Algoritmos y Estructuras de Datos II

Elección de Estructuras II

Matías Barbeito
2 de Octubre de 2015

Voto electrónico

Nos encargaron implementar un sistema de voto electrónico.

- Se cuenta con una base de datos de todos los votantes empadronados, con DNI y nombre completo. El nombre completo se escribe con la forma "APELLIDO(S), NOMBRE(S)".
- Cada persona vota con su DNI.
- En el sistema están cargados los nombres de los candidatos para la elección abierta.

Se desea realizar de manera eficiente cada voto, y saber cuántos votos acumuló cada candidato. Se tiene especial interés en obtener de manera eficiente, dado un apellido, todas las personas con ese apellido que no votaron.

Especificación

DNI es NAT, CANDIDATO,NOMBRE es STRING

TAD ELECCION

géneros eleccion

exporta géneros, generadores, observadores, operaciones

observadores básicos

faltanVotar	:	eleccion	$\longrightarrow$	dicc(DNI,nombre)	
candidatos	:	eleccion	$\longrightarrow$	conj(candidato)	
votos	:	candidato c $\times$ eleccion e	$\longrightarrow$	Nat	$\{c \in candidatos(e)\}$

generadores

crearEleccion	:	dicc(DNI,nombre) d conj(candidato) c	$\times$		$\longrightarrow$	eleccion	$\{\neg\emptyset?(c) \wedge \neg\emptyset?(d)\}$
votar	:	DNI d $\times$ candidato c $\times$ eleccion e			$\longrightarrow$	eleccion	$\{c \in candidatos(e) \wedge d \in faltanVotar(e)\}$

Fin TAD

Complejidades requeridas

Sean:

- C el nombre de candidato más largo,
- M el nombre más largo de persona, y
- k la cantidad de personas que falta votar,
- n es la cantidad de personas que falta votar con apellido a;

se desea que ciertas operaciones sean especialmente eficientes:

Complejidades requeridas

Sean:

- C el nombre de candidato más largo,
- M el nombre más largo de persona, y
- k la cantidad de personas que falta votar,
- n es la cantidad de personas que falta votar con apellido a;

se desea que ciertas operaciones sean especialmente eficientes:

1. **votos** en $O(|C|)$ *peor caso*,

Complejidades requeridas

Sean:

- C el nombre de candidato más largo,
- M el nombre más largo de persona, y
- k la cantidad de personas que falta votar,
- n es la cantidad de personas que falta votar con apellido a;

se desea que ciertas operaciones sean especialmente eficientes:

1. **votos** en $O(|C|)$ *peor caso*,
2. **votar** en $O(\log k + |C|)$ *peor caso*,

Complejidades requeridas

Sean:

- C el nombre de candidato más largo,
- M el nombre más largo de persona, y
- k la cantidad de personas que falta votar,
- n es la cantidad de personas que falta votar con apellido a;

se desea que ciertas operaciones sean especialmente eficientes:

1. **votos** en $O(|C|)$ *peor caso*,
2. **votar** en $O(\log k + |C|)$ *peor caso*,
3. obtener todos los que no votaron (nombre y DNI) con apellido a en $O(n|M|)$.

Complejidades requeridas

Sean:

- C el nombre de candidato más largo,
- M el nombre más largo de persona, y
- k la cantidad de personas que falta votar,
- n es la cantidad de personas que falta votar con apellido a;

se desea que ciertas operaciones sean especialmente eficientes:

1. **votos** en $O(|C|)$ *peor caso*,
2. **votar** en $O(\log k + |C|)$ *peor caso*,
3. obtener todos los que no votaron (nombre y DNI) con apellido a en $O(n|M|)$.

Algunas decisiones...

¿Cómo implemento devolver los que no votaron?

Algunas decisiones...

¿Cómo implemento devolver los que no votaron?

- Hay una diferencia entre la especificación y lo que nos piden para el diseño

Algunas decisiones...

¿Cómo implemento devolver los que no votaron?

- Hay una diferencia entre la especificación y lo que nos piden para el diseño
- Puedo implementar directamente la operación de "obtener los que no votaron con cierto apellido": Se pueden devolver todos si se da un apellido vacío.

Algunas decisiones...

¿Cómo implemento devolver los que no votaron?

- Hay una diferencia entre la especificación y lo que nos piden para el diseño
- Puedo implementar directamente la operación de "obtener los que no votaron con cierto apellido": Se pueden devolver todos si se da un apellido vacío.
- ¿Devuelvo un diccionario o un iterador?

Algunas decisiones...

¿Cómo implemento devolver los que no votaron?

- Hay una diferencia entre la especificación y lo que nos piden para el diseño
- Puedo implementar directamente la operación de "obtener los que no votaron con cierto apellido": Se pueden devolver todos si se da un apellido vacío.
- ¿Devuelvo un diccionario o un iterador?...En principio da lo mismo. Usemos un iterador.

Módulo **eleccion**: Interfaz (I)

Nueva(in votantes: dicc(DNI,nombre), in candidatos: conj(candidato)) $\rightarrow$ ret: eleccion
$Pre \equiv \{\neg Vacio(votantes) \wedge \neg Vacio(candidatos)\}$
$Post \equiv \{ret = crearEleccion(votantes, candidatos)\}$
Aliasing: No hay.
Descripción: Crea una votación nueva.

Módulo **eleccion**: Interfaz (I)

Nueva(in votantes: dicc(DNI,nombre), in candidatos: conj(candidato)) $\rightarrow$ ret: eleccion
$Pre \equiv \{\neg Vacio(votantes) \wedge \neg Vacio(candidatos)\}$
$Post \equiv \{ret = crearEleccion(votantes, candidatos)\}$
Aliasing: No hay.
Descripción: Crea una votación nueva.

Votar(inout e: eleccion, in votante: DNI, in voto: candidato)
$Pre \equiv \{e = e_0 \wedge votante \in faltanVotar(e) \wedge voto \in candidatos(e)\}$
$Post \equiv \{e = votar(votante, voto, e_0)\}$
Aliasing: No hay.
Descripción: Efactúa el voto de *votante* por el candidato *voto*.

Módulo **eleccion**: Interfaz (II)

Votos(inout e: eleccion, in quien: candidato) $\rightarrow$ ret: Nat
$Pre \equiv \{e = e_0 \wedge quien \in candidatos(e)\}$
$Post \equiv \{e = e_0 \wedge ret = votos(quien, e_0)\}$
Aliasing: No hay.
Descripción: Devuelve los votos obtenidos por el candidato *quien*.

Módulo **eleccion**: Interfaz (II)

Votos(inout e: eleccion, in quien: candidato) $\rightarrow$ ret: Nat
$Pre \equiv \{e = e_0 \wedge quien \in candidatos(e)\}$
$Post \equiv \{e = e_0 \wedge ret = votos(quien, e_0)\}$
Aliasing: No hay.
Descripción: Devuelve los votos obtenidos por el candidato *quien*.

...y cómo devolvemos los que no votaron?

Módulo **eleccion**: Interfaz (II)

Votos(inout e: eleccion, in quien: candidato) $\rightarrow$ ret: Nat
$Pre \equiv \{e = e_0 \wedge quien \in candidatos(e)\}$
$Post \equiv \{e = e_0 \wedge ret = votos(quien, e_0)\}$
Aliasing: No hay.
Descripción: Devuelve los votos obtenidos por el candidato *quien*.

...y cómo devolvemos los que no votaron?

NoVotaron(inout e: eleccion, in apellido: string) $\rightarrow$ ret: itDicc(nombre,DNI)
$Pre \equiv \{e = e_0\}$
$Post \equiv \{e =$
$e_0 \wedge alias(esPermutacion(ret, filtrarClavesPrefijo(apellido, faltanVotar(e_0))\}$
Aliasing: El iterador devuelto puede verse afectado por la modificación de la elección (en particular si un elector vota)
Descripción: Devuelve un iterador (no modificable) a un diccionario de los que no votaron con prefijo *apellido*.

¿Qué estructura elijo?

Miremos las complejidades más de cerca:

¿Qué estructura elijo?

Miremos las complejidades más de cerca:

- **Votos** en $O(|C|)$...

¿Qué estructura elijo?

Miremos las complejidades más de cerca:

- **Votos** en $O(|C|)$... sugiere un Diccionario sobre Trie con el conteo de votos.

¿Qué estructura elijo?

Miremos las complejidades más de cerca:

- **Votos** en $O(|C|)$... sugiere un Diccionario sobre Trie con el conteo de votos.
- **NoVotaron** en $O(n|M|)$...

¿Qué estructura elijo?

Miremos las complejidades más de cerca:

- **Votos** en $O(|C|)$... sugiere un Diccionario sobre Trie con el conteo de votos.
- **NoVotaron** en $O(n|M|)$... también! (ordenado por nombre)

¿Qué estructura elijo?

Miremos las complejidades más de cerca:

- **Votos** en $O(|C|)$... sugiere un Diccionario sobre Trie con el conteo de votos.
- **NoVotaron** en $O(n|M|)$... también! (ordenado por nombre)
- **Votar** en $O(log(k) + |C|)$...

¿Qué estructura elijo?

Miremos las complejidades más de cerca:

- **Votos** en $O(|C|)$... sugiere un Diccionario sobre Trie con el conteo de votos.
- **NoVotaron** en $O(n|M|)$... también! (ordenado por nombre)
- **Votar** en $O(log(k) + |C|)$... necesita los que no votaron ordenados por DNI!

¿Qué estructura elijo?

Miremos las complejidades más de cerca:

- **Votos** en $O(|C|)$... sugiere un Diccionario sobre Trie con el conteo de votos.
- **NoVotaron** en $O(n|M|)$... también! (ordenado por nombre)
- **Votar** en $O(log(k) + |C|)$... necesita los que no votaron ordenados por DNI!

¿¿Cómo mantenemos actualizados a los que NoVotaron respetando la complejidad de Votar??

¿Qué estructura elijo?

Miremos las complejidades más de cerca:

- **Votos** en $O(|C|)$... sugiere un Diccionario sobre Trie con el conteo de votos.
- **NoVotaron** en $O(n|M|)$... también! (ordenado por nombre)
- **Votar** en $O(log(k) + |C|)$... necesita los que no votaron ordenados por DNI!

¿¿Cómo mantenemos actualizados a los que NoVotaron respetando la complejidad de Votar??

...Podemos jugar con iteradores!!

¿Qué estructura elijo?

Miremos las complejidades más de cerca:

- **Votos** en $O(|C|)$... sugiere un Diccionario sobre Trie con el conteo de votos.
- **NoVotaron** en $O(n|M|)$... también! (ordenado por nombre)
- **Votar** en $O(log(k) + |C|)$... necesita los que no votaron ordenados por DNI!

¿¿Cómo mantenemos actualizados a los que NoVotaron respetando la complejidad de Votar??

...Podemos jugar con iteradores!!

Idea: Podemos mantener en una estructura iteradores que apunten a elementos de otra, y usarlos como "punteros".

Estructura propuesta (I)

ELECCION se representa con **estr** donde

estr es tupla con

Estructura propuesta (I)

ELECCION se representa con **estr** donde

estr es tupla con

- **NoVotaronPorDNI**: $Dicc(DNI, < nom : nombre, ref : itDicc(nombre, DNI) >)$,

Estructura propuesta (I)

ELECCION se representa con **estr** donde

estr es tupla con

- **NoVotaronPorDNI**: $Dicc(DNI, < nom : nombre, ref : itDicc(nombre, DNI) >)$,
- **NoVotaronPorNombre**: $Dicc(nombre, DNI)$,

Estructura propuesta (I)

ELECCION se representa con **estr** donde

estr es tupla con

- **NoVotaronPorDNI**: $Dicc(DNI, < nom : nombre, ref : itDicc(nombre, DNI) >)$,
- **NoVotaronPorNombre**: $Dicc(nombre, DNI)$,
- **Votos**: $Dicc(candidato, Nat))$

Estructura propuesta (I)

ELECCION se representa con **estr** donde

estr es tupla con

- **NoVotaronPorDNI**: $Dicc(DNI, < nom : nombre, ref : itDicc(nombre, DNI) >)$,
- **NoVotaronPorNombre**: $Dicc(nombre, DNI)$,
- **Votos**: $Dicc(candidato, Nat))$

Y necesitamos que:

Estructura propuesta (I)

ELECCION se representa con **estr** donde

estr es tupla con

- **NoVotaronPorDNI**: $Dicc(DNI, < nom : nombre, ref : itDicc(nombre, DNI) >)$,
- **NoVotaronPorNombre**: $Dicc(nombre, DNI)$,
- **Votos**: $Dicc(candidato, Nat))$

Y necesitamos que:

- **NoVotaronPorDNI** sea un diccionario implementado sobre AVL;

Estructura propuesta (I)

ELECCION se representa con **estr** donde

estr es tupla con

- **NoVotaronPorDNI**: $Dicc(DNI, < nom : nombre, ref : itDicc(nombre, DNI) >)$,
- **NoVotaronPorNombre**: $Dicc(nombre, DNI)$,
- **Votos**: $Dicc(candidato, Nat))$

Y necesitamos que:

- **NoVotaronPorDNI** sea un diccionario implementado sobre AVL;
- **NoVotaronPorNombre** y **Votos**: sean diccionarios implementados sobre Trie;

Estructura propuesta (I)

ELECCION se representa con **estr** donde

estr es tupla con

- **NoVotaronPorDNI**: $Dicc(DNI, < nom : nombre, ref : itDicc(nombre, DNI) >)$,
- **NoVotaronPorNombre**: $Dicc(nombre, DNI)$,
- **Votos**: $Dicc(candidato, Nat))$

Y necesitamos que:

- **NoVotaronPorDNI** sea un diccionario implementado sobre AVL;
- **NoVotaronPorNombre** y **Votos**: sean diccionarios implementados sobre Trie;
- **NoVotaronPorNombre** borre claves en $O(1)$;

Estructura propuesta (I)

ELECCION se representa con **estr** donde

estr es tupla con

- **NoVotaronPorDNI**: $Dicc(DNI, < nom : nombre, ref : itDicc(nombre, DNI) >)$,
- **NoVotaronPorNombre**: $Dicc(nombre, DNI)$,
- **Votos**: $Dicc(candidato, Nat))$

Y necesitamos que:

- **NoVotaronPorDNI** sea un diccionario implementado sobre AVL;
- **NoVotaronPorNombre** y **Votos**: sean diccionarios implementados sobre Trie;
- **NoVotaronPorNombre** borre claves en $O(1)$;
- **NoVotaronPorNombre** tenga una operación "ObtenerSufijos" que nos de un itDicc a los sufijos de un string dado. Debería sólo dejar iterar los sufijos del string, y no el resto...

Estructura propuesta (I)

ELECCION se representa con **estr** donde

estr es tupla con

- **NoVotaronPorDNI**: $Dicc(DNI, < nom : nombre, ref : itDicc(nombre, DNI) >)$,
- **NoVotaronPorNombre**: $Dicc(nombre, DNI)$,
- **Votos**: $Dicc(candidato, Nat))$

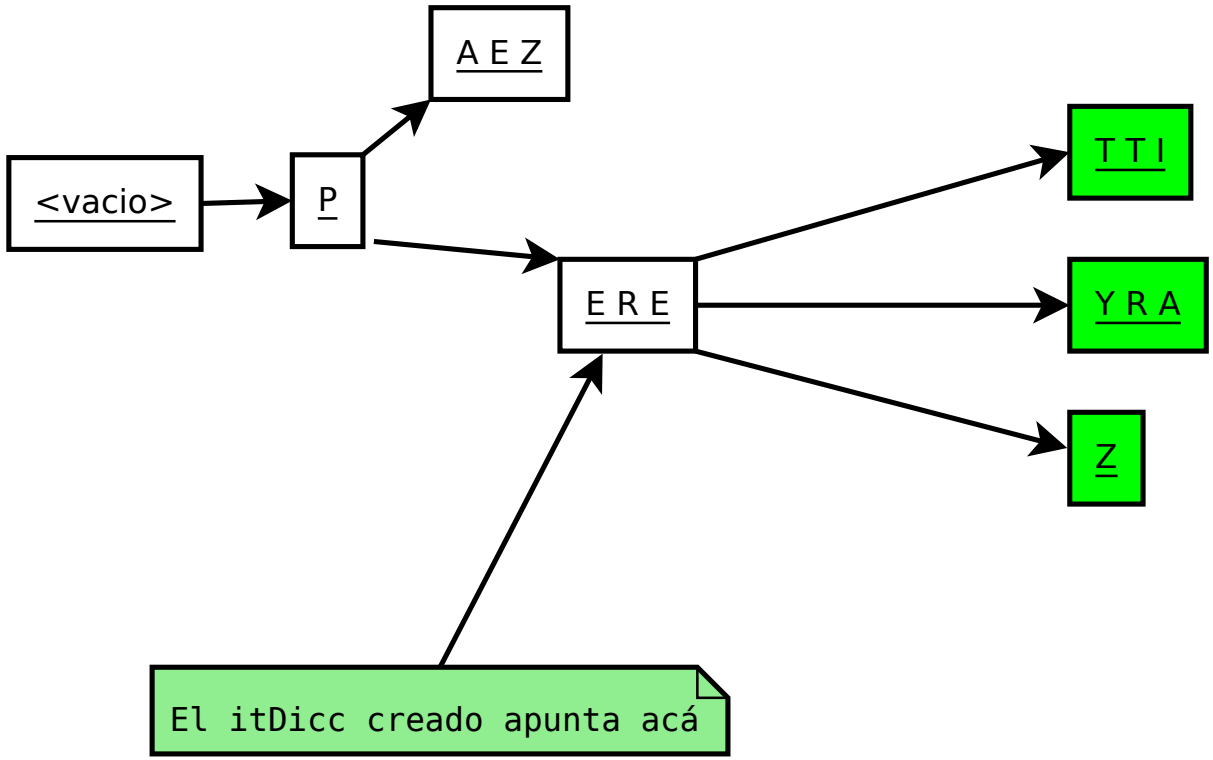
Y necesitamos que:

- **NoVotaronPorDNI** sea un diccionario implementado sobre AVL;
- **NoVotaronPorNombre** y **Votos**: sean diccionarios implementados sobre Trie;
- **NoVotaronPorNombre** borre claves en $O(1)$;
- **NoVotaronPorNombre** tenga una operación "ObtenerSufijos" que nos de un itDicc a los sufijos de un string dado. Debería sólo dejar iterar los sufijos del string, y no el resto...

Nos vamos a abstraer *un poco* de cómo implementar los requisitos recién mencionados.

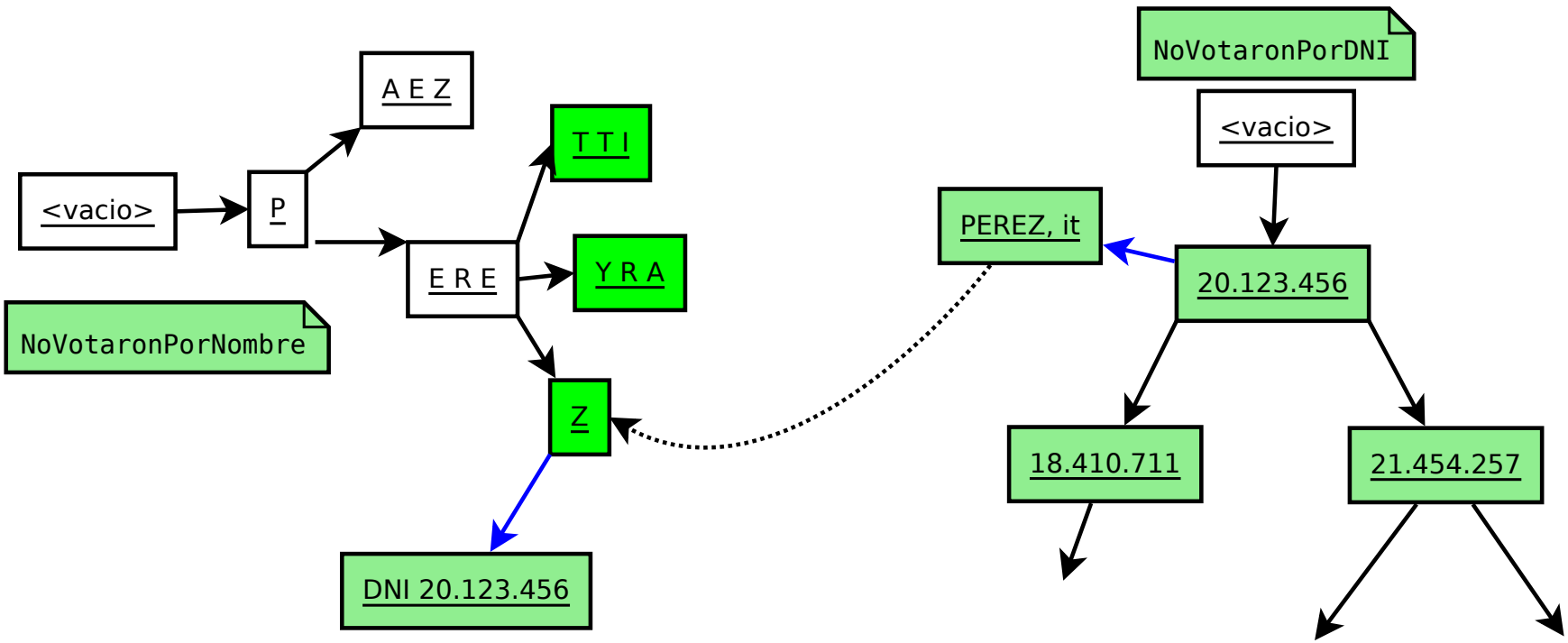
Estructura propuesta (II)

Sufijos de "PERE" en el Trie



Estructura propuesta (III)

NoVotaronPorDNI y NoVotaronPorNombre



Algoritmos: NoVotaron y Votos

iVotos(inout e: eleccion, in voto: candidato) $\rightarrow$ ret: Nat

1: $ret \leftarrow Obtener(e.Votos, voto)$

Algoritmos: NoVotaron y Votos

iVotos(inout e: eleccion, in voto: candidato) $\rightarrow$ ret: Nat

1: $ret \leftarrow Obtener(e.Votos, voto)$

iNoVotaron(inout e: eleccion, in apellido: string) $\rightarrow$ ret: itDicc(nombre, DNI)

$ret \leftarrow ObtenerPrefijos(e.NoVotaronPorNombre, apellido)$

Algoritmos: NoVotaron y Votos

iVotos(inout e: eleccion, in voto: candidato) $\rightarrow$ ret: Nat

1: $ret \leftarrow Obtener(e.Votos, voto)$

iNoVotaron(inout e: eleccion, in apellido: string) $\rightarrow$ ret: itDicc(nombre, DNI)

$ret \leftarrow ObtenerPrefijos(e.NoVotaronPorNombre, apellido)$

Algoritmo: Votar

iVotar(inout e: eleccion, in votante: DNI, in voto: candidato)

1: $Definir(e.Votos, voto, Obtener(e.Votos, voto) + 1)$
2: $EliminarSiguiente(Obtener(e.NoVotaronPorDNI, votante).ref)$
3: $Borrar(e.NoVotaronPorDNI, votante)$

Algoritmo: Votar

iVotar(inout e: eleccion, in votante: DNI, in voto: candidato)

1: $Definir(e.Votos, voto, Obtener(e.Votos, voto) + 1)$
2: $EliminarSiguiente(Obtener(e.NoVotaronPorDNI, votante).ref)$
3: $Borrar(e.NoVotaronPorDNI, votante)$

Complejidades:
1: Definir, Obtener: $O(|voto|)$, $O(|voto|)$

Algoritmo: Votar

iVotar(inout e: eleccion, in votante: DNI, in voto: candidato)

1: $Definir(e.Votos, voto, Obtener(e.Votos, voto) + 1)$
2: $EliminarSiguiente(Obtener(e.NoVotaronPorDNI, votante).ref)$
3: $Borrar(e.NoVotaronPorDNI, votante)$

Complejidades:
1: Definir, Obtener: $O(|voto|)$, $O(|voto|)$
2: Obtener,EliminarSiguiente: $O(\log k)$,$O(1)$

Algoritmo: Votar

iVotar(inout e: eleccion, in votante: DNI, in voto: candidato)

1: $Definir(e.Votos, voto, Obtener(e.Votos, voto) + 1)$
2: $EliminarSiguiente(Obtener(e.NoVotaronPorDNI, votante).ref)$
3: $Borrar(e.NoVotaronPorDNI, votante)$

Complejidades:
1: Definir, Obtener: $O(|voto|)$, $O(|voto|)$
2: Obtener,EliminarSiguiente: $O(\log k)$,$O(1)$
2: Borrar: $O(\log k)$

Algoritmo: Votar

iVotar(inout e: eleccion, in votante: DNI, in voto: candidato)

1: $Definir(e.Votos, voto, Obtener(e.Votos, voto) + 1)$
2: $EliminarSiguiente(Obtener(e.NoVotaronPorDNI, votante).ref)$
3: $Borrar(e.NoVotaronPorDNI, votante)$

Complejidades:
1: Definir, Obtener: $O(|voto|)$, $O(|voto|)$
2: Obtener,EliminarSiguiente: $O(\log k)$,$O(1)$
2: Borrar: $O(\log k)$
Total: $O(|voto| + \log k)$

Algoritmo: Votar

¡Votar(inout e: eleccion, in votante: DNI, in voto: candidato)

1: $Definir(e.Votos, voto, Obtener(e.Votos, voto) + 1)$
2: $EliminarSiguiente(Obtener(e.NoVotaronPorDNI, votante).ref)$
3: $Borrar(e.NoVotaronPorDNI, votante)$

Complejidades:
1: Definir, Obtener: $O(|voto|)$, $O(|voto|)$
2: Obtener,EliminarSiguiente: $O(\log k)$,$O(1)$
2: Borrar: $O(\log k)$
Total: $O(|voto| + \log k)$
¡Notar que el paso (2) está actualizando el diccionario NoVotaronPorNombre!

Algoritmo: Nueva Eleccion

iNueva(in votantes: dicc(DNI,nombre), in candidatos: conj(candidato)) $\rightarrow$ ret: eleccion

$itc : itDicc(candidato, Nat)$
$itc \leftarrow CrearIt(candidatos)$
Mientras $HaySiguiente(itc)$:
    $Definir(ret.Votos, Siguiente(itc), 0)$
    $Avanzar(itc)$

$it : itDicc(DNI, nombre)$
$it \leftarrow CrearIt(votantes)$
Mientras $HaySiguiente(it)$:
    $it3 : itDicc(nombre, DNI)$
    $it3 \leftarrow Definir(ret.NoVotaronPorNombre, SiguienteSignificado(it), SiguienteClave(it))$
    $Definir(ret.NoVotaronPorDNI, SiguienteClave(it), < SiguienteSignificado(it), it3 >)$
    $Avanzar(it)$

Listo!

¿Preguntas?