la implementación: las estructuras de datos y algoritmos utilizados, así como un análisis de recursos para cada operación (en notación asintótica), tanto en tiempo como en espacio. Haga énfasis particular en las técnicas que utilizó para hacer cada componente lo más eficiente posible.

Se recomienda que esta sección del informe se ubique en un archivo `README.md` (escrito en el lenguaje *Markdown*), de tal forma que se muestre correctamente en la página inicial de su repositorio en Github.