Algoritmos y Estructuras de Datos - DC - UBA

Segundo cuatrimestre de 2023

Trabajo práctico 2: Diseño e implementación de estructuras

Normativa

Límite de entrega: Sábado 11 de Noviembre a las 23:59hs.

Normas de entrega: Subir el archivo `SistemaCNE.java`, y todos los archivos de otras clases que hagan, al repositorio grupal.

Versión: 3

Cambios con respecto a la versión 2:

- El proc distritoDeMesa devuelve el nombre del distrito y no su ID, de forma que concuerde con el resto del enunciado y con los tests.
- Algunos tests declaraban $n + 1$ nombres de partidos y luego pasaban votos para n partidos. Se arregló esto de forma que ambos números coincidan.

Cambios con respecto a la versión 1:

- En el constructor, en un lugar decía `primerasMesasDistritos` pero debía ser `ultimasMesasDistritos`
- En el constructor, en un lugar tomaba `seq` pero debía tomar `Array`
- Se cambió los `==` por `=`

Enunciado

El trabajo práctico consiste en el diseño de módulos que implementen el sistema de la *Cámara Nacional Electoral* (`SistemaCNE`), cuya especificación encontrarán en la siguiente sección. La solución propuesta debe cumplir las restricciones que se imponen sobre la complejidad temporal de las operaciones, detalladas a continuación. Usamos las siguientes variables:

- P: cantidad de partidos políticos.
- D: cantidad de distritos (provincias).
- D_d: cantidad de bancas de diputados en el distrito d. Cada distrito d puede tener una cantidad D_d distinta.

Nota: No se puede asumir como constante ninguna de las variables que se mencionan. Por ejemplo, no se puede asumir una determinada cantidad de provincias o partidos políticos.

Deberán entregar al menos una clase `Java` que implemente la solución al problema y que tenga comentado, en lenguaje natural/semiformal, el Invariante de Representación. Recomendamos (y valoraremos) modularizar la solución en varias clases.

Operaciones a implementar

1. `nuevoSistema`: Inicializa el sistema de la Cámara Nacional Electoral. $O(P * D)$
2. `nombrePartido`: Devuelve el nombre del partido a partir de su id. $O(1)$
3. `nombreDistrito`: Devuelve el nombre del distrito a partir de su id. $O(1)$
4. `diputadosEnDisputa`: Devuelve la cantidad de bancas de diputados en disputa en un distrito dado. $O(1)$
5. `distritoDeMesa`: Devuelve el nombre del distrito al que pertenece una mesa. $O(log(D))$
6. `registrarMesa`: Registra los votos de una mesa. $O(P + log(D))$
7. `votosPresidenciales`: Devuelve la cantidad de votos que obtuvo un partido sobre su candidata/o a presidente a nivel nacional. $O(1)$
8. `votosDiputados`: Devuelve la cantidad de votos que obtuvo un partido en sus diputados en un distrito dado. $O(1)$
9. `resultadosDiputados`: Devuelve la cantidad de bancas que obtuvo cada partido en un distrito dado d. $O(D_d * log(P))$

10. `hayBallotage`: Devuelve *true* si hay ballotage, *false* en caso contrario. $O(1)$

IMPORTANTE: No se manden a programar sin pensar antes una solución al problema. La idea es que piensen "en papel" una estructura de representación que permita cumplir las complejidades y la consulten con su corrector durante la clase. Una vez que su corrector les de el OK, pueden comenzar a programar la solución.

Aclaraciones

- Como en el primer TP, los IDs de los partidos van entre 0 y P-1. El ID número P se reserva para los votos en blanco.
- Notar que las mesas electorales por cada distrito son **intervalos ordenados**. Es decir, el distrito 0 tiene las mesas $[0, k_1 - 1]$, el distrito 1 las mesas $[k_1, k_2 - 1]$, etc.
- Recuerden que las implementaciones deben respetar la especificación, pero se puede implementar mediante una idea distinta a la planteada en la especificación.

Condiciones de entrega y aprobación

La entrega deberá consistir de un archivo `SistemaCNE.java` que implemente la solución al problema y el cual tenga comentado, en lenguaje natural/semiformal, el Invariante de Representación. El mismo deberá ser subido al repositorio grupal correspondiente antes de la fecha y hora límite de entrega. Podrán hacer otras clases en otros archivos `.java`, los cuales deberán implementar, comentar su Invariante de Representación y entregar de la misma manera.

Para la aprobación del trabajo práctico, la implementación debe superar todos los tests provistos y, además, debe cumplir con las complejidades temporales especificadas en la sección anterior. Se debe dejar comentado (de forma breve y concisa) la justificación de la complejidad obtenida. Solamente pueden utilizar las estructuras vistas en la teórica hasta ahora (arreglo, vector, lista enlazada, ABB, AVL, heap, trie), implementadas por ustedes mismos. Pueden usar el código de las estructuras que hicieron para los talleres. **No se puede usar ninguna clase predefinida en la biblioteca estándar de Java**. `Recomendamos (y valoraremos) modularizar la solución en varias clases`. También evaluaremos la claridad del código, del Invariante de Representación y de las justificaciones de las complejidades.

Tests de tiempo

La suite de tests incluye tres tests que tienen la expresión `@Timeout` y cuyo nombre comienza con `complejidad`. `@Timeout(n)` hace que un test falle si tarda más de n segundos en ejecutar. Estos tests están para comprobar que su solución cumplan con las cotas de complejidad de los principales métodos. Si bien corregiremos cada TP manualmente para comprobar que cumplan y justifiquen las complejidades, estos tests sirven para darles mayor seguridad de que cumplen lo pedido. Como siempre, los tests **no** son una demostración de correctitud.

Especificación

```
VotosPartido es <presidente: nat, diputados: nat>
ActaMesa es Array<VotosPartido>
IdPartido es nat
IdDistrito es nat

TAD SistemaCNE {
    obs nombresPartidos: dict<IdPartido, string>
    obs nombresDistritos: dict<IdDistrito, string>
    obs diputadosDeDistritos: dict<IdDistrito, nat>
    obs rangoMesasDistritos: dict<IdDistrito, <nat, nat>>
    obs votosPresidenciales: dict<IdPartido, nat>
    obs votosDiputados: dict<IdDistrito, dict<IdPartido, nat>>
    obs mesasRegistradas: conj<nat>

    proc nuevoSistema(in nombresDistritos: Array<string>,
                      in diputadosPorDistrito: Array<nat>,
                      in nombresPartidos: Array<string>,
                      in ultimasMesasDistritos: Array<nat>): SistemaCNE {
        requiere sinRepetidos(nombresDistritos) && sinRepetidos(nombresPartidos)
        requiere |nombresDistritos| = |diputadosPorDistrito|
        requiere |ultimasMesasDistritos| = |nombresDistritos|
                 && estrictamenteCreciente(ultimasMesasDistritos)
        requiere nombresPartidos[|nombresPartidos|−1] = "Blanco"

        asegura sonLasClaves({0..|nombresPartidos|−1}, res.nombresPartidos)
        asegura sonLasClaves({0..|nombresPartidos|−1}, res.votosPresidenciales)
        asegura forall idPartido: nat :: 0 <= idPartido < |nombresPartidos| ==>L
            res.nombresPartidos[idPartido] = nombresPartidos[idPartido]
            && res.votosPresidenciales[idPartido] = 0
        asegura sonLasClaves({0..|nombresDistritos|−1}, res.nombresDistritos)
        asegura sonLasClaves({0..|nombresDistritos|−1}, res.diputadosDeDistritos)
        asegura sonLasClaves({0..|nombresDistritos|−1}, res.rangoMesasDistritos)
        asegura forall idDistrito: nat :: 0 <= idDistrito < |nombresDistritos| ==>L
            res.nombresDistritos[idDistrito] = nombresDistritos[idDistrito]
            && res.diputadosDeDistritos[idDistrito] = diputadosPorDistrito[idDistrito]
            && res.rangoMesasDistritos[idDistrito] = if idDistrito = 0
                    then <0, ultimasMesasDistritos[0]>
                    else <ultimasMesasDistritos[idDistrito − 1],
                          ultimasMesasDistritos[idDistrito]> fi
            && forall idPartido: nat :: 0 <= idPartido < |nombresPartidos| ==>L
                    res.votosDiputados[idDistrito][idPartido] = 0
        asegura mesasRegistradas = {}
    }

    pred sonLasClaves(claves:conj<K>, d:dict<K, V>) {
        forall c: K :: c in claves <==> c in d
    }

    pred estrictamenteCreciente(secu: seq<nat>) {
        forall i: nat :: 0 <= i < |secu| − 1 ==>L secu[i] < secu[i + 1]
    }

    pred sinRepetidos(secu: seq<T>) {
        forall i: nat :: 0 <= i < |secu| ==>L
            forall j: nat :: 0 <= j < |secu| && j != i ==>L
                secu[i] != secu[j]
    }

    proc nombrePartido(in sistema: SistemaCNE, in idPartido: nat): string {
        requiere idPartido in sistema.nombresPartidos
        asegura res = sistema.nombresPartidos[idPartido]
    }

    proc nombreDistrito(in sistema: SistemaCNE, in idDistrito: nat): string {
```

```
    requiere idDistrito in sistema.idsDistritos
    asegura res = sistema.nombresDistritos[idDistrito]
}

proc diputadosEnDisputa(in sistema: SistemaCNE, in idDistrito: nat): nat {
    requiere idDistrito in sistema.diputadosDeDistritos
    asegura res = sistema.diputadosDeDistritos[idDistrito]
}

proc distritoDeMesa(in sistema: SistemaCNE, in idMesa: nat): string {
    requiere exists idDistrito: nat :: idDistritos in sistema.rangoMesasDistritos
                                    &&L mesaEnRango(sistema, idDistrito, idMesa)
    asegura exists idDistrito: IdDistrito :: mesaEnRango(sistema, idDistrito, idMesa) &&L res = sistema.nombresDistritos[idDistrito]
}

pred mesaEnRango(sistema: SistemaCNE, idDistrito: nat, idMesa: nat) {
    sistema.rangoMesasDistritos[idDistrito][0] <= idMesa
    && idMesa < sistema.rangoMesasDistritos[idDistrito][1]
}

proc registrarMesa(inout sistema: SistemaCNE,
                   in idMesa: nat,
                   in resultados: ActaMesa) {
    requiere exists idDistrito: IdDistrito :: idDistrito in sistema.rangoMesasDistritos :: mesaEnRango(sistema, idDistrito, idMesa)
    requiere |resultados| = |sistema.nombresPartidos|
    requiere !(idMesa in sistema.mesasRegistradas)

    asegura sistema.nombresPartidos = old(sistema).nombresPartidos
    asegura sistema.nombresDistritos = old(sistema).nombresDistritos
    asegura sistema.diputadosDeDistritos = old(sistema).diputadosDeDistritos
    asegura sistema.rangoMesasDistritos = old(sistema).rangoMesasDistritos
    asegura forall idPartido:IdPartido :: idPartido in sistema.votosPresidenciales ==>L
            sistema.votosPresidenciales[idPartido] = old(sistema).votosPresidenciales[idPartido]
                                                   + resultados[idPartido].presidente
    asegura forall idDistrito:IdDistrito :: idDistrito in sistema.votosDiputados ==>L
            forall idPartido:IdPartido :: idPartido in sistema.votosDiputados[idDistrito] ==>L
                (sistema.votosDiputados[idDistrito][idPartido] = old(sistema).votosDiputados[idDistrito][idPartido]
                        && !mesaEnRango(sistema, idDistrito, idMesa))
                || (sistema.votosDiputados[idDistrito][idPartido] = old(sistema).votosDiputados[idDistrito][idPartido]
                                                                  + resultados[idPartido].diputados
                        && mesaEnRango(sistema, idDistrito, idMesa))
    asegura sistema.mesasRegistradas = old(sistema).mesasRegistradas + {idMesa}
}

proc votosPresidenciales(in sistema : SistemaCNE, in idPartido: nat): nat {
    requiere idPartido in sistema.votosPresidenciales
    asegura res = sistema.votosPresidenciales[idPartido]
}
proc votosDiputados(in sistema: SistemaCNE, in idPartido: IdPartido, in idDistrito: IdDistrito): nat {
    requiere idDistrito in sistema.votosDiputados
    requiere idPartido in sistema.votosDiputados[idDistrito]
    asegura res = sistema.votosDiputados[idDistrito][idPartido]
}

proc resultadosDiputados(in sistema: SistemaCNE, in idDistrito: nat): Array<nat> {
    requiere idDistrito in sistema.votosDiputados
    requiere exists dhondt: seq<seq<nat>> :: dHondtValida(dhondt, idDistrito, sistema)

    asegura |res| = |sistema.nombresPartidos| - 1
    asegura sumaTotal(res) = sistema.diputadosDeDistritos[idDistrito]
    asegura forall dhondt: seq<seq<nat>> :: dHondtValida(dhondt, idDistrito, sistema) ==>L
            forall maximos: seq<nat> :: n_maximos(sistema.diputadosDeDistritos[idDistrito], dhondt, maximos) ==>L
                forall i: nat :: 0 <= i < |res| ==>L
                    res[i] = contarMaximos(maximos, dhondt[i])
}

aux sumaTotal(s:seq<nat>) = sum i: nat :: 0 <= i < |res| :: res[i]
```

```
    pred dHondtValida(dhondt: seq<seq<nat>>, idDistrito: nat, sistema: SistemaCNE) {
        |dhondt| = |sistema.nombresPartidos| − 1
        && forall i : nat :: i < |dhondt| ==>L
            (partidoSuperaUmbral(sistema, i, idDistrito) && |dhondt[i]| = sistema.diputadosDeDistritos[idDistrito])
            || (!partidoSuperaUmbral(sistema, i, idDistrito) && |dhondt[i]| = 0)
            && forall j : nat :: j < |dhondt[i]| ==>L
                dhondt[i][j] = sistema.diputadosDeDistritos[idDistrito][i] / (j + 1)
        && sinRepetidosDHondt(dhondt)
    }

    pred sinRepetidosDHondt(dhondt: seq<seq<nat>>) {
        forall p1, p2, a, b :: p1, p2 < |dhondt| && a < |dhondt[p1]| && b < |dhondt[p2]| ==>L
            dhondt[p1][a] != dhondt[p2][b]
    }

    pred n_maximos(n: nat, matriz: seq<seq<nat>>, maximos: seq<nat>) {
        |maximos| = n && sinRepetidos(maximos)
        && forall elem: nat :: elem in maximos ==>L exists j: nat :: j < |matriz| &&L elem in matriz[j]
            && forall i: nat :: i < |matriz| ==>L
                forall j: nat :: j < |matriz[k]| ==>L
                    matriz[i][j] in maximos || matriz[i][j] <= elem
    }

    aux contarMaximos(maximos:seq<nat>, dhondt:seq<seq<nat>>) =
        sum elem: nat :: elem in maximos :: if elem in dhondt[i] then 1 else 0 fi

    pred partidoSuperaUmbral(sistema: SistemaCNE, idPartido: nat, idDistrito: nat) {
        sistema.votosDiputados[idDistrito][idPartido] * 100 / votosTotales(sistema.votosDiputados[idDistrito]) >= 3
    }

    aux votosTotales(votosPorPartido: dict<IdPartido, nat>) =
        sum idPartido:IdPartido :: idPartido in votosPorPartido :: votosPorPartido[idPartido]

    proc hayBallotage(in sistema: SistemaCNE): bool {
        asegura res = true <==> !(exists idPartido: nat :: idPartido < |sistema.idsPartidos| − 1 &&L
                                  supera45(sistema, idPartido) || supera40ConDifDe10(sistema, idPartido))
    }

    pred supera45(sistema: SistemaCNE, idPartido: nat) {
        sistema.votosPresidenciales[idPartido] * 100 / votosTotales(sistema.votosPresidenciales) >= 45
    }

    pred supera40ConDifDe10(sistema: SistemaCNE, idPartido: nat) {
        sistema.votosPresidenciales[idPartido] * 100 / votosTotales(sistema.votosPresidenciales) >= 40
        && !(exists otroPartido: nat :: otroPartido in sistema.idsPartidos &&L
                otroPartido != idPartido &&
                (sistema.votosPresidenciales[idPartido] − sistema.votosPresidenciales[otroPartido])
                    * 100 / votosTotales(sistema) <= 10)
    }
}
```