Módulo SADAIC

Interfaz

se explica con: SADAIC
géneros: `sadaic`
usa: Conjunto Lineal(α), Diccionario Rapido(α)

Operaciones básicas de SADAIC

INICIAR(**in** *as*: conj(autor)) $\rightarrow$ *res* : *sadaic*
Pre $\equiv$ {$\neg\emptyset(as)$}
Post $\equiv$ {$autores(res) = as \wedge temas(res) = \emptyset \wedge_L (\forall a : autor)(a \in autores(a)) \Rightarrow [temasDe(res, a) = \emptyset \wedge temasDisputadosPor(res, a) = \emptyset]$}
Complejidad: O(#(as) x maximo(as)) (maximo(as) = el tamao del nombre del autor con nombre mas largo)
Descripción: genera un SADAIC con los autores del parámetro sin temas.

NUEVOTEMA(**in/out** *s*: sadaic, **in** *t*: tema, **in** *as*: conj(autor))
Pre $\equiv$ { t $\notin$ temas(s) $\wedge$ $\neg$ $\emptyset$(as) $\wedge$ as $\subseteq$ autores(s) $\wedge$ s $=_{obs}$ s_0}
Post $\equiv$ {s $=_{obs}$ nuevoTema(s_0,t,as) $\wedge$ temas(s) $=_{obs}$ Ag(t,temas(s_0)) $\wedge$
($\forall$ a:autor) a $\in$ as $\Rightarrow$ (temasDe(s,a) = Ag(t,temasDe(s_0,a)))}
Complejidad: O(A x (long(l)+ T + #(as)) + long(t))
Descripción: agrega el tema **t** al sadaic **s**.

DISPUTARTEMA(**in/out** *s*: sadaic, **in** *a*: autor, **in** *t*: tema)
Pre $\equiv$ {$t \in temas(s) \wedge a \in autores(s) \wedge t \notin temasDe(s, a) \cup temasDisputadosPor(s, a) \wedge s = s_0$}
Post $\equiv$ {s $=_{obs}$ disputarTema(s_0,a,t) $\wedge$ temasDisputadosPor(s,a) = Ag(t, temasDisputadosPor(s_0,a))}
Complejidad:O((long(a) + A x (long(Autor) + T)))
Descripción: agrega el tema **t** a los temas disputados de **a** en el sadaic **s**.

TEMAS(**in** *s*: sadaic) $\rightarrow$ *res* : $itConj(tema)$
Pre $\equiv$ {true}
Post $\equiv$ {alias(esPermutacion?(secuSuby(res), temas(s)))}
Complejidad: O(1)
Descripción: devuelve un iterador a los temas del SADAIC.
Aliasing: El res no es modificable, no apunta a una copia del conjunto. El iterador se invalida sí y solo sí se elimina el elemento siguiente del iterador sin utilizar EliminarSiguiente.

AUTORES(**in** *s*: sadaic) $\rightarrow$ *res* : $itConj(autor)$
Pre $\equiv$ {true}
Post $\equiv$ {alias(esPermutacion?(secuSuby(res),autores(s)}
Complejidad: O(1)
Descripción: devuelve un iterador a un conjunto con los autores del SADAIC.
Aliasing: El res no es modificable, no apunta a una copia del conjunto. El iterador se invalida sí y solo sí se elimina el elemento siguiente del iterador sin utilizar EliminarSiguiente.

TEMASDE(**in** *s*: sadaic, **in** *a*: autor) $\rightarrow$ *res* : $itConj(itConj(tema))$
Pre $\equiv$ {a $\in$ autores(s)}
Post $\equiv$ {($\forall$ it: itConj(tema)) esta?(it, secuSuby(res)) $\Leftrightarrow$ ($\exists$ t:tema) t $\in$ temasDe(s,a) $\wedge_L$ alias(siguiente(it),t))}
Complejidad: O(long(a))
Descripción: devuelve los temas del autor **a** en el SADAIC **s**.
Aliasing: El res no es modificable, no apunta a una copia del conjunto. El iterador se invalida sí y solo sí se elimina el elemento siguiente del iterador sin utilizar EliminarSiguiente.

TemasDisputadosPor(**in** s: sadaic, **in** a: autor) $\rightarrow$ res : $itConj(itConj(tema))$
Pre $\equiv$ {a $\in$ autores(s)}
Post $\equiv$ {($\forall$ it:itConj(tema)) esta?(it, secuSuby(res)) $\Leftrightarrow$ ($\exists$ t:tema) t $\in$ temasDisputadosPor(s,a) $\wedge_L$ alias(siguiente(it),t))}
Complejidad: O(long(a))
Descripción: devuelve los temas disputados por el autor **a** en el SADAIC **s**.
Aliasing: El res no es modificable, no apunta a una copia del conjunto. El iterador se invalida sí y solo sí se elimina el elemento siguiente del iterador sin utilizar EliminarSiguiente.

DameUnTemaDisputadoPor(**in** s: sadaic, **in** a: autor) $\rightarrow$ res : $tema$
Pre $\equiv$ {a $\in$ autores(s)}
Post $\equiv$ {res = dameUno(temasDisputadosPor(s,a))}
Complejidad: O(long(a)) $\wedge$ O(1) si la llamada anterior a esta funcion fue con el mismo parametro **a**
Descripción: devuelve un tema de los temas disputados por el autor **a**. Llamar a la funcion con el mismo parametro **a**, te va devolviendo todos los temas no disputados por el mismo.
Aliasing: El res no es modificable, se pasa por referencia.

TemasNoDisputados(**in** s: sadaic, **in** a: autor) $\rightarrow$ res : $itConj(itConj(tema))$
Pre $\equiv$ {a $\in$ autores(s)}
Post $\equiv$ {($\forall$ it:itConj(tema)) esta?(it, secuSuby(res)) $\Leftrightarrow$ ($\exists$ t:tema) t $\in$ temasNoDisputados(s,a) $\wedge_L$ alias(siguiente(it),t)}
Complejidad: O(long(a) + A $\times$ (l + (T-D) $\times$ log(T-D))) (***l*** es la longitud del autor con el nombre mas largo)
Descripción: devuelve los temas que no le disputaron al autor **a** y los temas que no le disputaron a los coautores de **a** en el SADAIC **s**.
Aliasing: El res no apunta a una copia del conjunto. El iterador se invalida sí y solo sí se elimina el elemento siguiente del iterador sin utilizar EliminarSiguiente.

FTS(**in** s: sadaic, **in** st: string) $\rightarrow$ res : $itConj(itConj(tema))$
Pre $\equiv$ {true}
Post $\equiv$ {($\forall$ it:itConj(tema)) esta?(it, secuSuby(res)) $\Leftrightarrow$ ($\exists$ t:tema) t $\in$ FTS(s,st) $\wedge_L$ alias(siguiente(it),t))}
Complejidad:O(long(s) + T $\times$ log(T) + A $\times$ log(A))
Descripción: devuelve los temas tales que el string **st** es prefijo del nombre del tema o prefijo de alguno de sus autores
Aliasing: El res no apunta a una copia del conjunto. El iterador se invalida sí y solo sí se elimina el elemento siguiente del iterador sin utilizar EliminarSiguiente.

Coautores(**in** s: sadaic, **in** a: autor) $\rightarrow$ res : $itConj(autor)$
Pre $\equiv$ {a $\in$ autores(s)}
Post $\equiv$ {alias(esPermutacion?(secuSuby(res),coautores(s,a)))}
Complejidad: O(1)
Descripción: devuelve todos los coautores de **a** en el sadaic **s**.
Aliasing: El res no apunta a una copia del conjunto. El iterador se invalida sí y solo sí se elimina el elemento siguiente del iterador sin utilizar EliminarSiguiente.

id(**in/out** s: sadaic, **in** a: autor) $\rightarrow$ res : nat
Pre $\equiv$ {a $\in$ autores(s)}
Post $\equiv$ {res = ord(a)}
Complejidad: O()
Descripción: devuelve el natural que representa al autor **a**.

Representacion

TEMA es STRING
AUTOR es STRING

SADAIC se representa con **estr**

donde **estr** =

<temas: conj(tema),
autores: conj(autor),
FTS: diccS(tema, conj(itConj(tema))),
porAutor:diccS(autor, infoAutor),
cacheoAutor>

donde **infoAutor** es

<susTemas: conj(itConj(tema)), temasQDisputo: conj(itConj(tema)), susCoautores: conj(autor),
temasNoDisputados: conj(itConj(tema))>

donde **cacheoAutor** es

<ultimoAutor: autor, itInicio: itConj(itConj(tema)), temasCacheo: itConj(itConj(tema))>

Algoritmos

iINICIAR(**in** *as*: conj(autor)) → *res* : *estr*

res.autores ← copiar(as)	O(#(as) x S)
res.temas ← Vacío()	O(1)
res.FTS ← CrearDicc()	O(1)
res.porAutor ← CrearDicc()	O(1)
(res.cacheoAutor).ultimoAutor ← "vacío"	O(1)
(res.cacheoAutor).itInicio ← CrearIt(Vacio())	O(1)
(res.cacheAutor).temasCacheo ← CrearIt(Vacio())	O(1)

Complejidad: O(#(as) x S)

donde **S** = el tamao del nombre del autor de as que tiene el nombre mas largo

iNUEVOTEMA(**in/out** *e*: estr, **in** *t*: str, **in** *as*: conj(autor))

```
itAutores: itConj(autor)
itNuevoTema: itConj(tema)
ConjAutor: Conj(autor)
conj1: conj(itConj(tema))
conj2: conj(itConj(tema))
conj3: conj(itConj(autor))
conj4: conj(itConj(tema))
p: tupla(milagrosa)
temas: conj(itConj(tema))

itNuevoTema ← Agregar(e.temas, t)                                   O(#(e.temas))
itAutores ← CrearIt(as)                                             O(1)
while haySiguiente?(itAutores)   O(#(e.autores) × Complejidad del IF cuando la guarda da TRUE)
    if Definido?(e.PorAutor, siguiente(itAutores)) then             O(l)
        p ← Significado(e.PorAutor,siguiente(it))                   O(l)
        AgregarRapido(p.susTemas, itNuevoTema)                      O(1)
        AgregarRapido(p.temasNoDisputados, itNuevoTema)             O(1)
```

ConjAutor ← copiar(as)	O(#(as) × l)
Eliminar(ConjAutor, siguiente(itAutores))	O(#(ConjAutor) × l)
unionPosta(p.susCoautores,ConjAutor)	O(#(p.susCoautores) × log(#(p.susCoautores)) + #(ConjAutor) × log(#(ConjAutor)))
else	
conj1 ← Vacio()	O(1)
conj2 ← Vacio()	O(1)
conj3 ← Vacio()	O(1)
conj4 ← Vacio()	O(1)
AgregarRapido(conj1, itNuevoTema)	O(1)
AgregarRapido(conj4, itNuevoTema)	O(1)
ConjAutor ← copiar(as)	O(#(as))
Eliminar(ConjAutor, siguiente(itAutores))	O(#(ConjAutor) × l)
conj3 ← ConjAutor	O(1)
p ← <conj1,conj2,conj3,conj4>	O(1)
Definir(e.PorAutor, siguiente(itAutores), p)	O(l)
fi	
avanzar(itAutores)	O(1)
endwhile	
temas ← Vacio()	O(1)
if pertenece(t, (e.autores)) **then**	O(#(e.autores))
temas ← copiar(significado(e.PorAutor,t).susTemas)	O(#(significado(e.PorAutor,t).susTemas))
Agregar(temas,itNuevoTema)	O(#(temas))
Definir(e.FTS, t, temas)	O(long(t))
else	
AgregarRapido(temas,itNuevoTema)	O(1)
Definir(e.FTS, t, temas)	O(long(t))
fi	

Complejidad:

O(#(e.temas) + #(e.autores) × (l + l + #(as) × l + #(ConjAutor) × l + #(p.susCoautores) × log(#(p.susCoautores)) + #(ConjAutor) × log(#(ConjAutor))) + #(e.autores) + #(significado(e.PorAutor, t).susTemas) + #(temas) + long(t))

O(T + A × (l + #(as) × l + #(as) × l + A × log(A) + #(as) × log(#(as))) + A + T + T + long(t))

O(T + A × (#(as) × (l + log(#(as)) + A × log(A)) + A + long(t))

O(T + A × (#(as) × (l + log(#(as)) + A × log(A)) + long(t))

donde l = la longitud del autor del conjunto as con nombre mas largo

iDisputarTema(**in/out** e: estr, **in** a: autor, **in** t: tema)

p1: conj(itConj(tema))	
p2: conj(itConj(tema))	
it: itConj(temas)	
itA: itConj(autor)	
it ← CrearIt(e.temas)	O(1)
while(siguiente(it) != t)	O(#(e.temas))
avanzar(it)	O(1)
endwhile	
p1 ← Significado(e.PorAutor, a).temasQDisputo	O(long(a))
AgregarRapido(p1, it)	O(1)
itA → CrearIt(e.autores)	O(1)
while(haySiguiente?(itA))	O(#(e.autores)) × (cuerpo))
p2 ← Significado(e.porAutor,Siguiente(itA)).temasNoDisputados	O(long(Siguiente(itA)))

Eliminar(p2, it) O(#(p2))
endwhile

Complejidad:

O(#(e.temas) + long(a) + #(e.autores) × (long(Siguiente(itA)) + #(p2))

O(T + long(a) + A x (*l* + (T-D)))

O(long(a) + A x (*l* + T-D))

donde ***l*** = el tamao del autor registrado con el nombre mas largo

iTemas(**in** *e*: estr) → *res* : *itConj*(*tema*)

res ← CrearIt(e.temas) O(1)

Complejidad: O(1)

iAutores(**in** *e*: estr) → *res* : *itConj*(*autor*)

res ← CrearIt(e.autores) O(1)

Complejidad: O(1)

iTemasDe(**in** *e*: estr, **in** *a*: autor) → *res* : *conj*(*itConj*(*tema*))

if Definido?(e.PorAutor, a) **then** O(long(a))
res ← Significado(e.porAutor,a).susTemas O(long(a))
else
res ← Vacio() O(1)
fi

Complejidad: O(long(a))

iTemasDisputadosPor(**in** *e*: estr, **in** *a*: autor) → *res* : *conj*(*itConj*(*tema*))

res ← Significado(e.porAutor,a).temasQDisputo O(long(a))

Complejidad: O(long(a))

iDameUnTemaDisputadoPor(**in** *e*: estr, **in** *a*: autor) → *res* : *tema*

it: itConj(itConj(tema))
if (e.cacheoAutor).ultimoAutor = a **then** O(1)
res ← Siguiente(Siguiente((e.cacheoAutor).temasCacheo)) O(1)
it ← (e.cacheoAutor).temasCacheo O(1)
if haySiguiente?(it) **then** O(1)
avanzar(it) O(1)
else
it ← copiar((e.cacheoAutor).itInicio) O(1)
fi
else
it ← CrearIt(Significado(e.porAutor,a).temasQdisputo) O(long(a))
res ← Siguiente(Siguiente(it)) O(1)
(e.cacheoAutor).itInicio ← copiar(it) O(1)
if haySiguiente?(it) **then** O(1)
avanzar(it) O(1)

else
it ← copiar((e.cacheoAutor).itInicio) O(1)
fi
(e.cacheoAutor).autor ← a O(1)
(e.cacheoAutor).temasCacheo ← it O(1)
fi

Complejidad: En el caso que se la invoca con el mismo parámetro que en la llamada anterior, la función es O(1), en caso contrario es O(long(a)).

iTemasNoDisputados(**in** *e*: estr, **in** *a*: autor) → *res* : *conj*(*itConj*(*tema*))

coAut: itConj(autor)
p1: conj(itConj(tema))
p2: conj(itConj(autor))
p3: conj(itConj(tema))

p1 ← Significado(e.porAutor,a).temasNoDisputados O(long(a))
res ← Copiar(p1) O(#(p1))
p2 ← Significado(e.porAutor,a).susCoautores O(long(a))
coAut ← CrearIt(p2) O(1)
while(haySiguiente?(coAut)) O(#(p2) × (cuerpo))
p3 ← Significado(e.porAutor, Siguiente(coAut)).temasNoDisputados O(*l*)
UnionPosta(res, p3) O(#(res) × log(#(res)) + #(p3) × log(#(p3)))
avanzar(coAut) O(1)
endwhile

Complejidad:

O(long(a) + #(p1) + long(a) + #(p2) × (*l* + #(res) × log(#(res)
+ #(p3) × log(#(p3)))

O(long(a) + (T-D) + long(a) + A x (*l* + (T-D) × log(T-D) + (T-D) × log(T-D))

O(long(a) + A × (*l* + (T-D) × log(T-D)))

donde ***l*** es la longitud del nombre del autor que tiene el nombre mas largo.

iFTS(**in** *e*: estr,**in** *s*: string) → *res* : *conj*(*itConj*(*tema*))

if s[Longitud(s)-1] != ∗ **then** O(1)
if Definido?(e.FTS,s) **then** O(long(s))
res ← Significado(e.FTS, s) O(long(s))
else
if Definido?(e.PorAutor,s) **then** O(long(s))
res → significado(e.PorAutor,s) O(long(s))
else
res ← Vacío() O(1)
fi
fi
else
p1 ← significadosHijos(e.FTS,s) O(long(s) + #claves(e.FTS) × log(#claves(e.FTS)))
p2 ← significadosHijos(e.porAutor,s) O(long(s) + #claves(e.porAutor) × log(#claves(e.porAutor)))
unionesConj(p1) O((#(p1) × #(maximo(p1))) × log(#(p1) × #(maximo(p1))))
unionesConj(p2) O((#(p2) × #(maximo(p2))) × log(#(p2) × #(maximo(p2))))
res ← unionPosta(p1,p2) O(#(p1) × log(#(p1)) + #(p2) × log(#(p2)))
fi

Complejidad:

O(long(s) + #claves(e.FTS) × log(#claves(e.FTS)) + long(s) + #claves(e.porAutor) × log(#claves(e.porAutor)) + (#(p1) × #(maximo(p1))) × log(#(p1) × #(maximo(p1))) + (#(p2) × #(maximo(p2))) × log(#(p2) × #(maximo(p2))) + #(p1) × log(#(p1)) + #(p2) × log(#(p2))

O(long(s) + T × log(T) + long(s) + A × log(A) + T × log(T) + T × log(T) + T × log(T) + T × log(T))

O(long(s) + T × log(T) + A × log(A))

iCoautores(**in** e: estr, **in** a: autor) → $res : itConj(autor)$

res ← CrearIt(Significado(e.PorAutor,a).susCoautores) O(1)

Complejidad: O(1)

Funciones Auxiliares

Interfaz

UnionNaif(**in/out** $c1$: conj(α), **in** $c2$: conj(α))
Pre ≡ {$true$}
Post ≡ {(∀ a: α) (a ∈ c1 ∨ a∈ c2 ⇔ a ∈ res)}
Complejidad: O(tam(b) × tam(a))
Descripción: guarda en c1 todos los elementos del c1 y c2

UnionPosta(**in/out** $c1$: conj(α), **in** $c2$: conj(α))
Pre ≡ {α tiene que tener relacion de orden}
Post ≡ {(∀ a: α) (a ∈ c1 ∨ a∈ c2 ⇔ a ∈ res)}
Complejidad: O((tam(a) × log(tam(a)) + tam(b) × log(tam(b)))
Descripción: guarda en c1 todos los elementos del c1 y c2

unionesConj(**in/out** c: conj(conj(α))) → res:conj(α)
Pre ≡ {$true$}
Post ≡ {(∀ a: α) a ∈ res ⇔ (∃ d:conj(α)) d ∈ c ∧$_L$ a ∈ d}
Complejidad: O(#(c) × ((#(c) × l) × log(#(c) × l)) (***l*** es la longitud del conjunto mas grande de **c**)
Descripción: guarda en **res** la union entre todos los conjuntos de **c**

Algoritmos

iUnionNaif(**in/out**$a : conj(\alpha)$, **in**$b : conj(\alpha)$)

itB ← crearIt(b)	O(1)
while(hayMas?(itb))	O(tam(b))
if ¬ pertenece(a, actual(itB)) **then**	O(tam(a))
agregarRapido(a, actual(itB))	O(1)
endif	O(1)
avanzar(itB)	O(1)
endwhile	O(1)

Complejidad: O(tam(b) × tam(a))

iUnionPosta(**in/out**$a : conj(\alpha)$, **in**$b : conj(\alpha)$)

itA ← crearIt(a) O(1)

itB ← crearIt(b)	O(1)
arregloA ← Vacía()	O(1)
arregloB ← Vacía()	O(1)
arregloRes ← Vacía()	O(1)
res ← Vacio()	O(1)
while(hayMas?(itA))	O(tam(a))
agregarAtras(arregloA, actual(itA))	O(1)
avanzar(itA)	O(1)
endwhile	O(1)
while(hayMas?(itB))	O(tam(b))
agregarAtras(arregloB, actual(itB))	O(1)
avanzar(itB)	O(1)
endwhile	O(1)
MergeSort(arregloA)	O(tam(a) × log(tam(a))))
MergeSort(arregloB)	O(tam(b) × log(tam(b))))
Merge(arregloRes, arregloA, arregloB)	O(tam(a) + tam(b))
agregarRapido(res, arregloRes[0])	O(1)
i ← 1	O(1)
a ← Vacío()	O(1)
while(i < tam(arregloRes))	O(tam(a) + tam(b))
if arregloRes[i] ! = arregloRes[i - 1] **then**	O(1)
agregarRapido(a, arregloRes[i])	O(1)
endif	
endwhile	O(1)

Complejidad:

O(tam(a) + tam(b) + tam(a) × log(tam(a)) + tam(b) × log(tam(b)) + tam(a) + tam(b))

O(tam(a) × (log(tam(a)) + 2) + tam(b) × (log(tam(b)) + 2)

O(tam(a) × log(tam(a)) + tam(b) × log(tam(b))

IUNIONESCONJ(**in/out** a: conj(conj(α))) → res:conj(α)

it ← crearIt(a)	O(1)
res ← Vacio()	O(1)
while haySiguiente?(it)	O(#(a) × cuerpo)
unionPosta(res, siguiente(it))	O(#(res) × log(#(res)) + l × log(l))
avanzar(it)	O(1)
endWhile	

Complejidad:

O(#(a) × (#(res) × log(#(res)) + l × log(l))

O(#(a) × ((#(a) × l) × log(#(a) × l) + l × log(l))

O(#(a) × ((#(a) × l) × log(#(a) × l))

donde l es el tamao del maximo conjunto que hay en **a**