Guía 3 — fecha límite: domingo 21 de junio

Algoritmos y Estructuras de Datos II, DC, UBA.

Primer cuatrimestre de 2020

Índice

1. Modalidad de trabajo

Como todes sabemos, este cuatrimestre las clases se dictarán a distancia con motivo del Aislamiento Social Preventivo y Obligatorio dispuesto por el Decreto 297/2020. Si el aislamiento terminara antes del final del cuatrimestre y la Facultad volviera a su funcionamiento normal, informaremos cómo continuar con la cursada. Por el momento no podemos asegurar que la cursada vuelva a ser presencial, ni que siga siendo a distancia hasta el final del cuatrimestre.

1.1. Cronograma de ejercicios para esta guía

Este es un cronograma *sugerido* para que vayan trabajando. La idea no es que sea fijo, sino brindarles lo que quienes hicimos la guía pensamos que sería un ritmo realizable.

Lunes	Martes	Miércoles	Jueves	Viernes	Sábado	Domingo
1 de junio	2 de junio	3 de junio	4 de junio	5 de junio Quiz 2.1.1 Quiz 2.1.2 Empezar 2.2	6 de junio	7 de junio
8 de junio Empezar **??**	9 de junio	10 de junio Quiz 2.1.3	11 de junio	12 de junio Corrección 2.2 Corrección **??**	13 de junio	14 de junio
15 de junio	16 de junio	17 de junio	18 de junio	19 de junio	20 de junio	21 de junio **límite entrega**

1.2. 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. Dado que es la primera vez que la mayoría de les docentes dicta cursos a distancia, solicitamos ser comprensives con todo tipo de falencias que las clases puedan llegar a presentar. Aceptamos todo tipo de crítica constructiva o sugerencia.

1.3. Turnos

Desde el punto de vista administrativo, docentes y alumnes estarán en un único turno.

1.4. 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 (a través de algún software que indicaremos y podrá depender de diversos factores, ej. Telegram o Zoom). El cronograma de consultas sincrónicas se ajustará a la demanda de consultas y la disponibilidad docente bajo consideración de las circunstancias especiales actuales. Buscaremos respetar que los horarios de consulta sincrónica estén dentro de los horarios de la cursada presencial con el fin de respetar las disponibilidades establecidas anteriormente. En particular, *trataremos* de que haya docentes disponibles durante las siguientes franjas horarias:

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

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

1.5. 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.

1.6. Guías de ejercitación y criterios de aprobación

La materia se dividirá en bloques. De manera tentativa, habría **cuatro bloques**, cada uno de **tres semanas** de duración (pero esto puede llegar a cambiar). 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 el foro y eventualmente se publicarán soluciones de algunas o varias de ellas.
2. **Ejercicios obligatorios de la práctica**. Los ejercicios son individuales y se deben entregar al finalizar el bloque. Los ejercicios serán corregidos y devueltos. Cada entrega podrá ser aprobada o devuelta para incorporar correcciones. Se detallan en la Sección 3 (margen azul).
3. **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 4 (margen verde).

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 finalice la cuarentena y la Facultad vuelva a funcionar con normalidad, se planifica también un **Examen integrador** presencial para el día viernes 17/07 (con su respectivo recuperatorio el viernes 31/07). Tener en cuenta que, dado el contexto excepcional de este cuatrimestre, el mecanismo de evaluación podrá sufrir modificaciones.

1.7. 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 13/04 — T01, T02: Especificación
   - Mié 15/04 — L01: Uso de clases
   - Vie 17/04 — Consultas
2. **Semana 2**
   - Lun 20/04 — P01: TADs y recursión (básico) ..... **Publicación de la guía 1**
   - Mié 22/04 — L02: Clases en C++ (básico), testing
   - Vie 24/04 — P02: TADs y recursión (avanzado)
3. **Semana 3**
   - Lun 27/04 — P03: TAD en dos niveles, T03: Complejidad
   - Mié 29/04 — L03: Clases en C++
   - Vie 01/05 — **Feriado: Día del Trabajador**

4. **Semana 4**
   - Lun 04/05 — T04: Diseño, T03: Consultas 11:00 y 17:00 hs., Consultas 12:00 y 18:00 hs.
   - Mié 06/05 — L04: Memoria dinámica
   - Vie 08/05 — T04: Consultas 11:00 y 17:00 hs.
   - Dom 10/05 ..... **Entrega de la guía 1**
5. **Semana 5**
   - Lun 11/05 — P04: Notación "O", complejidad ..... **Publicación de la guía 2**
   - Mié 13/05 — L05: Listas enlazadas
   - Vie 15/05 — T05: Diccionarios sobre ABBs, T03 y T04: Consultas 11:00 y 17:00 hs., Consultas 12:00 y 18:00 hs.
6. **Semana 6**
   - Lun 18/05 — P05: Invariante de representación y función de abstracción
   - Mié 20/05 — L06: Templates y algoritmos genéricos
   - Vie 22/05 — T05': Diccionarios sobre AVL, T05: Consultas 11:00 y 17:00 hs. P06: Repaso
7. **Semana 7**
   - Lun 25/05 — **Feriado: Día de la Revolución de Mayo**
   - Mié 27/05 — Consultas
   - Vie 29/05 — T06: Tries, T05 y T05': Consultas 11:00 y 17:00 hs.
   - Dom 31/05 ..... **Entrega de la guía 2**
8. **Semana 8**
   - Lun 01/06 — T07: Hashing 1 y 2 ..... **Publicación de la guía 3**
   - Mié 03/06 — L07: ABBs
   - Vie 05/06 — P07: Interfaces, elección de estructuras
9. **Semana 9**
   - Lun 08/06 — T08: Colas de prioridad
   - Mié 10/06 — Consultas
   - Vie 12/06 — P08: Elección de estructuras, iteradores
10. **Semana 10**
    - Lun 15/06 — **Feriado: Paso a la Inmortalidad del General Martín Miguel de Güemes**
    - Mié 17/06 — L08: Tries
    - Vie 19/06 — P09: Ejercitación avanzada
    - Dom 21/06 ..... **Entrega de la guía 3**
11. **Semana 11**
    - Lun 22/06 — T09: Sorting básico ..... **Publicación de la guía 4**
    - Mié 24/06 — L09: Heaps
    - Vie 26/06 — P10: Sorting
12. **Semana 12**
    - Lun 29/06 — T10: Divide and conquer
    - Mié 01/07 — Consultas
    - Vie 03/07 — P11: Divide and conquer, recurrencias y teorema maestro

13. **Semana 13**
    - Lun 06/07 — P12: D&C avanzado
    - Mié 08/07 — L10: Desarrollo de iteradores
    - Vie 10/07 — **Feriado puente**
    - Dom 12/07 ..... **Entrega de la guía 4**
14. **Semana 14**
    - Lun 13/07 — T11: Sorting y diccionarios en memoria externa
    - Mié 15/07 — Consultas
    - Vie 17/07 — Consultas
15. **Semana 15**
    - Lun 20/07 — T12: Splay trees y skip lists — Consultas ..... **Presentación del recuperatorio**
    - Mié 22/07 — L12: Sorting (ex taller de sorting)[2]
    - Vie 24/07 — Consultas
16. **Semana 16**
    - Lun 27/07 — Consultas ..... **Entrega del recuperatorio**
    - Mié 29/07 — Consultas

[2] Esta actividad solamente está programada de manera presencial, en caso de que se levante la cuarentena y la Facultad vuelva a su funcionamiento normal.

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 $\times$ nat d $\times$ nat h $\longrightarrow$ nat $\quad \{1 \leq d \wedge d \leq h \wedge h \leq totDias(i)\}$

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

generadores

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

otras operaciones

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

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 de la práctica

IMPORTANTE: Los ejercicios de esta sección son de carácter individual y serán calificados. Cada ejercicio podrá estar aprobado, o será devuelto para incorporar correcciones. Las consultas sobre estos ejercicios deben limitarse a expresar dudas de interpretación sobre el enunciado. Además, como el objetivo de estos ejercicios es evaluar el desempeño individual, no se permite trabajar grupalmente ni compartir soluciones. La fecha límite de entrega es la fecha de finalización del bloque.

3.1. Elección de estructuras

Ejercicio 1: elección de estructuras

En varios negocios de la ciudad Roque publicará listas de regalos que le interesa recibir para su cumpleaños. Cada regalo tiene un precio. Como sus invitades son extremadamente mezquinos, cuando alguno quiere comprar un regalo para Roque, siempre se decide por el más barato de los disponibles entre todos los negocios en los que haya algún regalo publicado. Cuando alguien compra el regalo más barato, ese regalo se retira de la lista de regalos del negocio en el que estuviera publicado. Los negocios y regalos se identifican usando códigos numéricos. Se transcribe un fragmento de la especificación:

TAD NEGOCIO es NAT

TAD REGALO es NAT

TAD CUMPLEAÑOS

observadores básicos

negocios	: cumple	$\longrightarrow$ conj(negocio)		
regalos	: cumple c $\times$ negocio n	$\longrightarrow$ conj(regalo)		$\{n \in \text{negocios}(c)\}$
precio	: cumple c $\times$ negocio n $\times$ regalo r	$\longrightarrow$ nat	$\{n \in \text{negocios}(c) \wedge_{\text{L}} r \in \text{regalos}(c, n)\}$	

generadores

nuevo	:	$\longrightarrow$ cumple	
publicarLista	: cumple c $\times$ negocio n $\times$ dicc(regalo, nat) ℓ	$\longrightarrow$ cumple	
	$\{n \notin \text{negocios}(c) \wedge (\forall n' : \text{negocio})(n' \in \text{negocios}(c) \Rightarrow_{\text{L}} \emptyset?(\text{regalos}(c, n') \cap \text{claves}(\ell)))\}$		
comprarRegaloMásBarato	: cumple c	$\longrightarrow$ cumple	
		$\{(\exists n\text{: negocio})(n \in \text{negocios}(c) \wedge_{\text{L}} \neg\emptyset?(\text{regalos}(c, n)))\}$	

otras operaciones

regaloMásBarato	: cumple c	$\longrightarrow$ regalo	
		$\{(\exists n\text{: negocio})(n \in \text{negocios}(c) \wedge_{\text{L}} \neg\emptyset?(\text{regalos}(c, n)))\}$	

Fin TAD

Diseñar el módulo CUMPLEAÑOS, de tal modo que provea las siguientes operaciones con las complejidades temporales en peor caso indicadas. Escribir la estructura y detallar los algoritmos de las cuatro operaciones siguientes:

1. PUBLICARLISTA(**in/out** c: cumple, **in** n: negocio, **in** ℓ: dicc(regalo, nat))
   Publica los regalos (con sus precios asociados) en el negocio indicado.
   Complejidad: $O(L + \log(N))$ donde $L = \#\text{claves}(\ell)$ es el tamaño de la lista de regalos que se publican en este momento y $N = \#\text{negocios}(c)$ es la cantidad total de negocios.

2. REGALOS(**in** c: cumple, **in** n: negocio) $\longrightarrow$ *res* : conj(regalo)
   Devuelve el conjunto de regalos publicados en el negocio indicado. Notar que este conjunto puede ir cambiando a medida que les invitades compran regalos (i.e. aprovechar aspectos de aliasing).
   Complejidad: $O(\log(N))$ donde $N = \#\text{negocios}(c)$ es la cantidad total de negocios.

3. NEGOCIOSCONREGALOS(**in** c: cumple) $\longrightarrow$ *res* : conj(negocio)
   Devuelve el conjunto de negocios que todavía cuentan con regalos. Este conjunto también puede ir cambiando a medida que les invitades compran regalos.
   Complejidad: $O(1)$.

4. REGALOMÁSBARATO(**in** c: cumple) $\longrightarrow$ *res* : regalo
   Devuelve el regalo más barato publicado, considerando las listas de todos los negocios en los que haya regalos publicados. En caso de que haya varios regalos más baratos con el mismo precio, se admite cualquier manera de "desempatar" entre ellos.
   Complejidad: $O(1)$.

5. compRarRegaloMásBarato(**in/out** c: cumple)
   Saca el regalo más barato de la lista del negocio en el que esté publicado.
   Complejidad: $O(N + \log R)$ donde $N = \#\text{negocios}(c)$ es la cantidad total de negocios y R es la cantidad total de regalos de todo el sistema.
   Esta operación se puede implementar todavía más eficientemente en $O(\log N + \log R)$, o incluso en $O(\log R)$. Se valorará que el alumne intente resolver el ejercicio con la mejor cota de complejidad posible, aunque el mínimo requisito para que el ejercicio esté aprobado es que cumpla con la cota $O(N + \log R)$.

1. Dar una estructura de representación del módulo Cumpleaños explicando detalladamente qué información se guarda en cada parte, las relaciones entre las partes, y las estructuras de datos subyacentes.

2. Justificar de qué manera es posible implementar los algoritmos para cumplir con las complejidades pedidas. Escribir el algoritmo para la operación CompRarRegaloMásBarato.

Para la resolución del ejercicio no esta permitido utilizar módulos implementados con tabla hash de base, esto se debe a dos motivos, uno porque queremos que combinen el resto de las estructuras vistas y otro porque los peores casos de la tabla hash exceden los que se piden en los ejercicios (duplicar el espacio en una tabla cuesta por ejemplo O(n) en el peor caso).

4. 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. La fecha límite de entrega es la fecha de finalización del bloque.

4.1. TP 2

4.2. Enunciado

En el trabajo práctico de diseño se busca escribir los módulos que comprenderán la futura implementación del `SimCity`. La idea general del juego, es la misma a la trabajada durante el TP de especificación. No obstante, para unificar criterios, presentamos una especificación de la cátedra (link).
El entregable consiste en la presentación de los módulos de diseño, lo que incluye Interfaz, Estructura de Representación, Invariante de Representación, Función de Abstracción y Algoritmos. La interfaz de cada módulo debe declarar las funciones que la componen, con aridad completa (valores de entrada y salida) y contrato (precondición, postcondición y complejidad).
Además, la elección de estructuras de repesentación y algoritmos deberá ser de forma que se cumplan las restricciones de complejidad presentadas a continuación. Para ello, primero repasamos las operaciones que producen cambios en un `SimCity`:

- `agregar casa`. Un SimCity puede recibir una posición donde debe agregarse una casa.
- `agregar comercio`. Un SimCity puede recibir una posición donde debe agregarse un comercio.
- `unir`. Un SimCity pueden recibir otro SimCity e incorporarlo mediante la unión de la información de ambos.

Luego, el diseño a realizar debe cumplir con las siguientes restricciones de complejidad:

1. Conocer el turno actual de una partida debe ser $O(1)$
2. Agregar una casa a una partida debe ser $O(1)$
3. Agregar un comercio a una partida debe ser $O(1)$
4. Unir dos partidas debe ser $O(1)$ (sin considerar el costo de copia de `SimCity`)

Tener en cuenta que:

- Dado que es un mapa infinito, no se puede asumir que un hash sería $O(1)$
- El resto de las operaciones no tienen restricciones de complejidad

4.3. Entrega

La entrega deberá ser un único documento en formato `pdf` con la descripción de los módulos, que incluye todo lo mencionado anteriormente. Este documento debe contener todos los módulos necesarios para una solución satisfactoria del `SimCity`.
Si bien no es necesario realizar la solución en LaTeX, en el campus se encuentra, junto a la especificación de referencia, una versión incompleta del módulo `Mapa` que puede ser utilizada también de referencia, junto con su archivo `.tex` para completar (link).
La fecha de entrega concuerda con la entrega de la guía 3 y seguirá la metodología de las otras entregas. En este caso debe haber encontrarse el documento en la carpeta `tpg3` del repositorio de entrega final.

4.4. 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

La entrega deberá consistir en agregar, commitear y pushear a su repositorio de trabajo indivudal el contenido del directorio con la ejercitación del taller (que contiene el archivo `CMakeLists.txt` y los directorios `src` y `tests`) al directorio `g3/taller`. Lo importante es que dentro del último (`g3/taller`) puedan encontrarse los archivos fuente del taller: `Conjunto.h` y `Conjunto.hpp`.