Facultad de Ingeniería de la Universidad de Buenos Aires (FIUBA)

Trabajo Práctico 2: Los Simuladores, informe sobre el avance del Programa

Algoritmos y Estructura de Datos

KLÖCKNER Martin, CETTOUR Ignacio, WILVERHT ROHR Camila

Schmidt Gustavo Adolfo

camiluanahi@gmail.com

21/06/2024

Los Simuladores

Cuestionario

1) ¿Qué es un **svn**?
2) ¿Qué es **git**?
3) ¿Qué es **Github**?
4) ¿Qué es un **valgrind**?

Para empezar, **svn**, también conocido como **Subversion,** es un sistema de control de versiones libre, de código fuente abierto y por lo tanto el directorio administrativo de la copia de trabajo. Subversion maneja ficheros y directorios a través del tiempo. Hay un árbol de ficheros en un repositorio central. El repositorio es como un servidor de ficheros ordinario, excepto porque recuerda todos los cambios hechos a sus ficheros y directorios. Esto le permite recuperar versiones antiguas de sus datos, o examinar el historial de cambios de los mismos. Además, puede acceder al repositorio a través de redes, lo que le permite ser usado por personas que se encuentran en distintos ordenadores y así progresar más rápidamente sin un único conducto por el cual deban pasar todas las modificaciones. Puesto que el trabajo se encuentra bajo el control de versiones, no hay razón para temer porque la calidad del mismo vaya a verse afectada por la pérdida de ese conducto único. Ya que sí se ha hecho un cambio incorrecto a los datos, simplemente deshaga ese cambio.

Tener en cuenta que Subversion es un sistema general que puede ser usado solamente para administrar cualquier conjunto de ficheros, y no para la administración de árboles de código fuente con características específicas del desarrollo de software.

Por otro lado tenemos a **git.** Git maneja sus datos como una secuencia de copias instantáneas y de un sistema de archivos miniatura. Cada vez que confirma un cambio, o guardas el estado de tu proyecto en Git, se toma una “foto” del aspecto de todos los archivos que correspondan al que los esta editando en ese momento y guarda una referencia a esa copia instantánea. Para ser eficiente, si los archivos no se han modificado, Git no almacena el archivo de nuevo, sino un enlace al archivo anterior idéntico que ya tiene almacenado.

Git almacena y maneja la información de forma muy diferente a esos otros sistemas. La principal diferencia entre Git y cualquier otro VCS (como Subversion) es la forma en la que manejan sus datos. Conceptualmente, la mayoría de los otros sistemas almacenan la información como una lista de cambios en los archivos que a su vez manejan la información que almacenan como un conjunto de archivos y las modificaciones hechas a cada uno de ellos a través del tiempo.

La mayoría de las operaciones en Git sólo necesitan archivos y recursos locales para funcionar, no es necesaria la red. Es decir, se tiene toda la historia del proyecto ahí mismo, en el disco local, y por ello la mayoría de las operaciones son inmediatas. Esto también significa que hay muy poco que no puedes hacer si se está desconectado a la red o sin VPN, ya que se pueden confirmar los cambios hasta conseguir una conexión de red para subirlos.

Tampoco puedes perder información durante la transmisión o sufrir corrupción de archivos sin que Git sea capaz de detectarlo y avisarlo.

Git tiene tres estados principales en los que se pueden encontrar los archivos: confirmado (committed), modificado (modified), y preparado (staged).

- Confirmado: significa que los datos están almacenados de manera segura en la base de datos local del programador.
- Modificado: significa que ha modificado el archivo pero todavía no lo ha confirmado a la base de datos del programador.
- Preparado: significa que ha marcado un archivo modificado en su versión actual para que vaya en la próxima confirmación.

Esto nos lleva a las tres secciones principales de un proyecto de Git: El directorio de Git (directorio Git), el directorio de trabajo (working directorio), y el área de preparación (staging area).

- El directorio de Git es donde se almacenan los metadatos y la base de datos de objetos para un proyecto. Además es lo que se copia cuando se clona un repositorio desde otra computadora.

- El directorio de trabajo es una copia de una versión del proyecto. Estos archivos se sacan de la base de datos comprimida en el directorio de Git, y se colocan en disco para que se los puedas usar o modificar.
- El área de preparación es un archivo, generalmente contenido en el directorio de Git del programador. Esta almacena información acerca de lo que va a ir en la próxima confirmación del programador.

La línea de comandos es el único lugar en donde puedes ejecutar todos los comandos de Git, a través de la Terminal, ya que de esta forma se puede asegurar que todos los usuarios tendrán las herramientas de línea de comandos instaladas y disponibles.

Ahora se comentará sobre lo que es **Github.** Este es uno de las principales plataformas para crear proyectos abiertos de herramientas y aplicaciones, y se caracteriza sobre todo por sus funciones colaborativas que ayudan a que todos puedan aportar para mejorar el código.

Como buen repositorio, el código de los proyectos que sean públicos, puede ser descargado y revisado por cualquier usuario, lo que ayuda a mejorar el producto y crear ramificaciones a partir de él. Para evitar que el código se vea, también pueden crearse proyectos privados.

Github es un portal creado para alojar el código de las aplicaciones de cualquier desarrollador, para que el usuario pueda descargar la aplicación y entrar al perfil del dueño de la aplicación para leer sobre ella o colaborar con el desarrollo de esta. Además utiliza el sistema de control de versiones Git.

Github también ofrece una serie de herramientas propias con las que complementar las ventajas que ya tiene el sistema Git de por sí solo. Por ejemplo, se puede crear una Wiki para cada proyecto, de forma que se ofrece toda la información sobre él y anota todos los cambios de las diferentes versiones.

También tiene un sistema de seguimiento de problemas, para que otras personas puedan hacer mejoras, sugerencias y optimizaciones en los proyectos. Ofrece también una herramienta de revisión de código, de forma que no sólo se pueda mirar el código fuente de una herramienta, sino que también

se pueden dejar anotaciones para que su creador o el usuario las puedan revisar. Se pueden crear discusiones también alrededor de estas anotaciones para mejorar y optimizar el código.

Por último, al igual que Git, Github tiene una línea de comandos a la cual se accede por la Terminal para obtener su máximo potencial. Sin embargo, esto también es posible de forma gráfica.

Por último, se hablará sobre **Valgrind.** Valgrind es un sistema para debuggear y hacer profiling (investigar el comportamiento de un programa utilizando información obtenida durante su ejecución) sobre programas en Linux. Es de licencia GPL.

Valgrind puede detectar automáticamente muchos errores de manejo de memoria, (como por ejemplo memory leaks) para obtener programas más estables.

Además, Valgrind funciona con programas escritos en cualquier lenguaje. Al trabajar directamente sobre los binarios, puede analizar programas compilados y programados en cualquier lenguaje de programación, sean compilados, pre-compilados o interpretados.

Valgrind esta pensado para funcionar principalmente con programas escritos en C y C++, ya que estos son los lenguajes que introducen este tipo de bugs con mayor facilidad. Pero puede ser usado, por ejemplo, en programas escritos incluso en varios lenguajes diferentes (C, C++, Java, Perl, Python, assembly, Fortran, y muchos otros).

Complicaciones a lo largo del Trabajo Práctico 2

La primer complicación que se tuvo fue la de aprender a usar Github para poder trabajar de la manera más eficiente con el grupo. Mediante la prueba y error y con la ayuda de los conocimientos que brindaron algunos integrantes, sumado a las investigaciones en páginas de Google y videos de Youtube sobre cómo utilizar Github, el equipo pudo avanzar se manera eficiente sobre el código.

Otra de las complicaciones que tuvo el equipo, fue con la comprensión de las consignas. Gracias a que hubo un encuentro con el profesor, las dudas que surgieron fueron resueltas con claridad.

También hubo una complicación que fue al momento de leer el archivo .cvs, donde el problema principal era el cómo leer los datos, y qué hacer cuando faltaba uno. Esto fue solucionado

gracias a que en vez de leer hasta una “**,**”, si se encontraba una “**”**”, entonces se optó por seguir leyendo hasta encontrar la otra “**”**”.

Manual del Usuario

Sistemas Operativos Basados en Unix

Clonar el repositorio con el siguiente comando:

$ git clone https://github.com/mjkloeckner/CB100-tp2.git

Luego navegue al directorio del repositorio creado:

$ cd CB100-tp2

Compile la aplicación utilizando la herramienta *make.*

$ make

Finalmente corra la aplicación con el siguiente comando:

./main

En el caso de Windows puede instalar WSL , para hacerlo puede seguir la guía oficial de Microsoft . Una vez instalado WSL siga los pasos para los Sistemas Operativos Basados en UNIX

Manual del Programador

En **Menu.cpp** en la función **“cargarDatos()”**, se lee un archivo llamado **"paradas-de-colectivo.csv"** y se recorren su datos para guardarlos en la lista de Barrios, en las listas de Paradas y en los vectores de Líneas. Luego en **“mostrarMenu()”** se imprimen por pantalla las opciones para el usuario y en base a la que elija se ejecuta una de ellas hasta que el usuario ingrese **“q”**.

Las opciones son:

- ***`1` Cantidad de paradas por barrio.***
- ***`2` Parada mas cercana a una coordenada.***
- ***`3` Listado de paradas de una linea de colectivo.***
- ***`4` Listado de cantidad de paradas por linea de colectivo.***
- ***`5` Listado de paradas de una linea mas cercano a una coordenada.***
- ***`b` Imprimir barrios.***
- ***`q` Salir.***

La **opción 1** imprime por pantalla la cantidad de Paradas por Barrio que hay en el archivo, recorriendo una lista de Barrios accediendo a la función **“getSizeListaDeParadas()”** para ir obteniendo la cantidad de Paradas que hay por barrio.

Continuando con la **opción 2**, se pide que el usuario ingrese por teclado una coordenada (latitud y longitud) para poder imprimir la Parada más cercana a ella. Para esto, va recorriendo toda la lista de Barrios obteniendo la parada más cercana a las coordenadas dadas de ese barrio usando la función **“paradaMasCercana()”** para luego ir comparando la de mayor cercanía. Una vez recorrida la lista de Barrios, se imprime por pantalla la dirección, la latitud, la longitud, y la distancia en kilómetros de la parada encontrada.

Avanzando tenemos la **opción 3**. En este caso, se le pide al usuario ingresar por teclado una Línea de colectivo para buscar en qué paradas pasa esa Línea. Para ello se llama a la función **“buscarParadas(barrios, linea)”,** la cuál recibe por parámetros una lista de Barrios y un entero (el número de la línea). Esta función devuelve una lista de Paradas y luego se imprime por pantalla la

cantidad de paradas que se encontró con esa linea y las direcciones de las paradas. En caso de no haber Paradas con esa linea, se imprime por pantalla **“no hay paradas de la linea indicada”.**

A continuación tenemos la **opción 4**. Esta es similar a la opción 3, con la diferencia de que no recibe ninguna línea, sino que primero crea un vector con todas las líneas que hay en el archivo recorriendo toda la lista de Barrios y todas las listas de Paradas por barrio y también sus líneas. Luego se imprime por pantalla una Línea y la cantidad de paradas que hay por línea.

También tenemos a la **opción 5**, la cual dado un Barrio, una Línea, una latitud y una longitud, imprime por pantalla las paradas que hay en el Barrio que tengan a la Línea por orden de cercanía con respecto a la latitud y a la longitud ingresadas. Mediante a la función:

“ordenarParadasPorDistanciaACoordenada(paradasDeLaLinea, lon, lat)”

Recibe una lista de Paradas que tienen la Línea, una longitud y una latitud. Para ordenar esta misma lista por distancia de menor a mayor, con respecto a la latitud y longitud. Saliendo de esta función, se imprime esta lista por pantalla mostrando las direcciones, las coordenadas, y la distancia en kilómetros.

Anteúltimo tenemos la **opción b** que es una función que imprime por pantalla los nombres de todos los Barrios que hay en la lista de Barrios.

Por último esta la **opción q** que sólo funciona cuando el usuario ve el menú por pantalla, consiguiendo que se termine la ejecución del programa.

Bibliografías

https://www.unc.edu.ar/sites/default/files/SVN-BOOK-GENERALIDADES.pdf

https://www.git-scm.com/book/es/v2/Inicio---Sobre-el-Control-de-Versiones-La-L%C3%ADnea-de-Comandos

https://www.git-scm.com/book/es/v2/Inicio---Sobre-el-Control-de-Versiones-Fundamentos-de-Git

https://www.xataka.com/basics/que-github-que-que-le-ofrece-a-desarrolladores

https://wiki.cs.famaf.unc.edu.ar/lib/exe/fetch.php?media=algo2:main:valgrind.pdf

https://github.com/mjkloeckner/CB100-tp2?tab=readme-ov-file#inicio-r%C3%A1pido