SoED

Practica6

Generado por Doxygen 1.7.5

Viernes, 13 de Diciembre de 2013 16:40:01

Índice general

Capítulo 1

SoED

Versión

v0

Autor

Juan F. Huete

1.1. Introducción

En esta práctica nuestro objetivo es trabajar con una estructura gerárquica, en concreto un árbol general (clase tree<T>). En ella el alumno practicará los conceptos básicos de árboles, realizará recorridos, y en general profundizará en el uso de las estructuras de datos gerárquicas complejas como es el tipo tree (definido en tree.h)

1.2. Sistema Operativo de Estructuras de Datos (SoED)

Toda la información, ya sean textos, imágenes, o información para la configuración del sistema, se almacena en "ficheros", que a su vez se guardan en directorios. Los ficheros son la estructura empleada por el sistema operativo para almacenar información en un dispositivo físico como un disco duro, un disquete, un CD-ROM.

En esta práctica implementaremos un simulador para un sistema de archivos, llamado SoED (Sistema operativo Estructuras de Datos). Nuestro sistema será multiusuario, por lo que para gestionarlo necesitaremos de dos partes esenciales, por un lado un gestor de usuarios y por otro un árbol de directorios.

1.2.1. Gestor de Usuarios

Consideraremos un sistema multiusuario, por lo tanto, la tarea de añadir, modificar, eliminar y en general administrar usuarios se convierte en algo no solo rutinario, sino

importante. Los usuarios en SoED se identifican por un identificador de usuario, UserID. Cada usuario solo podra ver aquellas entradas de las que sea propietario (las haya generado).

Podemos distinguir dos tipos esenciales de usuarios:

- usuario root (superusuario) que tiene privilegios sobre todo el sistema, pudiendo acceder a todos los directorios/archivos del sistema, independientemente de su propietario.
- usuarios normales. Solo tienen privilegios sobre las entradas que cuelgan a partir de su directorio de trabajo.

1.2.2. Arbol de Directorios

El árbol de directorios es una herramienta que nos ayuda a saber dónde se encuentra un archivo en disco. En el árbol se distinguen dos tipos de entradas (ver figura):

- Archivos o ficheros ordinarios: Son los elementos que contienen a indormación del usuario (fotos, documentos, etc.)
- Directorios (o carpetas): Es un archivo especial que contiene otras entradas (ficheros/directorios)

Para cada entrada (archivo/directorio) debemos conocer los siguientes datos,

- **Nombre** el cual debe cumplir unas ciertas reglas:

1. Un nombre de entrada es un string de caracteres. Se puede utilizar cualquier carácter que sean letras {a,...,z,A,...,Z}, números {0,...,9} y el caracter '.'
2. Las letras mayúsculas y minúsculas se consideran diferentes, y por lo tanto no es lo mismo carta.txt que Carta.txt ó carta.Txt.
3. Todos los nombres de directorios del sistema acaban con el caracter de barra inclinada '/'.

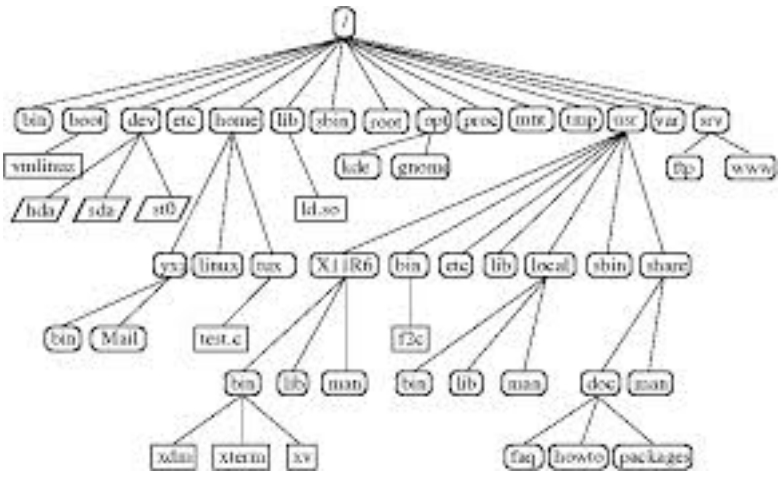
- **Propietario:** Identificador del usuario al que pertenece la entrada (quien lo creó). Un usuario sólo podrá acceder a las entradas que cuelgan de su directorio de trabajo (salvo el usuario root, que tiene acceso a todas las entradas en el sistema).

- **Tama:** Nos indica el tamaño de la entrada. En nuestro caso, como no existen realmente los archivos haremos las siguientes suposiciones:

  1. Un directorio tiene tamaño cero
  2. Un archivo tendrá un tamaño en Kbytes que será conocido en el momento de creación del mismo.

Como es normal, en nuestro sistema no permitiremos que haya dos usuarios con el mismo UserID, ni tampoco que hayas dos entradas dentro de un mismo directorio con el mismo nombre. Esto nos permitirá asegurar que no tenemos dos entradas iguales dentro del sistema

En un arbol de directorio, visto de forma global (en la siguiente imagen podemos ver el árbol de directorios de linux), se verifica que todas las entradas cuelgan de un directorio principal llamado "raíz", que se representa como "/". El directorio raíz es la base para todo el arbol, es allí donde están contenidos todos las entradas del sistema, este directorio es propiedad del usuario "root". El camino completo hasta la raíz se representa como una concatenación de nombres de ficheros/directorios, separadas por la barra inclinada (/). Por ejemplo, "/home/luis/ED/practicas/practica2/pract2.txt"



En todo momento, para cada usuario podemos distinguir dos directorios:

- Directorio raíz. Es el directorio del que cuelgan todos las entradas de las que es propietario. Por definición, el directorio raíz para el usuario "root" es la raíz del árbol, "/". Para el resto de los usuarios, su directorio raíz cuelga de un directorio que llamaremos "home/". El directorio raíz del usuario se creará cuando éste sea añadido al sistema, y tendrá como nombre el UserID. Así, al añadir los usario "pedro" y "sara" al sistema, sus directorios raíz serán "/home/pedro/" y "/home/-sara/", respectivamente. El directorio raíz del usuario se puede identificar como "∼/"

- Directorio de trabajo, que nos indica en que directorio se encuentra actualmente el usuario. El usuario sólo puede ver los elementos que hay en su directorio de trabajo. Sin embargo el sistema proporcionará comandos que permitan a un usuario moverse libremente por todos los directorios de los que es propietario, cambiando así su directorio de trabajo. El directorio de trabajo se denotará como "./"

1.3. Trabajando con SoED

Para gestionar nuestro sistema operativo, debemos de permitir añadir y borrar usuarios, así como que un usuario pueda añadir y borrar ficheros y/o directorios, que un usario se mueva por el árbol de directorios etc. Todos estos métodos los podemos encontrar en la especificación del la clase SoED, proporcionada en el fichero SoED.h

Además de la propia clase, SoED, podemos destacar distintos el tipo de datos:

- SoED::path -> string que nos permite referenciar a un camino en el sistema. Como hemos dicho, un path puede ser

  1. /home/usuario/dir1/dir11/ si es un nombre de directorio
  2. /home/usario/dir1/fichero.dat si es un nombre de fichero. Como se ha comentado, cuando si consideramos un camino completo tenemos tres alternativas para describirlo. Supongamos que consideramos el usuario "pedro", que se encuentra en directorio de trabajo "/home/pedro/ED/", las tres alternativas siguietnes harían referencia al directorio DIR, que se encontraría como subdirectorio de ED:

- dir ="/home/pedro/ED/DIR/"
- dir ="∼/ED/DIR/"
- dir ="./DIR/"

- SoED::userID -> String que representa un identificador de usuario

- SoED::error -> Representa un codigo de error Los códigos de error son -- 0. operación correcta -- 1. camino incorrecto (no existe) -- 2. un usario no tiene permiso para acceder al camino -- 3. directorio vacío

- SoED::size_type -> Representa el tamaño de los ficheros.

A modo de ilustración, el siguiente trozo de código nos permite comprender las funcionalidades de nuestro sistema operativo.

```
//Gestion del sistema

SoED sistema;
SoED::path camino;
list<SoED::path> l;
list<SoED::path>::iterator it;

sistema.useradd("pedro");
sistema.useradd("sara");

sistema.mkdir("pedro","./ED/");
sistema.cd("pedro","./ED/");
sistema.mkdir("pedro","./DIR");
sistema.mkdir("pedro","./practica6/");
sistema.cd("pedro","practica6/");
sistema.mkFile("pedro","datos.txt",100);
sistema.mkFile("pedro","codigo.cpp",30);
camino = sistema.pwd("pedro");
sistema.mkdir("pedro","/home/pedro/nuevo/");
sistema.cd("pedro");
sistema.rm("pedro","/home/pedro/ED/practica6/datos.txt");
.....
l = sistema.ls("pedro");
for (it =l.begin();it!=l.end();++it)
  cout << *it<< endl;

sistema.mkdir("sara","./ED/");
sistema.mkdir("sara","./OTRA/");
sistema.cd("sara","./ED/");
sistema.mkFile("sara","misdatos.txt",100);

sistema.mv("sara","/home/sara/ED/carta.txt","/home/sara/");
sistema.mv("sara","/home/sara/ED/","/home/sara/OTRA/");
sistema.cp("sara","home/sara/carta.txt","./");
sistam.du("sara","~/");
.....

l = sistema.locate("sara","ED");
...
l=sistema.locate("root","ED");
l = sistema.tree("root","/");

sistema.who();
sistema.du("root","/");
```

1.4. Generar la Documentación.

Al igual que en la práctica anterior la documentación se entrega mediante un fichero pdf, así como mediante un fichero zip que contiene todos los fuentes junto a los archivos necesarios para generar la documentación (en latex y html). Para generar los ficheros html con la documentación de la misma es suficiente con ejecutar desde la línea de comando

```
doxygen dox_SoED
```

Como resultado nos genera dos directorios, uno con la documentación en html y el otro con la documentación en latex. Para generar la documentación en pdf podemos ejecutar

```
cd latex
make
```

Al hacer make se ejecuta una llamada al programa latex (si lo tenemos instalado) que como salida nos genera el fichero refman.pdf

Se entregan los siguientes ficheros

- tree.h Especificación de tree<T> y tree<T>::node
- tree.hxx Implementación del tree<T>
- nodetree.hxx Implementación del tree<T>::node
- SoED.h Especificación del Sistema Operativo de Estructuras de Datos

Pasamos a detallar cada una de las partes de la práctica.

1.5. Representación de SoED

En primer lugar, veremos lo que será una entrada del directorio. Para ello consideramos la siguiente estructura

```
struct entrada{
  string nombre; //nombre fichero/directorio
  size_type tama; //tama del fichero
  userID propietario;
  };
```

Donde **nombre** representa en nombre de la entrada, en este caso, cuando es un directorio, el último caracter debe ser '/'.

El árbol de directorio se representará como un tree<entrada>

```
tree<entrada> arbol;
```

Cuando se construye el sistema (constructor), se generarán tanto el directorio raíz "/" como el directorio "/home/", que serán propiedad del usuario root.

Para cada usuario del sistema debemos conocer dos elementos de información, su directorio raíz y su directorio de trabajo (o directorio actual).Para la gestión de esta información utilizaremos un map (diccionario), cuya clave es el nombre de usuario y cuya definición será un par que contiene estos dos directorios. Aunque un usuario puede ver un directorio como una cadena (string), internamente, un directorio no es más que un nodo del árbol. Por tanto, la estructura que nos permite gestionar a los usuarios es

```
map<string, pair< tree<entrada>::node , tree<entrada>::node >  > users;
```

donde para un usuario x,

```
users[x].first hace referencia al directorio por defecto
users[x].second hace referencia al directorio actual.
```

1.6. SE PIDE

En concreto se pide implementar todos los métodos asociados al SoED y entregar un fichero donde se pruebe su correcto funcionamiento.

1.6.1. A ENTREGAR

El alumno debe entregar los siguientes ficheros, con las correciones necesarias para poder trabajar

- SoED.h Especificación del TDA diccionario.
- SoED.hxx segunda versión del diccionario.
- prueba.cpp fichero de prueba del diccionario

Dicha entrega se debe realizar antes del viernes 24 de Enero, a las 20:00 horas (pm).

Capítulo 2

Lista de tareas pendientes

Clase SoED

Tareas a realizar: El alumno debera implementar la clase SoED, juntos con el codigo de prueba de los distintos metodos.

Capítulo 3

Índice de clases

3.1. Lista de clases

Lista de las clases, estructuras, uniones e interfaces con una breve descripción:

Capítulo 4

Indice de archivos

4.1. Lista de archivos

Lista de todos los archivos documentados y con descripciones breves:

Capítulo 5

Documentación de las clases

5.1. Referencia de la Clase tree::const_inorderiterator

Métodos públicos

- **const_inorderiterator** (node n)
- bool **operator!=** (const const_inorderiterator &i) const
- const T & **operator*** () const
- const_inorderiterator & **operator++** ()
- bool **operator==** (const const_inorderiterator &i) const

La documentación para esta clase fue generada a partir del siguiente fichero:

- tree.h

5.2. Referencia de la Clase tree::const_leveliterator

Métodos públicos

- **const_leveliterator** (node n)
- bool **operator!=** (const const_leveliterator &i) const
- const T & **operator*** () const
- const_leveliterator & **operator++** ()
- bool **operator==** (const const_leveliterator &i) const

La documentación para esta clase fue generada a partir del siguiente fichero:

- tree.h

5.3. Referencia de la Clase tree::const_postorderiterator

Métodos públicos

- **const_postorderiterator** (node n)
- bool **operator!=** (const const_postorderiterator &i) const
- const T & **operator*** () const
- const_postorderiterator & **operator++** ()
- bool **operator==** (const const_postorderiterator &i) const

La documentación para esta clase fue generada a partir del siguiente fichero:

- tree.h

5.4. Referencia de la Clase tree::const_preorderiterator

Métodos públicos

- **const_preorderiterator** (node n)
- bool **operator!=** (const const_preorderiterator &i) const
- const T & **operator*** () const
- const_preorderiterator & **operator++** ()
- bool **operator==** (const const_preorderiterator &i) const

La documentación para esta clase fue generada a partir del siguiente fichero:

- tree.h

5.5. Referencia de la Estructura SoED::entrada

Atributos públicos

- string **nombre**
- userID **propietario**
- size_type **tama**

La documentación para esta estructura fue generada a partir del siguiente fichero:

- SoED.h

5.6. Referencia de la Clase tree::inorderiterator

```
#include <tree.h>
```

Métodos públicos

- **inorderiterator** (node n)
- bool **operator!=** (const inorderiterator &i) const
- T & **operator*** ()
- inorderiterator & **operator++** ()
- bool **operator==** (const inorderiterator &i) const

5.6.1. Descripción detallada

Clase iterator para recorrer el árbol en Inorder

La documentación para esta clase fue generada a partir del siguiente fichero:

- tree.h

5.7. Referencia de la Clase tree::leveliterator

`#include <tree.h>`

Métodos públicos

- **leveliterator** (node n)
- bool **operator!=** (const leveliterator &i) const
- T & **operator*** ()
- leveliterator & **operator++** ()
- bool **operator==** (const leveliterator &i) const

5.7.1. Descripción detallada

Clase iterator para recorrer el árbol por niveles

La documentación para esta clase fue generada a partir del siguiente fichero:

- tree.h

5.8. Referencia de la Clase tree::node

Métodos públicos

- node left () const

  Devuelve el hizo izquierdo del nodo receptor.
- node next_sibling () const

Devuelve el hermano derecho del nodo receptor.

- node ()

  Constructor primitivo.

- node (const node &n)

  Constructor copia.

- bool null () const

  Devuelve si el nodo es nulo.

- bool operator!= (const node &n) const

  Operador de comparación de desigualdad.

- T & operator* ()

  Devuelve la etiqueta del nodo.

- const T & operator* () const

  Devuelve la etiqueta del nodo.

- node & operator= (const node &n)

  Operador de asignación.

- bool operator== (const node &n) const

  Operador de comparación de igualdad.

- node parent () const

  Devuelve el padre del nodo receptor.

- void setlabel (const T &e)

  Modifica la etiqueta.

Amigas

- class **tree**< **T** >

5.8.1. Descripción detallada

Descripción

Representa a los nodos del árbol

5.8.2. Documentación de las funciones miembro

5.8.2.1. node tree::node::left ( ) const [inline]

Devuelve el hizo izquierdo del nodo receptor.

Precondición

El nodo receptor no puede ser nulo

5.8.2.2. node tree::node::next_sibling ( ) const [inline]

Devuelve el hermano derecho del nodo receptor.

Precondición

El nodo receptor no puede ser nulo

5.8.2.3. bool tree::node::operator!= (const node & *n*) const [inline]

Operador de comparación de desigualdad.

Parámetros

n	el nodo con el que se compara

5.8.2.4. T& tree::node::operator∗ ( ) [inline]

Devuelve la etiqueta del nodo.

Precondición

Si se usa como consultor, !n.Nulo()

5.8.2.5. const T& tree::node::operator∗ ( ) const [inline]

Devuelve la etiqueta del nodo.

Precondición

El nodo receptor no puede ser nulo

5.8.2.6. node& tree::node::operator= (const node & *n*) [inline]

Operador de asignación.

Parámetros

n	el nodo a asignar

5.8.2.7. bool tree::node::operator== (const node & *n*) const [inline]

Operador de comparación de igualdad.

Parámetros

n	el nodo con el que se compara

5.8.2.8. node tree::node::parent ( ) const [inline]

Devuelve el padre del nodo receptor.

Precondición

El nodo receptor no puede ser nulo

La documentación para esta clase fue generada a partir del siguiente fichero:

- tree.h

5.9. Referencia de la Clase tree::nodewrapper

Métodos públicos

- **nodewrapper** (const T &)

Atributos públicos

- T **etiqueta**
- node **hermanodcha**
- node **izda**
- node **pad**

5.9.1. Descripción detallada

TDA nodo. Modela los nodos del árbol.

La documentación para esta clase fue generada a partir del siguiente fichero:

- tree.h

5.10. Referencia de la Clase tree::postorderiterator

```
#include <tree.h>
```

Métodos públicos

- bool **operator!=** (const postorderiterator &i) const
- T & **operator*** ()
- postorderiterator & **operator++** ()
- bool **operator==** (const postorderiterator &i) const
- **postorderiterator** (node n)

5.10.1. Descripción detallada

Clase iterator para recorrer el árbol en PostOrden

La documentación para esta clase fue generada a partir del siguiente fichero:

- tree.h

5.11. Referencia de la Clase tree::preorderiterator

`#include <tree.h>`

Métodos públicos

- bool **operator!=** (const preorderiterator &i) const
- T & **operator*** ()
- preorderiterator & **operator++** ()
- bool **operator==** (const preorderiterator &i) const
- **preorderiterator** (node n)

5.11.1. Descripción detallada

Clase iterator para recorrer el árbol en PreOrden

La documentación para esta clase fue generada a partir del siguiente fichero:

- tree.h

5.12. Referencia de la Clase SoED

`#include <SoED.h>`

Clases

- struct entrada

Tipos públicos

- typedef unsigned int error
- typedef string path
- typedef unsigned int size_type
- typedef string userID

Métodos públicos

- error cd (const userID &u, const path &d)

  Cambio de directorio.
- void cd (const userID &u)

  Cambio de directorio.
- error cp (const userID &u, const path &org, const path &dest)

  copia una entrada orgien a una destino
- pair< size_type, error > du (const userID &u, const path &p) const

  tamaño ocupado por el sistema
- size_type du (const userID &u) const

  tamaño ocupado por el sistema
- list< path > locate (const userID &u, const string &s) const

  busca una cadena en el árbol de directorios
- list< string > ls (const userID &u) const

  lista las entradas de un directorio
- error mkdir (const userID &u, const path &dir)

  Crea un directorio.
- error mkFile (const userID &u, const string &s, size_type tama)

  Crea un fichero enel directorio de trabajo.
- error mv (const userID &u, const path &org, const path &dest)

  renombra org a dest o mueve org a un directorio
- path pwd (const userID &user) const

  devuelve el camino completo del directorio de trabajo para el usuario user
- void reading_SoED ()

  Lectura de un directorio.
- error rm (const userID &u, const path &p)

  borra un fichero
- error rmdir (const userID &u, const path &dir)

  Borra un directorio.
- SoED ()

  Constructor primitivo Realiza las siguientes acciones.
- SoED (const SoED &x)

  Constructor de copia.
- list< path > tree (const userID &u, const path &p) const

  lista de todos los directorios del sistema

- list< path > tree (const userID &u) const

  lista de todos los directorios del sistema
- bool useradd (const userID &user)

  añade un nuevo usuario al sistema
- void userdel (const userID &user)

  borra un usuario del sistema
- list< userID > who () const

  lista todos los usuarios del sistem

5.12.1. Descripción detallada

Descripción

Toda la información, ya sean textos, imágenes, o información para la configuración del sistema, se almacena en "ficheros", que a su vez se guardan en directorios. Los ficheros son la estructura empleada por el sistema operativo para almacenar información en un dispositivo físico como un disco duro, un disquete, un CD-ROM.

En esta práctica implementaremos un simulador para un sistema de archivos, llamado SoED (Sistema operativo Estructuras de Datos). Nuestro sistema será multiusuario, por lo que para gestionarlo necesitaremos de dos partes esenciales, por un lado un gestor de usuarios y por otro un árbol de directorios.

Tareas pendientes Tareas a realizar: El alumno debera implementar la clase SoED, juntos con el codigo de prueba de los distintos metodos.

Ver la documentación de la práctica (Introducción)

5.12.2. Documentación de los 'Typedef' miembros de la clase

5.12.2.1. SoED::error

Hace referencia al tipo asociado a los errores que se pueden generar. Los códigos de error son

- 0. operación correcta
- 1. camino incorrecto (no existe)
- 2. un usario no tiene permiso para acceder al camino
- 3. directorio vacío

Definición en la línea 47 del archivo SoED.h.

5.12.2.2. SoED::path

Hace referencia al un camino completo en el árbol de directorios, que puede hacer referencia a un directorio o archivo concreto.

- /home/usario/dir1/fichero.dat si es un nombre de fichero.
- /home/usuario/dir1/dir11/ si es un nombre de directorio Como se ha comentado, cuando si consideramos un camino completo tenemos tres alternativas para describirlo. Supongamos que consideramos el usuario "pedro", que se encuentra en directorio de trabajo "/home/pedro/ED/", las tres alternativas harían referencia al directorio DIR, que se encontraría como subdirectorio de ED
- dir ="/home/pedro/ED/DIR/"
- dir ="∼/ED/DIR/"
- dir ="./DIR/"

Definición en la línea 68 del archivo SoED.h.

5.12.2.3. SoED::size_type

Hace referencia al tipo asociado al tamaño de los ficheros

Definición en la línea 52 del archivo SoED.h.

5.12.2.4. SoED::userID

Hace referencia al tipo asociado al nombre del usuario. En principio un string con números y/o caracters

Definición en la línea 57 del archivo SoED.h.

5.12.3. Documentación del constructor y destructor

5.12.3.1. SoED::SoED ( )

Constructor primitivo Realiza las siguientes acciones.

1. crea el usuario "root"
2. crea los directorios raíz ("/") y "/home/", ambos teniendo como propietario al root
3. asigna "/" como directorio por defecto y directorio actual de root

5.12.3.2. SoED::SoED (const SoED & *x*)

Constructor de copia.

Parámetros

in	*x*	SoED que se copia

5.12.4. Documentación de las funciones miembro

5.12.4.1. error SoED::cd (const userID & *u*, const path & *d*)

Cambio de directorio.

Parámetros

in	*u*	identificador del usuario
in	*d*	camino Hace que el usuario cambie al directorio de trabajo representado por d

Devuelve

codigo de error

5.12.4.2. void SoED::cd (const userID & *u*)

Cambio de directorio.

Parámetros

in	*u*	identificador del usuario

Hace que el usuario cambie a su directorio raíz, en /home/...

5.12.4.3. error SoED::cp (const userID & *u*, const path & *org*, const path & *dest*)

copia una entrada orgien a una destino

Parámetros

in	*u*	nombre de usuario
in	*org*	nombre de la entrada a mover (fichero o directorio)
in	*dest*	destino de la entrada (fichero o directorio)

Copia el directorio/fichero origen al directorio dest.

Devuelve

codigo de error

5.12.4.4. pair<size_type,error> SoED::du (const userID & *u,* const path & *p*) const

tamaño ocupado por el sistema

Parámetros

in	*u*	identificador del usuario
in	*p*	camino a un directorio

Devuelve el numero de K ocupadas por todos los ficheros que se encuentran a partir de un directorio p. El directorio p debe ser propiedad del usuario u (excepto para root, que tiene acceso a todo el sistema)

Devuelve

Devuelve un par donde el primer campo representa el tamaño y segundo campo representa si se ha podido realizar la acción (codgio de error)

5.12.4.5. size_type SoED::du (const userID & *u*) const

tamaño ocupado por el sistema

Parámetros

in	*u*	identificador del usuario

Devuelve

Devuelve el numero de K ocupadas por todos los ficheros propiedad del usuario u

5.12.4.6. list<path> SoED::locate (const userID & *u,* const string & *s*) const

busca una cadena en el árbol de directorios

Parámetros

in	*s*	cadena a buscar
in	*u*	nombre de usuario

- Busca la cadena s dentro del subdirectorio del usuario u, si u == root busca la cadena s en todo el directorio.
- La cadena s debe ser subcadena de un nombre de directorio o de un nombre de

archivo.

Devuelve

lista de nombres de paths que contienen a s.

5.12.4.7. list<string> SoED::ls (const userID & *u*) const

lista las entradas de un directorio

Parámetros

in	*u*	identificador del usuario lista todas las entradas de un directorio

Devuelve

la lista con los nombres de entradas de un directorio

5.12.4.8. error SoED::mkdir (const userID & *u,* const path & *dir*)

Crea un directorio.

Parámetros

in	*u*	nombre de usuario
in	*dir*	nombre del directoria a crear, debe acabar en /

En dir podemos distinguir dos partes, el camino y en nombre del nuevo directorio ".../camino/nuevoDir/", Si existe el camino en árbol de directorios y el usuario tiene acceso al mismo, crea un nuevo directorio

Devuelve

codigo de error

5.12.4.9. error SoED::mkFile (const userID & *u,* const string & *s,* size_type *tama*)

Crea un fichero enel directorio de trabajo.

Parámetros

in	*u*	identificador del usuario
in	*s*	nombre del fichero

in	*tama*	tamaño del fichero Crea un nuevo fichero con nombre s en el directorio de trabajo con tamaño tama. En el caso en que exista ya un fichero o directorio con en mismo nombre, el fichero no se puede crear. El nuevo fichero sera propiedad del usuario u.

Devuelve

codigo de error

5.12.4.10. error SoED::mv (const userID & *u,* const path & *org,* const path & *dest*)

renombra org a dest o mueve org a un directorio

Parámetros

in	*u*	nombre de usuario
in	*org*	nombre de la entrada a mover (fichero o directorio)
in	*dest*	destino de la entrada (fichero o directorio)

Mueve o renombre el directorio/fichero origen al directorio dest. Si el dest se encuentra en el mismo directorio que org, entonces renombra el directorio, si dest se encuentra en otro directorio lo que hace es mover la entrada al directorio destino (eliminándose del directorio origen)

Devuelve

codigo de error

5.12.4.11. path SoED::pwd (const userID & *user*) const

devuelve el camino completo del directorio de trabajo para el usuario user

Parámetros

in	*user*	Nombre el usuario

5.12.4.12. void SoED::reading_SoED ( )

Lectura de un directorio.

Se genera el arbol de directorio utilizando un recorrido por nivel. Esta método (que se entrega implementado) sólo se utilizará para poder generarnos un directorio de prueba. Se le asignará el usuario ficticio "kkk"

5.12.4.13. error SoED::rm (const userID & *u,* const path & *p*)

borra un fichero

Parámetros

in	*u*	identificador del usuario
in	*p*	camino del fichero

Borra el fichero indicado por el camino p. Si p=="∗", borra todos los ficheros del directorio de trabajo

Devuelve

codigo de error

5.12.4.14. error SoED::rmdir (const userID & *u,* const path & *dir*)

Borra un directorio.

Parámetros

in	*u*	nombre de usuario
in	*dir*	nombre del directoria a borrar, debe acabar en /

En dir podemos distinguir dos partes, el camino y en nombre del directorio a borrar ".../camino/ DirABorrar/". Si existe el camino en árbol de directorios, y el usuario tiene permisos borra DirABorrar/ siempre que esté vacio

Devuelve

codigo de error

5.12.4.15. list<path> SoED::tree (const userID & *u,* const path & *p*) const

lista de todos los directorios del sistema

Parámetros

in	*u*	identificador del usuario
in	*p*	camino a un directorio

Devuelve

Devuelve la lista de caminos completos hacia todos los archivos/directorios que cuelgan de p. En caso de error, u no tiene acceso a p, o p es un camino erroneo devuelve la lista vacía.

5.12.4.16. list<path> SoED::tree (const userID & *u*) const

lista de todos los directorios del sistema

Parámetros

in	*u*	identificador del usuario

Devuelve

Devuelve la lista de caminos completos de todos los archivos/directorios que cuelgan de p.

5.12.4.17. bool SoED::useradd (const userID & *user*)

añade un nuevo usuario al sistema

Parámetros

in	*user*	identificador del usuario Añade un nuevo usuario, el nombre user no debe existir en el sistema. Crea el directorio raíz del usuario en /home/user, que es propiedad de dicho usuario

Devuelve

Devuelve true si se ha podido realizar la acción, false si ya existe en usuario

5.12.4.18. void SoED::userdel (const userID & *user*)

borra un usuario del sistema

Parámetros

in	*user*	identificador del usuario Borra el usuario, y todo su directorio del sistema.

5.12.4.19. list<userID> SoED::who ( ) const

lista todos los usuarios del sistem

Devuelve

Devuelve una lista con todos los usuarios del sistema

La documentación para esta clase fue generada a partir del siguiente fichero:

- SoED.h

5.13. Referencia de la Clase tree

`#include <tree.h>`

Clases

- class const_inorderiterator
- class const_leveliterator
- class const_postorderiterator
- class const_preorderiterator
- class inorderiterator
- class leveliterator
- class node
- class nodewrapper
- class postorderiterator
- class preorderiterator

Tipos públicos

- typedef unsigned int size_type

Métodos públicos

- void assign_subtree (const tree< T > &a, node n)

  Reemplaza el receptor por una copia de subárbol.
- inorderiterator **beginInorder** ()
- const_inorderiterator **beginInorder** () const
- leveliterator **beginlevel** ()
- const_leveliterator **beginlevel** () const
- postorderiterator **beginPostorder** ()
- const_postorderiterator **beginPostorder** () const
- preorderiterator **beginPreorder** ()
- const_preorderiterator **beginPreorder** () const
- void clear ()

  Hace nulo un árbol.
- bool empty () const

  Comprueba si un árbol esta vacío.
- inorderiterator **endInorder** ()
- const_inorderiterator **endInorder** () const
- leveliterator **endlevel** ()
- const_leveliterator **endlevel** () const
- postorderiterator **endPostorder** ()
- const_postorderiterator **endPostorder** () const
- preorderiterator **endPreorder** ()

- const_preorderiterator **endPreorder** () const
- node insert_left (node n, const T &e)

  Insertar un nodo como hijo a la izquierda de un nodo.
- node insert_left (node n, tree< T > &rama)

  Insertar un árbol como subárbol hijo a la izquierda de un nodo.
- node insert_right_sibling (node n, const T &e)

  Insertar un nodo como hermano a la derecha de un nodo.
- node insert_right_sibling (node n, tree< T > &rama)

  Insertar un árbol como subárbol hermano a la derecha de un nodo.
- bool is_external (node v) const

  Comprueba si un nodo es exterior.
- bool is_internal (node v) const

  Comprueba si un nodo es interior.
- bool is_root (node n) const

  Comprueba si un nodo es la raíz.
- bool null () const

  Comprueba si un árbol es nulo.
- bool operator!= (const tree< T > &a) const

  Operador de comparación de desigualdad.
- tree< T > & operator= (const tree< T > &a)

  Operador de asignación.
- bool operator== (const tree< T > &a) const

  Operador de comparación de igualdad.
- void prune_left (node n, tree< T > &dest)

  Podar el subárbol hijo a la izquierda de un nodo.
- void prune_right_sibling (node n, tree< T > &dest)

  Podar el subárbol hermano a la derecha de un nodo.
- node root () const

  Obtener el nodo raíz.
- node setroot (const T &v)

  Asigna la raiz al arbol vacío.
- size_type size () const

  Obtiene el nmero de nodos.
- tree ()

  Constructor primitivo por defecto.
- tree (const T &e)

  Constructor primitivo.
- tree (const tree< T > &a)

  Constructor de copia.
- ~tree ()

  Destructor.

5.13.1. Descripción detallada

tree::tree, assign_subtree, setroot, root, ∼tree, =, prune_left, prune_right_sibling,insert-_left, insert_right_sibling, clear, size, empty, ==, !=, is_root, internal, external

Representa un árbol general con nodos etiquetados con datos del tipo T.

T debe tener definidas las operaciones:

- T & operator=(const T & e);
- bool operator!=(const T & e);
- bool operator==(const T & e);

Son mutables. Residen en memoria dinámica.

5.13.2. Documentación de los 'Typedef' miembros de la clase

5.13.2.1. tree::size_type

Hace referencia al tipo asociado la tamaño del tree

Definición en la línea 45 del archivo tree.h.

5.13.3. Documentación del constructor y destructor

5.13.3.1. tree::tree ( )

Constructor primitivo por defecto.

Crea un árbol nulo.

5.13.3.2. tree::tree (const T & *e*)

Constructor primitivo.

Parámetros

e,:	Etiqueta para la raíz.

Crea un árbol con un único nodo etiquetado con e.

5.13.3.3. tree::tree (const tree< T > & *a*)

Constructor de copia.

Parámetros

	a árbol que se copia.

Crea un árbol duplicado exacto de a.

5.13.3.4. tree::∼tree ( )

Destructor.

Destruye el receptor liberando los recursos que ocupaba.

5.13.4. Documentación de las funciones miembro

5.13.4.1. void tree::assign_subtree (const tree< T > & *a,* node *n*)

Reemplaza el receptor por una copia de subárbol.

Parámetros

a,:	árbol desde el que se copia.
n,:	nodo raíz del subárbol que se copia.

El receptor se hace nulo y después se le asigna una copia del subárbol de a cuya raíz es n.

5.13.4.2. void tree::clear ( )

Hace nulo un árbol.

Destruye todos los nodos del árbol receptor y lo hace un árbol nulo.

5.13.4.3. bool tree::empty ( ) const

Comprueba si un árbol esta vacío.

Devuelve

true, si el receptor es un árbol vacío. false, en otro caso.

5.13.4.4. node tree::insert_left (node *n,* const T & *e*)

Insertar un nodo como hijo a la izquierda de un nodo.

Parámetros

n,:	nodo del receptor. n != nodo_nulo.
e,:	etiqueta del nuevo nodo.

Inserta un nuevo nodo con etiqueta e como hijo a la izquierda, el anterior hijo más a la izquierda queda como hermano a la derecha del recién insertado

5.13.4.5. node tree::insert_left (node *n,* tree< T > & *rama*)

Insertar un árbol como subárbol hijo a la izquierda de un nodo.

Parámetros

n,:	nodo del receptor. n != nodo_nulo.
rama,:	subárbol que se inserta. Es MODIFICADO.

Si rama no es un árbol vacío: Inserta rama como hijo a la izquierda de n, el anterior hijo más a la izquierda queda como hermana a la derecha del recién insertado. y rama se hace árbol nulo. En caso contrario no se hace nada

5.13.4.6. node tree::insert_right_sibling (node *n,* const T & *e*)

Insertar un nodo como hermano a la derecha de un nodo.

Parámetros

n,:	nodo del receptor. !n.Nulo().
e,:	etiqueta del nuevo nodo.

Inserta un nuevo nodo con etiqueta e como hermano a la derecha, el anterior hermano a la derecha de n queda como hermano a la derecha del nodo insertado

5.13.4.7. node tree::insert_right_sibling (node *n,* tree< T > & *rama*)

Insertar un árbol como subárbol hermano a la derecha de un nodo.

Parámetros

n,:	nodo del receptor. !n.Nulo().
rama,:	subárbol que se inserta. Es MODIFICADO.

Si rama no es un árbol vacío: Asigna el valor de rama como nuevo subárbol hermano a la derecha, el anteriot hermano a la derecha de n queda como hermano a la derecha del nodo insertado y rama se hace árbol nulo. En caso contrario no se hace nada

5.13.4.8. bool tree::is_external (node *v*) const [inline]

Comprueba si un nodo es exterior.

Parámetros

v,:	nodo que se evala.

Devuelve

true, si n es exterior. false, en otro caso.

Definición en la línea 246 del archivo tree.h.

5.13.4.9. bool tree::is_internal (node *v*) const [inline]

Comprueba si un nodo es interior.

Parámetros

v,:	nodo que se evala.

Devuelve

true, si n es interior. false, en otro caso.

Definición en la línea 236 del archivo tree.h.

5.13.4.10. bool tree::is_root (node *n*) const [inline]

Comprueba si un nodo es la raíz.

Parámetros

n,:	nodo que se evala.

Devuelve

true, si n es la raíz del receptor. false, en otro caso.

Definición en la línea 226 del archivo tree.h.

5.13.4.11. bool tree::null ( ) const

Comprueba si un árbol es nulo.

Devuelve

true, si el receptor es un árbol nulo. false, en otro caso.

5.13.4.12. bool tree::operator!= (const tree< T > & *a*) const

Operador de comparación de desigualdad.

Parámetros

a,:	árbol con que se compara el receptor.

Devuelve

true, si el receptor no es igual, en estructura o etiquetas a a. false, en otro caso.

5.13.4.13. tree<T>& tree::operator= (const tree< T > & *a*)

Operador de asignación.

Parámetros

a,:	árbol que se asigna.

Destruye el contenido previo del receptor y le asigna un duplicado de a.

5.13.4.14. bool tree::operator== (const tree< T > & *a*) const

Operador de comparación de igualdad.

Parámetros

a,:	árbol con que se compara el receptor.

Devuelve

true, si el receptor es igual, en estructura y etiquetas a a. false, en otro caso.

5.13.4.15. void tree::prune_left (node *n,* tree< T > & *dest*)

Podar el subárbol hijo a la izquierda de un nodo.

Parámetros

n,:	nodo del receptor. n != nodo_nulo.
dest,:	subárbol hijo a la izquierda de n. Es MODIFICADO.

Desconecta el subárbol hijo a la izquierda de n, que pasa a ser el árbol que era su hermano a la derecha, si lo tuviera. El subárbol anterior se devuelve sobre dest.

5.13.4.16. void tree::prune_right_sibling (node *n,* tree< T > & *dest*)

Podar el subárbol hermano a la derecha de un nodo.

Parámetros

n,:	nodo del receptor. n != nodo_nulo.
dest,:	subárbol hermano a la derecha de n. Es MODIFICADO.

Desconecta el subárbol hermano a la derecha de n, que pasa a ser el árbol que era su hermano a la derecha, si lo tuviera. El subárbol anterior se devuelve sobre dest.

5.13.4.17. node tree::root ( ) const

Obtener el nodo raíz.

Devuelve

nodo raíz del receptor.

5.13.4.18. node tree::setroot (const T & *v*)

Asigna la raiz al arbol vacío.

Parámetros

v,:	el valor a almacenar en la raiz.

Precondición

el receptor es el árbol nulo.

5.13.4.19. size_type tree::size ( ) const

Obtiene el nmero de nodos.

Devuelve

nmero de nodos del receptor.

La documentación para esta clase fue generada a partir del siguiente fichero:

- tree.h

Capítulo 6

Documentación de archivos

6.1. Referencia del Archivo SoED.h

TDA SoED.

```
#include <iostream>  #include <string>  #include <map> ×
#include "tree.h"
```

Clases

- struct SoED::entrada
- class SoED

6.1.1. Descripción detallada

TDA SoED.

Definición en el archivo SoED.h.

6.2. Referencia del Archivo tree.h

TDA tree.

```
#include <queue>  #include <iostream>  #include <stack> ×
#include <list>  #include "tree.hxx"  #include "nodetree.-
hxx"
```

Clases

- class tree::const_inorderiterator
- class tree::const_leveliterator

- class tree::const_postorderiterator
- class tree::const_preorderiterator
- class tree::inorderiterator
- class tree::leveliterator
- class tree::node
- class tree::nodewrapper
- class tree::postorderiterator
- class tree::preorderiterator
- class tree

6.2.1. Descripción detallada

TDA tree.

Definición en el archivo tree.h.