Trabajo práctico

LinkLinkIt

Fecha de compilación: 3 de Octubre de 2012

Normativa

Fecha de entrega: 24 de Octubre de 2012.
Normas de entrega: Las contenidas en la página web de la materia.

Enunciado

El objetivo de este trabajo práctico consiste en realizar el diseño completo del módulo que se explica con el TAD LinkLinkIt (ver la sección Especificación de este documento), así como todos aquellos módulos necesarios para la tarea.

Con respecto al enunciado del tp1 hay algunos cambios.

- Si bien se mantiene el requisito que solo se almacenen los accesos recientes: los accesos de los últimos tres días, se desea que sean los últimos días desde el acceso más reciente *por link*, y no en todo el sistema como era en el tp1.
- Por otro lado, se desea contar con una funcionalidad que, dada una categoría, se permita recorrer los links dentro de ella y de sus subcategorias. Dicho listado debe estar ordenado por cantidad de accesos recientes. Primero los links que tuvieron más visitas. Ahora bien, la cantidad de accesos debe normalizarse. Como cada link tiene registrados los accesos de potencialmente distintos dias, deben hacerle los calculos pertinentes.

Estos cambios están especificados como son deseados. Confíen en la axiomatización más que en el español.

Complejidades Requeridas

Dentro de los módulos a diseñar deberá haber uno que se corresponda con el TAD ArbolCategorias y otro con el TAD LinkLinkIt.

Para el módulo del árbol de categorias se deberán exponer, al menos, las siguientes funcionalidad y complejidades:

- crear un árbol con categoría *raiz* debe ser $O(|raiz|)$.
- dado un árbol, obtener el nombre de la categoría raíz debe ser $O(1)$.
- dado un árbol, el nombre de una categoría *padre* existente y el nombre de una categoría *hija* no existente en el árbol agregar *hija* a *padre* debe ser $O(|padre| + |hija|)$.
- dado un árbol *ac* y el nombre de una categoría *padre* retornar un iterador unidireccional de los hijos directos de *padre* en *ac* en $O(|padre|)$. El iterador debe proyectar, al menos, el nombre de las categorías hijas. Las operaciones de iteración deben ser $O(1)$.
- dado un árbol *ac* y el nombre de una categoría *c* existente retornar el *id* de *c* debe ser $O(|c|)$.

Nota: Se sugiere estudiar un diseño que incluya referencias o punteros inteligentes a las categorías de un árbol. Esto puede permitirles "navegar" el árbol con costos ínfimos y quizás agregar algunas de las operaciones enunciadas como interfaz de dichos punteros.

Para el módulo del sistema se deberán exponer, al menos, las siguientes funcionalidades y complejidades:

- crear un sistema dado un árbol *ac* de categorias deber ser $O(\#categorias(ac))$.
- dado un sistema *s* agregar un link *l* en una categoría de nombre *c* debe ser $O(|l| + |c| + h)$.
- dado un sistema *s* acceder a un link *l* debe ser $O(|l|)$, o en caso de necesitarlo $O(|l| + h)$.
- dado un sistema *s* y el nombre de una categoria *c* retornar $cantLinks(s, c)$ debe ser $O(|c|)$.
- dado un sistema *s* y el nombre de una categoria *c* se desea retornar un iterador unidireccional que corresponda con $linksOrdenadosPorAccesos(s, c)$. El iterador debe proyectar, al menos, el link, el nombre de la categoria a la que pertenece exactamente y la cantidad de accesos recientes según $accesosRecientes(s, c, l)$ en $O(1)$. Las operaciones de iteración deben ser $O(1)$. La operación que devuelve el iterador, en peor caso puede ser $O(n^2)$. Pero debe ser menor cuando resulte posible. Por ejemplo dadas dos llamadas consecutivas con los mismos argumentos, la segunda debe ser $O(n)$ (o incluso $O(1)$ si lo desean).

Donde h representa la altura del árbol de categorias del sistema: $h = altura(categorias(s))$. n representa $long(linksYAccesosOrdenados(s, c))$.

La función $|\bullet| : string \to nat$ equivale a *long* de secuencia.

Todas las operaciones deben ser espeficicadas formalmente según las herramientas vistas en clase. Agregar comentarios necesarios para entender la forma en la cual deben ser utilizadas para su correcto funcionamiento. Además todos los algoritmos deben tener su desarrollo que justifique los ordenes de complejidad, si algún paso es no trivial pueden hacer notas a continuación del mismo.

Cuando se formalicen los Rep y Abs, identifiquen cada parte de la fórmula y comentenlas para facilitar su seguimiento y corrección.

Se espera que los módulos tengan bajo nivel de acoplamiento. Una forma de ver esto es que, por ejemplo, al cambiar el módulo del árbol de categorias no se deba alterar la interfaz del sistema por cambios de aridad (exceptuando la creación del sistema).

Cuentan con los TADs y módulos:

- CHAR que representan los posibles caracteres. Siendo un tipo enumerado de 256 valores. con funciones *ord* y ord^{-1} para la correspondencia de cada *Char* a *Nat*.
- string como sinónimo de $Vector < Char >$.
- definidos en el apunte de TADs básicos.
- definidos en el apunte de módulos básicos.

Especificación

TAD LINK es string

TAD FECHA es nat

TAD CATEGORIA es string

TAD ARBOLCATEGORIAS

géneros acat

exporta generadores, categorias, raíz, padre, id, altura, está?, esSubCategoria, alturaCategoria, hijos

usa bool, nat, conjunto

generadores

nuevo : categoria c $\longrightarrow$ acat {¬ vacía?(c)}

agregar : acat ac × categoria c × categoria h $\longrightarrow$ acat {está?(c, ac) ∧ ¬ vacía?(h) ∧ ¬ está?(h, ac)}

observadores básicos

categorias : acat ac $\longrightarrow$ conj(categoria)

raíz : acat ac $\longrightarrow$ categoria

padre : acat ac × categoria h $\longrightarrow$ categoria {está?(h, ac) ∧ raíz(ac) ≠ h}

id : acat ac × categoria c $\longrightarrow$ nat {está?(c, ac)}

otras operaciones

altura : acat ac $\longrightarrow$ nat

está? : categoria c × acat ac $\longrightarrow$ bool

esSubCategoria : acat ac × categoria c × categoria h $\longrightarrow$ bool {está?(c,ac) ∧ está?(h,ac)}

alturaCategoria : acat ac × categoria c $\longrightarrow$ nat {está?(c, ac)}

hijos : acat ac × categoria c $\longrightarrow$ conj(categoria) {está?(c, ac)}

axiomas $\forall\, c, h, h_1, h_2$: categoria, $\forall\, ac$: acat

categorias(nuevo(c)) ≡ {c}

categorias(agregar(ac,c,h)) ≡ Ag(h, categorias(ac))

raíz(nuevo(c)) ≡ c

raíz(agregar(ac,c,h)) ≡ raíz(ac)

padre(agregar(ac,c,h_1),h_2) $\equiv$ **if** $h_1 = h_2$ **then** c **else** padre(ac,c,h_2) **fi**

id(nuevo(c_1), c_2) $\equiv$ 1
id(agregar(ac,c,h_1), h_2) $\equiv$ **if** $h_1 = h_2$ **then** #categorias(ac) + 1 **else** id(ac,h_2) **fi**

altura(nuevo(c)) $\equiv$ alturaCategoria(nuevo(c), c)
altura(agregar(ac,c,h)) $\equiv$ max(altura(ac), alturaCategoria(agregar(ac,c,h), h))
alturaCategoria(ac, c) $\equiv$ **if** c = raíz(ac) **then** 1 **else** 1 + alturaCategoria(ac, padre(ac, c)) **fi**

está?(c,ac) $\equiv$ c $\in$ categorias(ac)

esSubCategoria(ac,c,h) $\equiv$ c = h $\vee_L$ (h $\neq$ raíz(ac) $\wedge_L$ esSubCategoria(ac, c, padre(ac, h)))

hijos(nuevo(c_1), c_2) $\equiv$ $\emptyset$
hijos(agregar(ac,c_1,h), c_2) $\equiv$ **if** h = c_2 **then**
$\emptyset$
else
(**if** c_1=c_2 **then** {h} **else** $\emptyset$ **fi**)$\cup$hijos(ac,c_1,c_2)
fi

Fin TAD

A continuación se usará constante *diasRecientes* = 3 para encapsular en ella la dependencia del tamaño de la ventana de tiempo que determina que una fecha sea reciente o no.

TAD LinkLinkIt

géneros lli

exporta generadores, categorias, links, categoriaLink, fechaActual, fechaUltimoAcceso, accesosRecientesDia, esReciente?, accesosRecientes, linksOrdenadosPorAccesos, cantLinks

usa bool, nat, conjunto, secuencia, arbolCategorias

generadores

iniciar : acat *ac*	$\longrightarrow$ lli	
nuevoLink : lli s $\times$ link l $\times$ categoria c	$\longrightarrow$ lli	{$\neg$ l $\in$ links(s) $\wedge$ está?(c, categorias(s))}
acceso : lli s $\times$ link l $\times$ fecha f	$\longrightarrow$ lli	{l $\in$ links(s) $\wedge$ f $\geq$ fechaActual(s)}

observadores básicos

categorias : lli s	$\longrightarrow$ acat	
links : lli s	$\longrightarrow$ conj(link)	
categoriaLink : lli s $\times$ link l	$\longrightarrow$ categoria	
fechaActual : lli s	$\longrightarrow$ fecha	
fechaUltimoAcceso : lli s $\times$ link l	$\longrightarrow$ fecha	{l $\in$ links(s)}
accesosRecientesDia : lli s $\times$ link l $\times$ fecha f	$\longrightarrow$ nat	{l $\in$ links(s) $\wedge$ esReciente?(s,l,f)}

otras operaciones

esReciente? : lli s $\times$ link l $\times$ fecha f	$\longrightarrow$ bool	{l $\in$ links(s)}
accesosRecientes : lli s $\times$ categoria c $\times$ link l	$\longrightarrow$ nat	{está?(c, categorias(s)) $\wedge$ l $\in$ links(s) $\wedge$ esSubCategoria(categorias(s), c, categoriaLink(s, l))}
linksOrdenadosPorAccesos : lli s $\times$ categoria c	$\longrightarrow$ secu(link)	{está?(c, categorias(s))}
cantLinks : lli s $\times$ categoria c	$\longrightarrow$ nat	{está?(c, categorias(s))}

(desde acá no se exportan)

menorReciente : lli s $\times$ link l	$\longrightarrow$ fecha	{l $\in$ links(s)}
diasRecientes : lli s $\times$ link l	$\longrightarrow$ conj(fecha)	{l $\in$ links(s)}
diasRecientesDesde : lli s $\times$ link l $\times$ fecha f	$\longrightarrow$ conj(fecha)	{l $\in$ links(s)}
linksCategoriaOHijos : lli s $\times$ categoria c	$\longrightarrow$ conj(link)	{está?(c, categorias(s))}

filtrarLinksCategoriaOHijos : lli s × categoria c × conj(link) ls $\longrightarrow$ conj(link)
{está?(c, categorias(s)) ∧ ls ⊆ links(s)}

diasRecientesParaCategoria : lli s × categoria c $\longrightarrow$ conj(fecha) {está?(c, categorias(s)}

linkConUltimoAcceso : lli s × categorias c × conj(link) ls $\longrightarrow$ link
{está?(c, categorias(s) ∧ ¬ ∅?(ls) ∧ ls ⊆ linksCategoriaOHijos(s, c)}

sumarAccesosRecientes : lli s × link l × conj(fecha) fs $\longrightarrow$ nat
{l ∈ links(s) ∧ fs ⊆ diasRecientes(s, l)}

linksOrdenadosPorAccesosAux : lli s × categoria c × conj(link) ls $\longrightarrow$ secu(link)
{está?(c, categorias(s)) ∧ ls ⊆ linksCategoriaOHijos(s, c)}

linkConMásAccesos : lli s × categoria c × conj(link) ls $\longrightarrow$ link
{está?(c, categorias(s)) ∧ #ls > 1 ∧ ls ⊆ linksCategoriaOHijos(s, c)}

β : bool b $\longrightarrow$ nat

axiomas ∀ b: bool, ∀ ac: acat, ∀ s: lli, ∀ l, l_1, l_2: link, ∀ c: categoria, ∀ f, f_1, f_2: fecha, ∀ ls: conj(link), ∀ fs: con(fecha)

categorias(iniciar(ac)) ≡ ac
categorias(nuevoLink(s, l, c)) ≡ categorias(s)
categorias(acceso(s, l, f)) ≡ categorias(s)

links(iniciar(ac)) ≡ ∅
links(nuevoLink(s, l, c)) ≡ Ag(l, links(s))
links(acceso(s, l, f)) ≡ links(s)

categoriaLink(nuevoLink(s, l_1, c), l_2) ≡ **if** $l_1 = l_2$ **then** c **else** categoriaLink(s, l_2) **fi**
categoriaLink(acceso(s, l_1, f), l_2) ≡ categoriaLink(s, l_2)

fechaActual(iniciar(ac)) ≡ 0
fechaActual(nuevoLink(s, l, c)) ≡ fechaActual(s)
fechaActual(acceso(s, l, f)) ≡ f

fechaUltimoAcceso(nuevoLink(s, l_1, c), l_2) ≡ **if** $l_1 = l_2$ **then**
fechaActual(s)
else
fechaUltimoAcceso(s, l_2)
fi
fechaUltimoAcceso(acceso(s, l_1, f), l_2) ≡ **if** $l_1 = l_2$ **then** f **else** fechaUltimoAcceso(s, l_2) **fi**

menorReciente(s,l) ≡ max(fechaUltimoAcceso(s, l) + 1,*diasRecientes*) - *diasRecientes*
esReciente?(s,l,f) ≡ menorReciente(s,l) ≤ f ∧ f ≤ fechaUltimoAcceso(s,l)

accesosRecientesDia(nuevoLink(s,l_1,c),l_2,f) ≡ **if** $l_1 = l_2$ **then** 0 **else** accesosRecientesDia(s,l_2,f) **fi**
accesosRecientesDia(acceso(s,l_1,f_1),l_2,f_2) ≡ $\beta(l_1 = l_2 \wedge f_1 = f_2)$ + (**if** esReciente?(s,l,f_2) **then**
accesosRecientesDia(s, l_2, f_2)
else
0
fi)

accesosRecientes(s, c, l) ≡ sumarAccesosRecientes(s, l, diasRecientesParaCategoria(s, c) ∩ diasRecientes(s, l))

linksOrdenadosPorAccesos(s, c) ≡ linksOrdernadosPorAccesosAux(s, c, linksCategoriaOHijos(s, c))

linksOrdernadosPorAccesosAux(s, c, ls) ≡ **if** ∅?(ls) **then**
∅
else
linkConMásAccesos(s, c, ls) •
linksOrdernadosPorAccesosAux(s, c, ls - {linkConMásAccesos(s, c, ls)})
fi

linkConMásAccesos(s, c, ls) ≡ **if** #ls=1 **then**
dameUno(ls)
else
if accesosRecientes(s,c,dameUno(ls)) > accesosRecientes(s,c,linkConMásAccesos(s,c,sinUno(ls))) **then** dameUno(ls)
else linkConMásAccesos(s,c,sinUno(ls)) **fi**
fi

cantLinks(s, c) ≡ # linksCategoriaOHijos(s, c)

diasRecientes(s, l) ≡ diasRecientesDesde(s, l, menorReciente(s, l))

diasRecientesDesde(s, l, f) ≡ **if** esReciente?(s, l, f) **then**
Ag(f, diasRecientesDesde(s, l, f+1))
else
∅
fi

linksCategoriaOHijos(s, c) ≡ filtrarLinksCategoriaOHijos(s, c, links(s))

filtrarLinksCategoriaOHijos(s, c, ls) ≡ **if** ∅?(ls) **then**
∅
else
((**if** esSubCategoria(categorias(s), c, categoriaLink(s, dameUno(ls))) **then** {dameUno(ls)} **else** ∅ **fi**) ∪ filtrarLinksCategoriaOHijos(s, c, sinUno(ls)))
fi

diasRecientesParaCategoria(s, c) ≡ **if** ∅?(linksCategoriaOHijos(s, c)) **then**
∅
else
diasRecientes(s, linkConUltimoAcceso(s, c, linksCategoriaOHijos(s, c)))
fi

sumarAccesosRecientes(s, l, fs) ≡ **if** ∅?(fs) **then**
0
else
accesosRecientesDia(s, l, dameUno(f)) +
sumarAccesosRecientes(s, l, sinUno(fs))
fi

β(b) ≡ **if** b **then** 1 **else** 0 **fi**

Fin TAD