Guía 3

Algoritmos y Estructuras de Datos II, DC, UBA.

Segundo cuatrimestre de 2020

Índice

1. Modalidad de trabajo **2**
- 1.1. Clases . . . 2
- 1.2. Vías de comunicación y espacios de consulta . . . 2
- 1.3. Correlativas . . . 2
- 1.4. Guías de ejercitación y criterios de aprobación . . . 2
- 1.5. Cronograma tentativo de la materia . . . 3

2. Ejercicios seleccionados **5**
- 2.1. Quiz sobre las clases teóricas . . . 5
  - 2.1.1. Clase T06: Tries . . . 5
  - 2.1.2. Clase T07: Hashing . . . 5
  - 2.1.3. Clase T08: Colas de prioridad . . . 6
- 2.2. Elección de estructuras . . . 6

3. Ejercicios obligatorios del laboratorio **10**
- 3.1. TP 3 — fecha de entrega: domingo 15 de noviembre . . . 10
- 3.2. Taller: Conjunto sobre Árbol Binario de Búsqueda . . . 15

1. Modalidad de trabajo

1.1. Clases

La materia se divide en clases teóricas, clases prácticas y clases de laboratorio. Como regla general, el material expositivo de las clases será grabado en diferido (no en vivo) y estará disponible en video en la sección **Clases** del sitio de la materia en el Campus[1]. Cualquier excepción a esta regla se informará con anticipación.

1.2. Vías de comunicación y espacios de consulta

Toda la información importante sobre la materia se comunicará a través de la lista de correo `algo2-alu@dc.uba.ar` que pueden leer y escribir todes les docentes y alumnes de la materia. Las consultas a docentes acerca de cuestiones administrativas o de índole personal se pueden hacer a través de la lista `algo2-doc@dc.uba.ar` que pueden leer todes les docentes de la materia, y a la que pueden escribir todes les alumnes.
Las consultas acerca de temas de la materia se responderán a través de los siguientes medios:

1. Para comunicación *asincrónica* (estilo foro) en la sección **Foro de Consultas** del sitio de la materia en el Campus se pueden dejar por escrito consultas que les docentes podrán leer y responder. Aconsejamos revisar el foro antes de formular una pregunta para ver si ya fue respondida con anterioridad.

2. Para comunicación *sincrónica* (estilo chat/videoconferencia), pondremos disponibles canales de comunicación (probablemente a través de Zoom). En general las consultas tendrán lugar en las siguientes franjas horarias:

   - **Laboratorio:** miércoles de 09:30 a 17:30 (turno mañana) y de 17:30 a 20:30 (turno noche).
   - **Práctica:** viernes de 11:00 a 14:00 (turno mañana) y de 17:00 a 20:00 (turno noche).

Tengan en cuenta que les docentes disponen de tiempo limitado para responder consultas asincrónicas, y que elaborar una respuesta por escrito puede consumir mucho tiempo y prestarse a malentendidos. Por eso aconsejamos **limitar el uso de la comunicación asincrónica** a dudas administrativas, y **aprovechar los espacios de consulta sincrónica** para resolver dudas de la materia.

1.3. Correlativas

Para poder cursar la materia se requiere tener aprobada la cursada (no así el final) de Algoritmos y Estructuras de Datos I antes del comienzo del cuatrimestre.

1.4. Guías de ejercitación y criterios de aprobación

La materia se dividirá en cuatro bloques. Cada bloque tendrá una **guía** como esta, que se encontrará disponible en la sección **Guías de ejercitación** del sitio de la materia en el Campus. Cada guía incluye:

1. **Ejercicios seleccionados (no se entregan).** Preguntas teóricas y ejercicios prácticos destinados a validar que hayan entendido el material presentado en las clases teóricas. Estas preguntas discutirán en los horarios de consulta y eventualmente se publicarán soluciones de algunas o varias de ellas.

2. **Ejercicios obligatorios del laboratorio (TPs y talleres)**. Los TPs son grupales (en grupos de 4 integrantes). Los talleres son individuales. Cada instancia se debe entregar antes de la fecha de finalización del bloque, y podrá ser aprobada o devuelta para incorporar correcciones. Se detallan en la Sección 3 (margen verde).

3. **Parciales.** Al final de cada bloque se publicará un parcial (individual). Tendrán tres días para resolver cada parcial. Cada ejercicio de cada parcial podrá ser aprobada o desaprobado y se recupera individualmente al final del cuatrimestre.

La forma de entrega de los ejercicios individuales y trabajos prácticos será informada oportunamente. Para aprobar los prácticos de la materia será necesario aprobar **todas** las instancias de evaluación (tras las instancias de recuperación correspondientes).

En caso de que la Facultad vuelva a funcionar de manera presencial, los mecanismos de evaluación podrían sufrir modificaciones.

[1] `https://campus.exactas.uba.ar/course/view.php?id=1846`

1.5. Cronograma tentativo de la materia

Se incluye un cronograma tentativo de la materia. Se publica con fines orientativos, pero está sujeto a todo tipo de cambios. Los colores corresponden a teórica (T), laboratorio (L) y práctica (P). Las clases están numeradas: por ejemplo el lunes 13/04 está programado que se publiquen las primeras dos clases teóricas (T01 y T02).

1. **Semana 1**
   - Lun 31/08 — Presentación de la materia. T01, T02: Especificación
   - Mié 02/09 — L01: Uso de clases (09:30-12:30 y 17:30-20:30)
   - Vie 04/09 — Consultas T01, T02 (11:00-12:00 y 17:00-18:00)
2. **Semana 2**
   - Lun 07/09 — P01: TADs y recursión (básico) ..... **Inicio del bloque 1**
   - Mié 09/09 — L02: Clases en C++ (básico), testing — **Presentación TP1** (09:30-12:30 y 17:30-20:30)
   - Vie 11/09 — Consultas P01 (11:00-14:00 y 17:00-20:00)
3. **Semana 3**
   - Lun 14/09 — T03: Complejidad — P02: TADs y recursión (avanzado)
   - Mié 16/09 — L03: Clases en C++ (09:30-12:30 y 17:30-20:30)
   - Vie 18/09 — Consultas T03 (11:00-12:00 y 17:00-18:00) — Consultas P02 (12:00-14:00 y 18:00-20:00)
4. **Semana 4**
   - Lun 21/09 — P03: TAD en dos niveles — T04: Diseño
   - Mié 23/09 — L04: Memoria dinámica (09:30-12:30 y 17:30-20:30)
   - Vie 25/09 — Consultas T04 (11:00-12:00 y 17:00-18:00) — Consultas P03 (12:00-14:00 y 18:00-20:00)
   - Dom 27/09 — **Entrega TP 1** – **Publicación Parcial 1**
5. **Semana 5**
   - Lun 28/09 — Consultas Parcial 1 (11:00-14:00 y 17:00-20:00)
   - Mar 29/09 — **Entrega Parcial 1** — P04: Notación "O", complejidad ..... **Inicio del bloque 2**
   - Mié 30/09 — L05: Listas enlazadas — **Presentación TP2** (09:30-12:30 y 17:30-20:30)
   - Vie 02/10 — Consultas P04 (11:00-14:00 y 17:00-20:00)
6. **Semana 6**
   - Lun 05/10 — T05: Diccionarios sobre ABBs y AVLs, P05: Rep y Abs
   - Mié 07/10 — L06: Templates y algoritmos genéricos (09:30-12:30 y 17:30-20:30) — **Devolución TP 1**
   - Vie 09/10 — Consultas T05 (11:00-12:00 y 17:00-18:00)
7. **Semana 7**
   - Lun 12/10 — **Feriado: Día del Respeto a la Diversidad Cultural**
   - Mié 14/10 — Consultas (09:30-12:30 y 17:30-20:30)
   - Vie 16/10 — Consultas pre-parcial (11:00-14:00 y 17:00-20:00)
   - Dom 18/10 — **Entrega TP 2** — **Publicación Parcial 2**
8. **Semana 8**
   - Lun 19/10 — Consultas Parcial 2 (11:00-14:00 y 17:00-20:00)
   - Mar 20/10 — **Entrega Parcial 2** — T06: Tries ..... **Inicio del bloque 3**
   - Mié 21/10 — L07: ABBs — **Presentación TP3** (09:30-12:30 y 17:30-20:30)
   - Vie 23/10 — Consultas T06 (11:00-12:00 y 17:00-18:00)

9. **Semana 9**
   - Lun 26/10 — T07: Hashing 1 y 2 — P06: Interfaces y módulos
   - Mié 28/10 — Consultas (09:30-12:30 y 17:30-20:30) — **Devolución TP 2**
   - Vie 30/10 — Consultas T07 (11:00-12:00 y 17:00-18:00) — Consultas P06 (12:00-14:00 y 18:00-20:00)
10. **Semana 10**
   - Lun 02/11 — T08: Colas de prioridad — P07: Elección de estructuras
   - Mié 04/11 — L08: Tries (09:30-12:30 y 17:30-20:30)
   - Vie 06/11 — Consultas T08 (11:00-12:00 y 17:00-18:00) — Consultas P07 (12:00-14:00 y 18:00-20:00)
11. **Semana 11**
   - Lun 09/11 — T09: Sorting básico — P08: Elección de estructuras avanzadas
   - Mié 11/11 — L09: Heaps — **Presentación TP4** (09:30-12:30 y 17:30-20:30)
   - Vie 13/11 — Consultas P08 y pre-parcial (11:00-14:00 y 17:00-20:00)
   - Dom 15/11 — **Entrega TP 3** — **Publicación Parcial 3**
12. **Semana 12**
   - Lun 16/11 — Consultas Parcial 3 (11:00-14:00 y 17:00-20:00)
   - Mar 17/11 — **Entrega Parcial 3** — P09: Sorting ..... **Inicio del bloque 4**
   - Mié 18/11 — Consultas (09:30-12:30 y 17:30-20:30)
   - Vie 20/11 — Consultas T09 (11:00-12:00 y 17:00-18:00) — Consultas P09 (12:00-14:00 y 18:00-20:00)
13. **Semana 13**
   - Lun 23/11 — **Feriado: Día de la Soberanía Nacional** — T10: Divide and conquer — P10: D&C
   - Mié 25/11 — L10: Desarrollo de iteradores (09:30-12:30 y 17:30-20:30) — **Devolución TP 3** — **Posible entrega TP 4 (fecha 1/3)**
   - Vie 27/11 — Consultas T10 (11:00-12:00 y 17:00-18:00) — Consultas P10 (12:00-14:00 y 18:00-20:00)
14. **Semana 14**
   - Lun 30/11 — Consultas pre-parcial (11:00-14:00 y 17:00-20:00)
   - Mié 02/12 — Consultas (09:30-12:30 y 17:30-20:30) — **Posible entrega TP 4 (fecha 2/3)**
   - Jue 03/12 — **Publicación Parcial 4**
   - Vie 04/12 — Consultas Parcial 4 (11:00-14:00 y 17:00-20:00)
   - Sáb 05/12 — **Entrega Parcial 4**
15. **Semana 15**
   - Lun 07/12 — **Feriado con fines turísticos** — T11: Sorting y diccionarios en memoria externa — T12: Splay trees y skip lists
   - Mié 09/12 — L12: Sorting (ex taller de sorting) (09:30-12:30 y 17:30-20:30) — **Límite entrega TP 4 (fecha 3/3)**
   - Vie 11/12 — Consultas T11, T12 (11:00-12:00 y 17:00-18:00) — **Límite entrega talleres**
16. **Semana 16**
   - Lun 14/12 — Consultas pre-recuperatorio (11:00-14:00 y 17:00-20:00)
   - Vie 18/12 — **Presentación Recuperatorio**[2]
17. **Semana 17**
   - Lun 21/12 — **Entrega Recuperatorio**

[2] Cada ejercicio de cada parcial se recupera de manera independiente.

2. Ejercicios seleccionados

Los **ejercicios seleccionados** son un conjunto de ejercicios *mínimos* destinados a que cada estudiante pueda realizar una autoevaluación sobre su progreso en el dominio de los contenidos que se presentan en la materia, tanto desde el aspecto conceptual (entendimiento de los temas) como en el aspecto procedimental (capacidad de aplicar los conocimientos para resolver problemas prácticos). Deberían servir como disparadores para repasar partes de las clases que no hayan quedado claras, referirse a bibliografía complementaria y formular consultas.

Corrección: se brindarán resoluciones de los ejercicios prácticos, y se comentará sobre estas resoluciones en las clases de consulta. La idea es que no acudan a la resolución brindada por la cátedra sin haber intentado resolverlos por su cuenta.

Alentamos que los piensen individualmente, los discutan con sus compañeres y los consulten con les docentes. **Advertencia:** la resolución de los ejercicios seleccionados en esta guía no sustituye la resolución de prácticas (guías de ejercicios) publicadas en el sitio de la materia en el Campus.

2.1. Quiz sobre las clases teóricas

2.1.1. Clase T06: Tries

- Para cada $h \in \mathbb{N}$, considerar un trie que incluye todas las palabras de longitud h en el alfabeto $\{a, b\}$. Por ejemplo si $h = 3$, las palabras son *aaa*, *aab*, *aba*, *abb*, *baa*, *bab*, *bba*, *bbb*. Notar que h coincide con la altura del trie. Sea n además la cantidad de nodos del trie. ¿Cuáles de las siguientes afirmaciones valen?

$$n \in O(1) \qquad n \in O(h) \qquad n \in O(h^2) \qquad n \in O(2^h)$$

- Considerar un trie similar al anterior, pero que sólo contiene aquellas palabras que constan de una secuencia de *a*s (posiblemente vacía) seguida de una secuencia de *b*s (posiblemente vacía). Por ejemplo si $h = 4$, las palabras son *aaaa*, *aaab*, *aabb*, *abbb*, *bbbb*. Igual que antes, h coincide con la altura del trie, y notamos n a la cantidad de nodos del trie. ¿Cuáles de las siguientes afirmaciones valen?

$$n \in O(1) \qquad n \in O(h) \qquad n \in O(h^2) \qquad n \in O(2^h)$$

- Un número natural n se puede escribir en una base de numeración $b \geq 2$ usando $\lfloor \log_b n \rfloor + 1$ dígitos. Por ejemplo, el número 12 se escribe en binario usando 4 dígitos: $12 = (1010)_2$, y su logaritmo en base 2 es $\log_2(12) = 3{,}584...$. Se tiene un conjunto de M números acotados por n. ¿Cuál es la complejidad temporal en peor caso de las operaciones de búsqueda, inserción y borrado si se representa el conjunto con un trie, escribiendo los números en base b? ¿Cómo se compara con las complejidades que tendrían las operaciones si se representara el conjunto con un AVL?

2.1.2. Clase T07: Hashing

- Un usuario malicioso desea insertar n elementos en un conjunto representado sobre una tabla de hash, de tal modo que el conjunto se comporte de la manera más ineficiente posible[3]. Recordemos que una función de hash $h : U \to \{1, \ldots, M\}$ recibe un elemento $x \in U$ y lo asocia a un índice de la tabla. En términos de la función de hash, ¿qué debería encontrar el usuario malicioso para poder realizar el ataque? ¿Cómo se podría diseñar una función de hash que no sea vulnerable a este tipo de ataques?

- Una tabla de hash con direccionamiento cerrado tiene M *buckets* y en ella se han insertado n elementos en total. Suponiendo que la función de hash distribuye uniformemente la entrada entre los M buckets, ¿cuáles serían las complejidades temporales en caso promedio para las operaciones de búsqueda, inserción y borrado, y cuál sería el número de colisiones esperadas por bucket en cada caso?:

  1. $M \in \Theta(1)$, por ejemplo si $M = 100$.
  2. $M \in \Theta(n)$, por ejemplo si se verifica $\frac{n}{2} \leq M \leq 2n$.
  3. $M \in \Theta(n^2)$

  Comparar cada caso con el costo de búsqueda, inserción y borrado en una lista enlazada sin repetidos.

[3]Haciendo esto puede, por ejemplo, saturar un servidor con pedidos que requieren mucho tiempo de cómputo para que otros usuarios no puedan utilizarlo, lo que se conoce como un ataque de denegación de servicio.

- Si en una tabla como la de arriba se siguen insertando elementos pero no se redimensiona la tabla, el valor de n aumenta mientras que M queda fijo. ¿Qué impacto tendrá esto sobre las complejidades de las operaciones?

2.1.3. Clase T08: Colas de prioridad

- Comparar las complejidades en peor caso de encolar un elemento, conocer cuál es el elemento de mayor prioridad, y desencolar el elemento de mayor prioridad en las siguientes estructuras:
  1. Lista sin repetidos.
  2. Lista sin repetidos, ordenada de mayor a menor prioridad.
  3. Heap.
- Se tienen dos colas de prioridad representadas sobre heap. La cola q_1 tiene n elementos y la cola q_2 tiene m elementos. Se quiere armar una nueva cola de prioridad que incorpore los $n + m$ elementos de las colas q_1 y q_2. Considerar estos dos algoritmos:
  - Algoritmo A: mientras q_1 no esté vacía, desencolar un elemento de q_1 y encolarlo en q_2.
  - Algoritmo B: juntar los elementos de q_1 y q_2 en un arreglo y aplicar el algoritmo *heapify* de Floyd.

  ¿Cuál es la complejidad en peor caso de cada algoritmo? ¿Cuándo conviene usar el algoritmo A y cuándo el B?

2.2. Elección de estructuras

Ejercicio 1: Matriz infinita

Una matriz finita posee las siguientes operaciones:

- *Crear*, con la cantidad de filas y columnas que albergará la matriz.
- *Definir*, que permite definir el valor para una posición válida.
- *#Filas*, que retorna la cantidad de filas de la matriz.
- *#Columnas*, que retorna la cantidad de columnas de la matriz.
- *Obtener*, que devuelve el valor de una posición válida de la matriz (si nunca se definió la matriz en la posición solicitada devuelve cero).
- *SumarMatrices*, que permite sumar dos matrices de iguales dimensiones.

Diseñe un módulo para el TAD Matriz finita de modo tal que dadas dos matrices finitas A y B,

∗ *Definir* y *Obtener* aplicadas a A se realicen cada una en $\Theta(n)$ en peor caso, y

∗ *SumarMatrices* aplicada a A y B se realice en $\Theta(n + m)$ en peor caso,

donde n y m son la cantidad de elementos no nulos de A y B, respectivamente.

Ejercicio 2: Sistema de estadísticas

Se desea diseñar un sistema de estadísticas para la cantidad de personas que ingresan a un banco. Al final del día, un empleado del banco ingresa en el sistema el total de ingresantes para ese día. Se desea saber, en cualquier intervalo de días, la cantidad total de personas que ingresaron al banco. La siguiente es una especificación del problema.

TAD IngresosAlBanco

observadores básicos

totDias : iab $\longrightarrow$ nat

cantPersonas : iab i × nat d × nat h $\longrightarrow$ nat $\{1 \leq d \wedge d \leq h \wedge h \leq totDias(i)\}$

generadores

```
    Comenzar : ⟶ iab
    TerminaDia : iab × nat ⟶ iab
  axiomas ...
    totDias(Comenzar) ≡ 0
    totDias(TerminaDia(i, n)) ≡ 1 + totDias(i)
    cantPersonas(TerminaDia(i, n),d, h) ≡ if totDias(i) < h then n else 0 fi + if totDias(i) < d then
                                            0
                                          else
                                            cantPersonas(i, d, mín(h,totDias(i)))
                                          fi
Fin TAD
```

1. Dar una estructura de representación que permita que la función *cantPersonas* tome $O(1)$.
2. Calcular cuánta memoria usa la estructura, en función de la cantidad de días que pasaron n.
3. Si el cálculo del punto anterior fue una función que no es $O(n)$, piense otra estructura que permita resolver el problema utilizando $O(n)$ memoria.

Ejercicio 3: Ranking

Se desea diseñar un sistema para manejar el ranking de posiciones de un torneo deportivo. En el torneo hay un conjunto fijo de equipos que juegan partidos (posiblemente más de un partido entre cada pareja de equipos) y el ganador de cada partido consigue un punto. Para el ranking, se decidió que entre los equipos con igual cantidad de puntos no se desempata, sino que todos reciben la mejor posición posible para ese puntaje. Por ejemplo, si los puntajes son: A: 5 puntos, B: 5 puntos, C: 4 puntos, D: 3 puntos, E: 3 puntos, las posiciones son: A: 1ro, B: 1ro, C: 3ro, D: 4to, E: 4to.
El siguiente TAD es una especificación para este problema.

```
TAD Equipo es Nat.
TAD Torneo
  observadores básicos
    equipos : torneo ⟶ conj(equipo)
    puntos  : torneo t × equipo e ⟶ nat                                   {e ∈ equipos(t)}
  generadores
    nuevoTorneo : conj(equipo) c ⟶ torneo                                 {¬∅?(c)}
    regPartido  : torneo t × equipo g × equipo p ⟶ torneo   {g ∈ equipos(t) ∧ p ∈ equipos(t) ∧ g ≠ p}
  otras operaciones
    pos         : torneo t × equipo e ⟶ nat                               {e ∈ equipos(t)}
    #masPuntos  : torneo t × conj(equipo) c × nat ⟶ nat                   {c ⊆ equipos(t)}
  axiomas
    equipos(nuevoTorneo(c))      ≡ c
    equipos(regPartido(t, g, p)) ≡ equipos(t)
    puntos(nuevoTorneo(c),e)     ≡ 0
    puntos(regPartido(t, g, p),e) ≡ puntos(t, e) + if e = g then 1 else 0 fi
    pos(t, e)                    ≡ 1+#masPuntos(t,equipos(t),puntos(t, e))
    #masPuntos(t, c, p)          ≡ if ∅?(c) then
                                     0
                                   else
                                     if puntos(t,dameUno(c)) > p then 1 else 0 fi +
                                     #masPuntos(t,sinUno(c),p)
                                   fi
Fin TAD
```

Se desea diseñar el sistema propuesto, teniendo en cuenta que las operaciones *puntos*, *regPartido* y *pos* deben realizarse en $O(\log n)$, donde n es la cantidad de equipos registrados.

a) Describir la estructura a utilizar.

b) Escribir un pseudocódigo del algoritmo para las operaciones con requerimientos de complejidad.

Ejercicio 4 (de parcial): Sistema de transporte

La empresa de optimización del transporte AED2 Company desea contribuir a que los alumnos de todas partes viajen más rápido hacia diversas universidades de una determinada ciudad. Para eso desean contar con un registro de universidades y de líneas de colectivos. En cualquier momento se pueden agregar o dar de baja tanto líneas como universidades. El objetivo principal es que se pueda consultar, dada una universidad, qué líneas llegan a ella, y dada una línea, a qué universidades llega. El mapa de la ciudad de subdividió en celdas, entonces, podríamos decir que cada universidad pertenece a una celda y cada línea visita un conjunto de celdas. Se puede asumir que la ciudad tiene límites bien definidos y no cambian con el tiempo (i.e., la cantidad de celdas del mapa no es una variable del problema).

TAD CELDA **es** <NAT, NAT>

TAD AED2COMPANY

observadores básicos

Universidades : AED2Company $\longrightarrow$ conj(Universidad)	
Líneas : AED2Company $\longrightarrow$ conj(Línea)	
DirecciónUniversidad : AED2Company a $\times$ Universidad u $\longrightarrow$ Celda	$\{u \in$ Universidades(a)$\}$
CeldasQueVisita : AED2Company a $\times$ Línea l $\longrightarrow$ conj(Celda)	$\{l \in$ Líneas(a)$\}$

generadores

Iniciar : $\longrightarrow$ AED2Company	
AgregarUniversidad : AED2Company a $\times$ Universidad u $\times$ Celda d $\longrightarrow$ AED2Company	$\{u \notin$ Universidades(u)$\}$
AgregarLínea : AED2Company a $\times$ Línea l $\times$ conj(Celda) c $\longrightarrow$ AED2Company	$\{c \neq \emptyset \wedge l \notin$ Líneas(u)$\}$
EliminarUniversidad : AED2Company a $\times$ Universidad u $\longrightarrow$ AED2Company	$\{u \in$ Universidades(u)$\}$
EliminarLínea : AED2Company a $\times$ Línea l $\longrightarrow$ AED2Company	$\{l \in$ Líneas(t)$\}$

otras operaciones

LíneasPorUniversidad : AED2Company a $\times$ Universidad u $\longrightarrow$ conj(Línea)	$\{u \in$ Universidades(a)$\}$
UniversidadesPorLínea : AED2Company a $\times$ Línea l $\longrightarrow$ conj(Universidad)	$\{l \in$ Líneas(a)$\}$

axiomas

...

Fin TAD

Las complejidades que tiene que cumplir nuestro sistema son:

- DarDeAlta / DarDeBaja Universidad: $O(log\ U)$
- DarDeAlta / DarDeBaja Línea: $O(log\ l)$
- ConsultarLíneasPorUniversidad: $O(log\ U)$
- ConsultarUniversidadesPorLínea: $O(log\ l + M_l)$

Donde U es la cantidad de universidades, l es el **número de línea** dado y M_l la cantidad de universidades que contiene el recorrido de la línea l.

1. Escriba la estructura de representación del módulo AED2 COMPANY explicando detalladamente qué información se guarda en cada parte de la misma y las relaciones entre las partes. Describa también las estructuras de datos subyacentes.
2. Escriba los algoritmos DarDeAltaLínea y ConsultarUniversidadesPorLínea. Justifique el cumplimiento de los órdenes solicitados. Para cada una de las demás funciones, descríbalas en castellano, justificando por qué se cumple el orden de complejidad pedido.

Ejercicio 5 (de parcial): Sistema de multas

El sistema MULT.AR se encarga de registrar las multas de los vehículos a lo largo y ancho del país. Cada *localidad* posee un conjunto de *cámaras* y tiene un conjunto de *vehículos* registrados (aunque los vehículos pueden circular libremente por todo el país). Cuando un vehículo sobrepasa la velocidad máxima y es registrado por una cámara, se emite una *multa* al infractor. Por lo tanto para registrar una multa se necesita el código identificador de la cámara, la patente del vehículo que cometió la infracción y el monto de la misma. En cualquier momento se pueden abonar las multas de un vehículo, pero en ese momento se deben abonar todas las multas existentes del vehículo. Esta acción elimina todos los registros del sistema asociados a esas multas.

TAD MULT.AR

observadores básicos

localidades	: mar	$\longrightarrow$	conj(loc)		
camarasDe	: mar m × loc ℓ	$\longrightarrow$	conj(camara)		$\{\ell \in \text{localidades}(m)\}$
vehiculosDe	: mar m × loc ℓ	$\longrightarrow$	conj(vehiculo)		$\{\ell \in \text{localidades}(m)\}$
multasPorV	: mar m × vehiculo v	$\longrightarrow$	multiconj(multa)		$\{v \in \text{vehiculos}(m)\}$

generadores

iniciar	: dicc(loc × camara)	$\longrightarrow$	mar	
registrarVehiculo	: mar m × loc ℓ × vehiculo v	$\longrightarrow$	mar	$\{\ell \in \text{localidades}(m) \land v \notin \text{vehiculos}(m)\}$
multar	: mar m × vehiculo v × camara c × nat	$\longrightarrow$	mar	$\{v \in \text{vehiculos}(m) \land c \in \text{camaras}(m)\}$
abonar	: mar m × vehiculo v	$\longrightarrow$	mar	$\{v \in \text{vehiculos}(m)\}$

otras operaciones

camaras	: mar	$\longrightarrow$	conj(camara)	{ }
vehiculos	: mar	$\longrightarrow$	conj(vehiculo)	{ }
multasPorLoc	: mar m × loc ℓ	$\longrightarrow$	multiconj(multa)	$\{\ell \in \text{localidades}(m)\}$

axiomas

...

Fin TAD

Las *localidades* y los *vehículos* se representan con strings (acotados en el caso de los vehículos, por ser las placas de las patentes), y las cámaras se representan con números naturales. Una *multa* se representa con una tupla *<v: vehiculo, c: camara, monto: nat>*.
Se debe realizar un diseño que cumpla con los siguientes órdenes de complejidad en el peor caso, siendo ℓ el nombre de la localidad, n la cantidad de cámaras del sistema y m la cantidad de multas del vehículo en cuestión:

- Dada una localidad ℓ, obtener los vehículos registrados, las cámaras y las multas (incluyendo las de sus vehículos y las de sus cámaras), cada operación en $O(|\ell|)$.
- Dado un vehículo, obtener su localidad y sus multas, ambos en $O(1)$.
- Abonar (eliminando del sistema) las multas de un vehículo en $O(m)$.
- Dada una cámara, un vehículo y un monto, registrar una nueva multa en $O(\log n)$.

1. Escriba la estructura de representación del módulo explicando detalladamente qué información se guarda en cada parte y las relaciones entre las partes. Describa también las estructuras de datos subyacentes.
2. Escriba el algoritmo para abonar (eliminando del sistema) las multas de un vehículo y justifique el cumplimiento de la complejidad solicitada. Para las demás funciones, justifique en castellano por qué se cumple la complejidad pedida.

3. Ejercicios obligatorios del laboratorio

IMPORTANTE: Los ejercicios de esta sección se dividen en TPs (grupales, grupos de 4 integrantes) y talleres (individuales), y serán calificados. Cada instancia podrá estar aprobada, o será devuelta para incorporar correcciones. Para la resolución de los TPs se espera que trabajen grupalmente y consulten todas sus dudas. No está permitido compartir soluciones detalladas entre grupos de trabajo distintos. En el caso de los talleres, si bien se permite que discutan ideas grupalmente, cada entrega debe ser individual y todo el código entregado debe ser de producción propia.

3.1. TP 3 — fecha de entrega: domingo 15 de noviembre

El objetivo de este TP es diseñar módulos destinados a implementar el juego Sokoban que se describió en los TPs anteriores. En la siguiente sección pueden encontrar la **especificación formal** del juego provista por la cátedra. Notar que el único cambio con respecto al enunciado del TP 2 es que el repositorio de niveles es ahora una **secuencia**. Cuando el jugador gana un nivel, pasa al nivel siguiente en la secuencia. El diseño propuesto debe cumplir con las siguientes **complejidades temporales en peor caso**, donde: B representa el número de bombas arrojadas desde el inicio del nivel, C representa el número de cajas del nivel actual, P representa el número de paredes que había originalmente en el nivel actual (antes de arrojar bombas) y D representa el número de depósitos del nivel actual

1. Determinar la **posición actual** del jugador en el mapa: $O(1)$.
2. Determinar el **número de bombas** que posee el jugador: $O(1)$.
3. Determinar si **hay una caja** en una posición (x, y) arbitraria: $O(C)$.
4. Determinar si **hay una pared** en una posición (x, y) arbitraria: $O(B + \log P)$.
5. Determinar si **hay un depósito** en una posición (x, y) arbitraria: $O(\log D)$.
6. Determinar si el jugador **se puede mover** hacia cualquiera de las cuatro direcciones: $O(B + C + \log P)$.
7. **Mover el jugador** hacia cualquiera de las cuatro direcciones:
   - Si el movimiento **no** provoca que el jugador gane el nivel, debe ser $O(B + C + \log P + \log D)$.
   - Si el movimiento provoca que el jugador gane el nivel, no hay restricciones de complejidad para esta operación.
8. **Arrojar una bomba**: $O(1)$.
9. **Deshacer el último movimiento**: $O(1)$.

Asimismo, se pide que la representación de una instancia del Sokoban en términos de **complejidad espacial en peor caso** sea $O(B + C + P + D + M)$, donde M es el número de movimientos realizados desde el inicio del nivel. En particular, esta restricción impide que utilicen una matriz para representar directamente el mapa. Por ejemplo, si el juego tiene sólo una pared en la coordenada $(0, 0)$ y otra pared en la coordenada $(0, n)$, sería inadmisible que esto consuma $\Omega(n)$ espacio en memoria cuando el valor de n es muy grande.

Todos los módulos diseñados deben contar con las siguientes partes. Se debe diseñar el módulo principal (`Juego`) y todos los módulos auxiliares. La única excepción son los módulos disponibles en el Apunte de Módulos Básicos, que se pueden utilizar sin diseñarlos: lista enlazada, pila, cola, vector, diccionario lineal, conjunto lineal y conjunto acotado de naturales.

1. **Interfaz.**
   a) *Tipo abstracto ("se explica con ...").* Género (TAD) que sirve para explicar las instancias del módulo, escrito en el lenguaje de especificación **formal** de la materia. Pueden utilizar la especificación que se incluye en el apéndice.
   b) *Signatura.* Listado de todas las funciones públicas que provee el módulo. La signatura se debe escribir con la notación de módulos de la materia, por ejemplo, apilar(**in/out** pila : Pila, **in** x : Elemento).

   c) *Contrato.* Precondición y postcondición de todas las funciones públicas. Las pre y postcondiciones de las funciones de la interfaz deben estar expresadas **formalmente** en lógica de primer orden.[4] Las pre y postcondiciones deben usar los TADs provistos en el apartado **Especificación de la cátedra** más abajo en este documento, y **no** a la especificación escrita por ustedes en el TP2.

   d) *Complejidades.* Complejidades de todas las funciones públicas, cuando corresponda.

   e) *Aspectos de* aliasing. De ser necesario, aclarar cuáles son los parámetros y resultados de los métodos que se pasan por copia y cuáles por referencia, y si hay *aliasing* entre algunas de las estructuras.

2. **Implementación.**

   a) *Representación ("se representa con ...").* Módulo con el que se representan las instancias del módulo actual.

   b) *Invariante de representación.* Puede estar expresado en lenguaje natural o formal.

   c) *Función de abstracción.* Puede estar expresada en lenguaje natural o formal. La función de abstracción debe referirse a los TADs provistos en el apartado **Especificación de la cátedra** más abajo en este documento, y **no** a la especificación escrita por ustedes en el TP2.

   d) *Algoritmos.* Pueden estar expresados en pseudocódigo, usando si es necesario la notación del lenguaje de módulos de la materia o notación tipo C++. Las pre y postcondiciones de las funciones auxiliares pueden estar expresadas en lenguaje natural (no es necesario que sean formales). Indicar de qué manera los algoritmos cumplen con el contrato declarado en la interfaz y con las complejidades pedidas. No se espera una demostración formal, pero sí una justificación adecuada.

3. **Servicios usados.** Módulos que se utilizan, detallando las complejidades, *aliasing* y otros aspectos que dichos módulos deben proveer para que el módulo actual pueda cumplir con su interfaz.

Sobre uso de tablas de hash.
Recomendamos **no** usar tablas de *hash* como parte de la solución a este TP. El motivo es que, si bien las tablas de *hash* proveen buenas garantías de complejidad *en caso promedio*—asumiendo ciertas propiedades sobre la función de *hash* y condiciones de buena distribución de la entrada—, no proveen en cambio buenas garantías de complejidad *en peor caso*. (En términos asintóticos, una tabla de *hash* se comporta en peor caso tan mal como una lista enlazada).

Sobre el uso de lenguaje natural y formal.
Las precondiciones y poscondiciones de las funciones auxiliares, el invariante y la función de abstracción pueden estar expresados en lenguaje natural. No es necesario que sean formales. Asimismo, los algoritmos pueden estar expresados en pseudocódigo. Por otro lado, está permitido que utilicen fórmulas en lógica de primer orden en algunos lugares puntuales, si consideran que mejora la presentación o subsana alguna ambigüedad. El objetivo del diseño es convencer al lector, y a ustedes mismos, de que la interfaz pública se puede implementar usando la representación propuesta y respetando las complejidades pedidas. Se recomienda aplicar el sentido común para priorizar la **claridad** y **legibilidad** antes que el rigor lógico por sí mismo. Por ejemplo:

Más claro	**Menos claro**
"Cada clave del diccionario D debe ser una lista sin elementos repetidos." ✓ "sinRepetidos?(claves(D))" ✓	"No puede haber repetidos." (¿En qué estructura?).
"Ordenar la lista A usando mergesort." ✓ "A.mergesort()" ✓	"Ordenar los elementos." (¿Qué elementos? ¿Cómo se ordenan?).
"Para cada tupla (x, y) en el conjunto C { x.apilar(y) n++ }" ✓	"Miro las tuplas del conjunto, apilo la segunda componente en la primera y voy incrementando un contador." (Ambiguo y difícil de entender).

[4] Si la implementación requiere usar funciones auxiliares, sus pre y postcondiciones pueden estar escritas en lenguaje natural, pero esto no forma parte de la interfaz.

Especificación de la cátedra

TAD Coord **es** Tupla$\langle x : \text{int}, y : \text{int}\rangle$

TAD Dirección

géneros dir

observadores básicos
ord : dir $\longrightarrow$ nat

generadores
norte : $\longrightarrow$ dir
este : $\longrightarrow$ dir
sur : $\longrightarrow$ dir
oeste : $\longrightarrow$ dir

otras operaciones
$\bullet \oplus \bullet$: coord $\times$ dir $\longrightarrow$ coord

axiomas
ord(norte) $\equiv$ 0
ord(este) $\equiv$ 1
ord(sur) $\equiv$ 2
ord(oeste) $\equiv$ 3

$c \oplus$ norte $\equiv \langle x\text{: } c.x, y\text{: } c.y + 1\rangle$
$c \oplus$ este $\equiv \langle x\text{: } c.x + 1, y\text{: } c.y\rangle$
$c \oplus$ sur $\equiv \langle x\text{: } c.x, y\text{: } c.y - 1\rangle$
$c \oplus$ oeste $\equiv \langle x\text{: } c.x - 1, y\text{: } c.y\rangle$

Fin TAD

TAD Mapa

géneros mapa

observadores básicos
hayPared? : mapa $\times$ coord $\longrightarrow$ bool
hayDepósito? : mapa $\times$ coord $\longrightarrow$ bool

generadores
nuevo$_M$: $\longrightarrow$ mapa
agPared : mapa m $\times$ coord c $\longrightarrow$ mapa $\{\neg\text{hayPared?}(m, c) \wedge \neg\text{hayDepósito?}(m, c)\}$
agDeposito : mapa m $\times$ coord c $\longrightarrow$ mapa $\{\neg\text{hayPared?}(m, c) \wedge \neg\text{hayDepósito?}(m, c)\}$

otras operaciones
tirarBomba : mapa $\times$ coord $\longrightarrow$ mapa
depósitos : mapa $\longrightarrow$ conj(coord)

axiomas $\forall\ m$: mapa, c, c': coord
hayPared?(nuevo$_M$, c') $\equiv$ false
hayPared?(agPared(m, c), c') $\equiv$ $(c =_{\text{obs}} c') \vee$ hayPared?(m, c')
hayPared?(agDepósito(m, c), c') $\equiv$ hayPared?(m, c')

hayDepósito?(nuevo$_M$, c') $\equiv$ false
hayDepósito?(agPared(m, c), c') $\equiv$ hayDepósito?(m, c')
hayDepósito?(agDepósito(m, c), c') $\equiv$ $(c =_{\text{obs}} c') \vee$ hayDepósito?(m, c')

tirarBomba(nuevo$_M$, c) $\equiv$ nuevo$_M$
tirarBomba(agPared(m, c), c') $\equiv$ **if** $c.x = c'.x \vee c.y = c'.y$ **then**
tirarBomba(m, c')
else
agPared(tirarBomba(m, c'), c)
fi
tirarBomba(agDepósito(m, c), c') $\equiv$ agDepósito(tirarBomba(m, c'), c)

depósitos(nuevo$_M$) $\equiv \emptyset$
depósitos(agPared(m, c)) $\equiv$ depósitos(m)

depósitos(agDepósito(m, c)) $\equiv$ Ag(c, depósitos(m))

Fin TAD

TAD Nivel

géneros nivel

observadores básicos

mapa_N	: nivel	$\longrightarrow$	mapa
persona_N	: nivel	$\longrightarrow$	coord
cajas_N	: nivel	$\longrightarrow$	conj(coord)
$\#\text{bombas}_N$	: nivel	$\longrightarrow$	nat

generadores

nuevo_N : mapa m $\times$ coord p $\times$ conj(coord) cs $\times$ nat b $\longrightarrow$ nat

$$\left\{\begin{array}{l} p \notin cs \wedge \neg\text{hayPared?}(m, p) \wedge \#(\text{depósitos}(m)) = \#(cs) \\ \wedge\ (\forall c : \text{coord})(c \in cs \Rightarrow \neg\text{hayPared?}(m, c)) \end{array}\right\}$$

axiomas $\forall$ m: mapa, p: coord, cs: conj(coord), b: nat

$\text{mapa}_N(\text{nuevo}_N(m, p, cs, b))$	$\equiv$	m
$\text{persona}_N(\text{nuevo}_N(m, p, cs, b))$	$\equiv$	p
$\text{cajas}_N(\text{nuevo}_N(m, p, cs, b))$	$\equiv$	cs
$\#\text{bombas}_N(\text{nuevo}_N(m, p, cs, b))$	$\equiv$	b

Fin TAD

TAD Sokoban

géneros soko

observadores básicos

mapa	: soko	$\longrightarrow$	mapa	
persona	: soko	$\longrightarrow$	coord	
hayCaja?	: soko $\times$ coord	$\longrightarrow$	bool	
#bombas	: soko	$\longrightarrow$	nat	
deshacer	: soko	$\longrightarrow$	soko	

generadores

nuevo_S	: nivel	$\longrightarrow$	soko	
mover	: soko s $\times$ dir d	$\longrightarrow$	soko	{puedeMover?(s, d)}
tirarBomba	: soko s	$\longrightarrow$	soko	$\{\#\text{bombas}(s) > 0\}$

otras operaciones

noHayParedNiCaja?	: soko $\times$ coord	$\longrightarrow$	bool
puedeMover?	: soko $\times$ dir	$\longrightarrow$	bool
ganó?	: soko	$\longrightarrow$	bool
hayCajas?	: soko $\times$ conj(coord)	$\longrightarrow$	bool

axiomas $\forall$ s: soko, d: dir, c: coord

$\text{mapa}(\text{nuevo}_S(nivel))$	$\equiv$	$\text{mapa}_N(nivel)$
mapa(mover(s, d))	$\equiv$	mapa(s)
mapa(tirarBomba(s))	$\equiv$	tirarBomba(mapa(s))
$\text{persona}(\text{nuevo}_S(nivel))$	$\equiv$	$\text{persona}_N(nivel)$
persona(mover(s, d))	$\equiv$	persona(s) $\oplus$ d
persona(tirarBomba(s))	$\equiv$	persona(s)
$\text{hayCaja?}(\text{nuevo}_S(nivel))$	$\equiv$	$c \in \text{cajas}_N(nivel)$
hayCaja?(mover(s, d), c)	$\equiv$	**if** $c = (\text{persona}(s) \oplus d) \oplus d$ **then** hayCaja?(s, persona(s) $\oplus$ d) $\vee$ hayCaja?(s, (persona(s) $\oplus$ d) $\oplus$ d) **else** hayCaja?(s, c) $\wedge$ $\neg(c = (\text{persona}(s) \oplus d))$ **fi**
hayCaja?(tirarBomba(s), c)	$\equiv$	hayCaja?(s, c)
$\#\text{bombas}(\text{nuevo}_S(nivel))$	$\equiv$	$\#\text{bombas}_N(nivel)$
#bombas(mover(s, d))	$\equiv$	#bombas(s)

#bombas(tirarBomba(s))	$\equiv$	#bombas(s) - 1
deshacer(nuevo$_S$($nivel$))	$\equiv$	nuevo$_S$($nivel$)
deshacer(mover(s, d))	$\equiv$	s
deshacer(tirarBomba(s))	$\equiv$	s
noHayParedNiCaja?(s, c)	$\equiv$	$\neg$hayPared?(mapa(s), c) $\wedge$ $\neg$hayCaja?(s, c)
puedeMover?(s, d)	$\equiv$	noHayParedNiCaja?(s, persona(s) $\oplus$ d) $\vee$ (hayCaja?(s, persona(s) $\oplus$ d) $\wedge$ noHayParedNiCaja?(s, (persona(s) $\oplus$ d) $\oplus$ d))
ganó?(s)	$\equiv$	hayCajas?(s, depósitos(mapa(s)))
hayCajas?(s, cs)	$\equiv$	$\emptyset$?(cs) $\vee_{\text{L}}$ (hayCaja?(s, dameUno(cs)) $\wedge$ hayCajas?(s, sinUno(cs)))

Fin TAD

TAD Juego

géneros juego

observadores básicos

nivelActual	: juego	$\longrightarrow$	soko	
nivelesPendientes	: juego	$\longrightarrow$	secu(nivel)	

generadores

nuevo$_J$	: secu(nivel) ns	$\longrightarrow$	juego	{$\neg$vacía?(ns)}
mover	: juego j $\times$ dir d	$\longrightarrow$	juego	{puedeMover?(nivelActual(j), d)}
tirarBomba	: juego j	$\longrightarrow$	juego	{#bombas(nivelActual(j)) $>$ 0}
deshacer	: juego j	$\longrightarrow$	juego	

axiomas

nivelActual(nuevo$_J$(ns))	$\equiv$	nuevo$_S$(prim(ns))
nivelActual(mover(j, d))	$\equiv$	**if** ganó?(mover(nivelActual(j), d)) $\wedge$ $\neg$vacía?(nivelesPendientes(ns)) **then** prim(nivelesPendientes(ns)) **else** mover(nivelActual(j), d)) **fi**
nivelActual(tirarBomba(j))	$\equiv$	tirarBomba(nivelActual(j))
nivelActual(deshacer(j))	$\equiv$	deshacer(nivelActual(j))
nivelesPendientes(nuevo$_J$(ns))	$\equiv$	fin(ns)
nivelesPendientes(mover(j, d))	$\equiv$	**if** ganó?(mover(nivelActual(j), d)) $\wedge$ $\neg$vacía?(nivelesPendientes(ns)) **then** fin(nivelesPendientes(ns)) **else** nivelesPendientes(ns) **fi**
nivelesPendientes(tirarBomba(j))	$\equiv$	nivelesPendientes(j)
nivelesPendientes(deshacer(j))	$\equiv$	nivelesPendientes(j)

Fin TAD

Entrega

Para la entrega deben hacer `commit` y `push` de un único documento digital en formato `pdf` en el repositorio **grupal** en el directorio `tpg3/`. El documento debe incluir el diseño completo del enunciado incluyendo todos los módulos, cada uno con su interfaz, estructuras de representación, invariante de representación, función de abstracción, implementación de los algoritmos y descripción de los servicios usados. Se recomienda el uso de los paquetes de LaTeX de la cátedra para lograr una mejor visualización del informe.

3.2. Taller: Conjunto sobre Árbol Binario de Búsqueda

En este taller se debe implementar un conjunto sobre árbol binario de búsqueda para un tipo paramétrico `T`. El enunciado corresponde al del taller de conjunto sobre ABB en L07.

Entrega

Para la entrega deben hacer `commit` y `push` de los archivos fuente (`.h`, `.cpp`) necesarios en el repositorio **individual** en el directorio `g3/taller/src/`. **Respetar los nombres originales de los archivos**. No es necesario que incluyan los tests en el directorio `g1/taller/tests/` pero pueden hacerlo. No incluir archivos binarios en el repositorio (`*.o`, `*.a`, `*.exe`, etc.).