Algoritmos y Estructuras de Datos - DC - UBA

Primer cuatrimestre de 2024

Trabajo práctico 2: Diseño e implementación de estructuras

Normativa

Revisión de mitad de TP: viernes 14 de junio, en el horario de la clase.
Límite de entrega: sábado 22 de junio a las 23:59.
Límite de re-entrega: sábado 14 de julio a las 23:59.
Forma de entrega: Subir el archivo `SistemaSIU.java`, y todos los archivos de otras clases que hagan, al campus.

Enunciado

A raíz de la falta de actualizaciones de los salarios docentes con respecto a la inflación reciente (lo cual motivó un aumento en las renuncias que está recibiendo el Departamento de Computación), desde las autoridades se nos pidió rediseñar el actual sistema de inscripciones *SIU Guaraní* (`SistemaSIU`). Este será un sistema simplificado que permita definir carreras de grado, junto con sus respectivas materias y planteles docentes, sobre las cuales estudiantes se puedan inscribir a través de sus libretas universitarias (las cuales tienen largo fijo).

El sistema debe permitir el agregado de docentes a los planteles de las materias, así como establecer un cupo sobre cada materia en base a la cantidad de docentes disponibles. Se estimó que debe haber a lo sumo 250 estudiantes por cada profesor, 100 estudiante por cada JTP, 20 estudiantes por cada Ay1 y 30 por cada Ay2.[1] En caso de que haya más que esa cantidad de estudiantes por cada cargo docente, se considerará que el cupo está excedido.

Por otro lado, como es sabido, una misma materia puede tener varios nombres, los cuales dependen de la carrera de grado. Por ejemplo, nuestra materia se llama "Algoritmos y Estructuras de Datos 2" (para Lic. en Cs. de Datos) y "Algoritmos y Estructuras de Datos" (para Lic. en Cs. de la Computación)

La especificación del sistema se encuentra al final del documento.

Aclaraciones

- Los estudiantes son identificados con un string (su libreta universitaria, o LU) de longitud acotada.
- Las materias y carreras son identificadas con un string (su nombre) de longitud **no** acotada.
- El orden lexicográfico sigue el orden ASCII, donde las mayúsculas preceden a las minúsculas y los números a las letras.

La solución propuesta debe cumplir las restricciones que se imponen sobre la complejidad temporal de las operaciones, detalladas a continuación. Usamos las siguientes variables:

- C: conjunto de las carreras de grado.
- M: conjunto de todas las materias.
- E: cantidad total de estudiantes en todas las carreras de grado.
- M_c: conjunto de materias de la carrera de grado c.
- N_m: conjunto de nombres de la materia m.
- E_m: cantidad de estudiantes inscriptos en la materia m.

La entrega debe incluir *al menos* una clase `Java` que implemente la solución al problema y que tenga comentado, en lenguaje natural/semiformal, el Invariante de Representación de cada clase que implementen. Recomendamos (y valoraremos) modularizar la solución en varias clases.

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.

[1] Aproximación ficticia a modo ilustrativa para el trabajo práctico.

Operaciones a implementar

1. `nuevoSistema(in infoMaterias: seq<InfoMateria>, in libretasUniversitarias:seq<string>): SistemaSIU`
   inicializa el sistema SIU Guaraní. $O(\sum_{c \in C} |c| * |M_c| + \sum_{m \in M} \sum_{n \in N_m} |n| + E)$

   Donde:

   `ParCarreraMateria es <carrera: string, nombreMateria: string>`

   `InfoMateria es seq<ParCarreraMateria>`

2. `inscribir(inout sistema: SistemaSIU, in estudiante: string,`
   `in carrera: string, in nombreMateria: string)`
   inscribe al estudiante en la materia m de la carrera c. $O(|c| + |m|)$

3. `inscriptos(in sistema: SistemaSIU, in carrera: string, in nombreMateria: string): ℕ`
   dada una carrera c y una materia m, devuelve la cantidad de estudiantes inscriptos en la materia, incluyendo a los estudiantes de todas las carreras. $O(|c| + |m|)$

4. `agregarDocente(inout sistema: SistemaSIU, in cargo: CargoDocente,`
   `in carrera: string, in nombreMateria: string)`
   agrega a un docente al plantel de la materia m de la carrera c, y al plantel de todas las otras materias que son sus equivalentes en el resto de las carreras. $O(|c| + |m|)$

5. `plantelDocente(in sistema: SistemaSIU, in carrera: string, in nombreMateria: string): seq<ℕ>`
   devuelve un arreglo de enteros donde sus posiciones representan (en orden) la cantidad de profesores, JTPs, AY1 y AY2, respectivamente, para la materia m dada la carrera c. $O(|c| + |m|)$

6. `excedeCupo?(in sistema: SistemaSIU, in carrera: string, in nombreMateria: string): bool`
   devuelve verdadero si la cantidad de inscriptos en la materia m de la carrera c excede el cupo. $O(|c| + |m|)$

7. `carreras(in sistema: SistemaSIU): seq<string>`
   devuelve el listado de carreras de grado, ordenado lexicográficamente $O(\sum_{c \in C} |c|)$

8. `materias(in sistema: SistemaSIU, in carrera: string): seq<string>`
   devuelve el listado de materias de una carrera c, ordenado lexicográficamente. $O(|c| + \sum_{m_c \in M_c} |m_c|)$.

9. `materiasInscriptas(in sistema: SistemaSIU, in estudiante: string): ℕ`
   devuelve la cantidad de materias en las que se encuentra inscripto/a un/a estudiante. $O(1)$

10. `cerrarMateria(inout sistema: SistemaSIU, in carrera: string, in nombreMateria: string)`
    a partir de una carrera c, cierra la materia m para todas las carreras de grado por falta de docentes. $O(|c| + |m| + \sum_{n \in N_m} |n| + E_m)$

Condiciones de entrega y aprobación

Durante la clase del día viernes 14 de junio deberán hablar con su corrector asignado para la **revisión obligatoria** de mitad de TP. En esta, deben mostrarle la estructura de representación propuesta para resolver el problema. Al finalizar esta clase, deben salir con una estructura que funcione para cumplir las complejidades pedidas.

La entrega deberá incluir el archivo `SistemaSIU.java`, el cual implementará la solución al problema y tendrá comentado, en lenguaje natural/semiformal, el Invariante de Representación. Las demás clases diseñadas para la solución se deben incluir en otros archivos `.java` y también deben tener su correspondiente Invariante de Representación. Todos estos archivos deben ser subidos al campus antes de la fecha y hora límite de entrega.

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, con la excepción de ArrayList, String y StringBuffer**. También evaluaremos la claridad del código, del Invariante de Representación y de las justificaciones de las complejidades.

Especificación

```
CargoDocente es PROFE ó JTP ó AY1 ó AY2
ParCarreraMateria es <carrera: string, nombreMateria: string>
InfoMateria es seq<ParCarreraMateria>

TAD SistemaSIU {
  obs carreras: conj(string)
  obs estudiantes: conj(string)
  obs materias(carrera: string): conj(string)
  obs inscriptos(carrera: string, nombreMateria: string): conj<string>
  obs inscripciones(estudiante: string): ℕ
  obs docentes(cargo: CargoDocente, carrera: string, nombreMateria: string): ℕ
  obs sonMismaMateria(carrera1: string, nombreMateria1: string
                      carrera2: string, nombreMateria2: string): bool

  pred sinRepetidos(s: seq<T>)
    {(∀i, j : ℕ) (i < |s| ∧ j < |s| ∧ i ≠ j ⟶_L s[i] ≠ s[j]) }

  pred ordenadoEstricto(s: seq<T>)
    {(∀i : ℕ) (i < |s| − 1 ⟶_L s[i] < s[i + 1]) }

  pred igualdadSeqConj(s: seq<T>, c: conj<T>)
    {(∀t : T) (t ∈ s ⟺ t ∈ c ∧ sinRepetidos(s)) }

  aux cupo(sistema: SistemaSIU, carrera: string, nombreMateria: string)=
     min(
          250 · sistema.docentes(PROFE, carrera, nombreMateria),
          100 · sistema.docentes(JTP, carrera, nombreMateria),
          20 · sistema.docentes(AY1, carrera, nombreMateria),
          30 · sistema.docentes(AY2, carrera, nombreMateria),
     )

  proc nuevoSistema(in infoMaterias: seq<InfoMateria>,
                    in libretasUniversitarias: seq<string>): SistemaSIU
    requiere {sinRepetidos(infoMaterias) ∧ sinRepetidos(libretasUniversitarias)}
    requiere {¬(∃ infoMateria1, infoMateria2: InfoMateria)(
                infoMateria1 ∈ infoMaterias ∧ infoMateria2 ∈ infoMaterias ∧
                infoMateria1 ≠ infoMateria2 ∧ (∃ carrera, nombreMateria: string)(
                     <carrera, nombreMateria>∈ infoMateria1 ∧
                     <carrera, nombreMateria>∈ infoMateria2)
           )}
    asegura {
      (∀ carrera: string)(
           carrera ∈ res.carreras ⟺ (∃ infoMateria: InfoMateria, p: ParCarreraMateria)(
                infoMateria ∈ infoMaterias ∧ p ∈ infoMateria ∧ p.carrera = carrera))
    }
    asegura {(∀ estudiante: string)(estudiante ∈ res.estudiantes ⟺
                           estudiante ∈ libretasUniversitarias)}
    asegura {
      (∀ carrera: string)(
           carrera ∈ res.carreras ⟶_L (∀ materia: string)(
                materia ∈ res.materias(carrera) ⟺ (∃ infoMateria: InfoMateria,
                                                   p: ParCarreraMateria)(
                     infoMateria ∈ infoMaterias ∧ p ∈ infoMateria ∧
                     p.carrera = carrera ∧ p.nombreMateria = materia)))
    }
    asegura {
      (∀ carrera, nombreMateria: string)(
           carrera ∈ res.carreras ∧_L nombreMateria ∈ materias(carrera) ⟶_L
                res.inscriptos(carrera, nombreMateria) = ⟨⟩)}
    asegura {
      (∀ estudiante: string)(
           estudiante ∈ res.estudiantes ⟶_L
                res.inscripciones(estudiante) = 0)}
```

```
    asegura {(∀ carrera, nombreMateria: string, cargo: CargoDocente)(
                res.docentes(cargo, carrera, nombreMateria) = 0
    )}
    asegura {
      (∀ c1, m1, c2, m2: string)(res.sonMismaMateria(c1, m1, c2, m2) ⟺
           (∃ infoMateria: InfoMateria)(
                infoMateria ∈ infoMaterias ∧ <c1,m1>∈ infoMaterias ∧ <c2,m2>∈ infoMaterias))
    }

proc inscribir(inout sistema: SistemaSIU, in estudiante: string,
               in carrera: string, in nombreMateria: string)
    requiere {carrera ∈ sistema.carreras}
    requiere {estudiante ∈ sistema.estudiantes}
    requiere {nombreMateria ∈ sistema.materias(carrera)}
    requiere {estudiante ∉ sistema.inscriptos(carrera, nombreMateria)}
    asegura {
      (∀ otraCarrera, otraMateria: string)(
           otraCarrera ∈ sistema.carreras ∧_L otraMateria ∈ sistema.materias(otraCarrera) ∧_L
           ¬ sistema.sonMismaMateria(carrera, nombreMateria, otraCarrera, otraMateria) ⟶_L
                sistema.inscriptos(otraCarrera, otraMateria) =
                old(sistema).inscriptos(c, otraMateria))}
    asegura {
      (∀ otraCarrera, otraMateria: string)(
           otraCarrera ∈ sistema.carreras ∧_L otraMateria ∈ sistema.materias(otraCarrera) ∧_L
           sistema.sonMismaMateria(carrera, nombreMateria, otraCarrera, otraMateria) ⟶_L
                sistema.inscriptos(otraCarrera, otraMateria) =
                old(sistema).inscriptos(c, otraMateria)) ∪ ⟨ estudiante ⟩}
    asegura {(∀ e: string)(e ∈ sistema.estudiantes ∧ e ≠ estudiante ⟶_L
                                        sistema.inscripciones(e) = old(sistema).inscripciones(e))}
    asegura {sistema.inscripciones(estudiante) = old(sistema).inscripciones(estudiante) + 1}
    asegura que el resto de observadores no se modifican

proc inscriptos(in sistema: SistemaSIU, in carrera: string, in nombreMateria: string): ℕ
    requiere {carrera ∈ sistema.carreras}
    requiere {nombreMateria ∈ sistema.materias(carrera)}
    asegura {res = |sistema.inscriptos(carrera, nombreMateria)|}

proc agregarDocente(inout sistema: SistemaSIU, in cargo: CargoDocente,
                    in carrera: string, in nombreMateria: string)
    requiere {carrera ∈ sistema.carreras}
    requiere {nombreMateria ∈ sistema.materias(carrera)}
    asegura {(∀ otraCarrera, otraMateria: string)(
                otraCarrera ∈ sistema.carreras ∧_L otraMateria ∈ sistema.materias(otraCarrera) ∧_L
                sistema.sonMismaMateria(carrera, nombreMateria, otraCarrera, otraMateria) ⟶_L
                (∀ otroCargo: CargoDocente)(otroCargo ≠ cargo →
                          sistema.docentes(otroCargo, otraCarrera, otraMateria) =
                          old(sistema).docentes(otroCargo, otraCarrera, otraMateria)))}
    asegura {(∀ otraCarrera, otraMateria: string)(
                otraCarrera ∈ sistema.carreras ∧_L otraMateria ∈ sistema.materias(otraCarrera) ∧_L
                sistema.sonMismaMateria(carrera, nombreMateria, otraCarrera, otraMateria) ⟶_L
                          sistema.docentes(cargo, otraCarrera, otraMateria) =
                          old(sistema).docentes(cargo, otraCarrera, otraMateria) + 1)}
    asegura {(∀ otraCarrera, otraMateria: string)(
                otraCarrera ∈ sistema.carreras ∧_L otraMateria ∈ sistema.materias(otraCarrera) ∧_L
                ¬ sistema.sonMismaMateria(carrera, nombreMateria, otraCarrera, otraMateria) ⟶_L
                (∀ otroCargo: CargoDocente)(
                          sistema.docentes(otroCargo, otraCarrera, otraMateria) =
                          old(sistema).docentes(otroCargo, otraCarrera, otraMateria)))}
    asegura que el resto de observadores no se modifican

proc plantelDocente(in sistema: SistemaSIU, in carrera: string, in nombreMateria: string): seq<ℕ>
    requiere {carrera ∈ sistema.carreras}
    requiere {nombreMateria ∈ sistema.materias(carrera)}
    asegura {
      res = [
```

```
            sistema.docentes(PROFE, carrera, nombreMateria),
            sistema.docentes(JTP, carrera, nombreMateria),
            sistema.docentes(AY1, carrera, nombreMateria),
            sistema.docentes(AY2, carrera, nombreMateria)
        ]
    }
  proc excedeCupo?(in sistema: SistemaSIU, in carrera: string, in nombreMateria: string): bool
    requiere {carrera ∈ sistema.carreras}
    requiere {nombreMateria ∈ sistema.materias(carrera)}
    asegura {|res.inscriptos(carrera, nombreMateria)| ≤ cupo(sistema, carrera, nombreMateria)}
  proc carreras(in sistema: SistemaSIU): seq<string>
    asegura {igualdadSeqConj(res, sistema.carreras)}
    asegura {ordenadoEstricto(res)}
  proc materias(in sistema: SistemaSIU, in carrera: string): seq<string>
    requiere {carrera ∈ sistema.carreras}
    asegura {igualdadSeqConj(res, sistema.materias(carrera))}
    asegura {ordenadoEstricto(res)}
  proc materiasInscriptas(in sistema: SistemaSIU, in estudiante: string): ℕ
    requiere {estudiante ∈ sistema.estudiantes}
    asegura {res = sistema.inscripciones(estudiante)}
  proc cerrarMateria(inout sistema: SistemaSIU, in carrera: string, in nombreMateria: string)
    requiere {carrera ∈ sistema.carreras}
    requiere {nombreMateria ∈ sistema.materias(carrera)}
    asegura {
      (∀ otraCarrera, otraMateria: string)(
            old(sistema).sonMismaMateria(carrera, nombreMateria, otraCarrera, otraMateria) ⟶_L
                  sistema.materias(carrera) = old(sistema).materias(otraCarrera) - ⟨ otraMateria ⟩})
    asegura {
      (∀ estudiante, otraCarrera, otraMateria: string)(
            otraCarrera ∈ sistema.carreras ∧_L otraMateria ∈ old(sistema).materias(carrera) ∧_L
            old(sistema).sonMismaMateria(carrera, nombreMateria, otraCarrera, otraMateria)∧
            estudiante ∈ old(sistema).inscriptos(otraCarrera, otraMateria) ⟶_L
                  sistema.inscripciones(estudiante) = old(sistema).inscripciones(estudiante) - 1
      )
    }
    asegura {
      (∀ estudiante, otraCarrera, otraMateria: string)(
            otraCarrera ∈ sistema.carreras ∧_L otraMateria ∈ old(sistema).materias(carrera) ∧_L
            old(sistema).sonMismaMateria(carrera, nombreMateria, otraCarrera, otraMateria)∧
            estudiante ∉ old(sistema).inscriptos(otraCarrera, otraMateria) ⟶_L
                  sistema.inscripciones(estudiante) = old(sistema).inscripciones(estudiante)
      )
    }
    asegura que el resto de observadores no se modifican
}
```