Guía 2 — fecha límite: domingo 31 de mayo

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
11 de mayo	12 de mayo	13 de mayo	14 de mayo	15 de mayo Quiz 2.1.1 Quiz 2.1.2 Empezar 2.2	16 de mayo	17 de mayo
18 de mayo Quiz 2.1.3 Empezar 2.3	19 de mayo	20 de mayo	21 de mayo	22 de mayo Corrección 2.2 Corrección 2.3	23 de mayo	24 de mayo
25 de mayo	26 de mayo	27 de mayo	28 de mayo	29 de mayo	30 de mayo	31 de mayo **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, iteradores, elección de estructuras
9. **Semana 9**
   - Lun 08/06 — T08: Colas de prioridad
   - Mié 10/06 — Consultas
   - Vie 12/06 — P08: Elección de estructuras 2
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: Elección de estructuras avanzadas
    - 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 T03: Complejidad

- En 1969, el matemático alemán Volker Strassen propuso un algoritmo que calculaba productos de matrices de $n \times n$ con complejidad temporal $O(n^{2.807})$. Un procesador típico hoy en día puede llegar a ser 100.000 veces más rápido que un procesador típico de 1969. ¿Cómo se ve afectada la complejidad del algoritmo por esta diferencia?
- El algoritmo $\mathcal{A}$ recibe como entrada una lista de n números y la procesa en tiempo lineal. Más precisamente, su complejidad temporal, tanto en mejor como en peor caso, es $\Theta(n)$. Roque ejecuta el algoritmo $\mathcal{A}$ sobre una lista de 100 elementos y observa que tarda 100 segundos (con un margen de error de 1 décima de segundo). ¿Qué puede decirse sobre lo que observaría Roque si ejecutara el mismo algoritmo sobre una lista de 200 elementos?
- El algoritmo $\mathcal{B}$ recibe una lista de números y la procesa en tiempo constante, es decir, en $O(1)$. Roque corre el algoritmo $\mathcal{B}$ sobre la misma lista de 100 elementos. ¿Qué puede decirse sobre lo que observa Roque?
- El algoritmo $\mathcal{C}$ recibe un árbol de n nodos y lo procesa de alguna manera. ¿Cuáles de las siguientes combinaciones son posibles y cuáles imposibles?
  1. La complejidad temporal de $\mathcal{C}$ es $O(1)$ en mejor caso y $O(n)$ en peor caso.
  2. La complejidad temporal de $\mathcal{C}$ es $O(n^2)$ en mejor caso y $\Omega(n)$ en peor caso.
  3. La complejidad temporal de $\mathcal{C}$ es $\Omega(2^n)$ en mejor caso y $O(n)$ en peor caso.

2.1.2. Clase T04: Diseño

- Recordemos que el **principio de abstracción** afirma que el usuario de un módulo sólo puede interactuar con él a través de su interfaz ("parte pública"), en tanto que no debe acceder a su representación ("parte privada"). ¿Para qué puede servir considerar diferentes representaciones de un mismo módulo, si el usuario nunca va a poder acceder a ellas?
- Supongamos que el módulo Lista está representado sobre un arreglo. El usuario del módulo, ¿puede usar el hecho de que la complejidad temporal de acceder al i-ésimo elemento de la lista es $O(1)$?
- Supongamos que el módulo Conjunto(Nat) se representa sobre una lista sin repetidos. Roque propone cambiar la representación para usar un arreglo ordenado de menor a mayor. ¿Qué impacto tiene este cambio en la interfaz del módulo? ¿Qué impacto tiene este cambio en la representación y algoritmos del módulo? ¿Qué impacto tiene este cambio en los demás módulos (usuarios del módulo Conjunto(Nat))? ¿Cómo se modifica el invariante de representación? ¿Cómo se modifica la función de abstracción?
- ¿Cuáles de las siguientes situaciones son normales y cuáles necesariamente implican un error de programación?
  - El programador del módulo rompe el invariante de representación de una instancia en una función **privada** y **no** lo reestablece al final de la función.

- Igual que arriba, pero **sí** reestablece el invariante.
- El programador del módulo rompe el invariante de representación de una instancia en una función **pública** (de la interfaz) y **no** lo reestablece al final de la función.
- Igual que arriba, pero **sí** reestablece el invariante.

¿Cómo podría hacer el usuario de un módulo para romper el invariante de representación de la instancia?

2.1.3. Clase T05: Diccionarios sobre ABBs y AVLs

- Si n es la cantidad de nodos de un árbol **perfectamente balanceado** y h es la altura del árbol. ¿Cuáles de las siguientes afirmaciones son ciertas?:

$$h \in O(n) \qquad h \in \Omega(n) \qquad h \in O(\log n) \qquad h \in \Omega(\log n)$$

- Igual que arriba, pero si el árbol es un árbol **arbitrario**, no necesariamente balanceado.
- Igual que arriba, pero si el árbol cumple con el invariante de balanceo de un **AVL**.

2.2. Complejidad

Ejercicio 1

Sean $f, g : \mathbb{N} \to \mathbb{N}$, y supongamos que está definido el límite $\lim_{n\to+\infty} \frac{f(n)}{g(n)} = \ell \in \mathbb{R}_{\geq 0} \cup \{+\infty\}$. Probar que:

a) $0 < \ell < +\infty$ si y sólo si $f \in \Theta(g)$.
b) $\ell = +\infty$ si y sólo si $f \in \Omega(g)$ y $f \notin O(g)$.
c) $\ell = 0$ si y sólo si $f \in O(g)$ y $f \notin \Omega(g)$.

Recordar las definiciones de límite:

- $\lim_{n\to+\infty} a_n = \ell \in \mathbb{R}$ si $\forall\, \varepsilon > 0.\, \exists n_0 \in \mathbb{N}.\, \forall n > n_0.\, |a_n - \ell| < \varepsilon$.
- $\lim_{n\to+\infty} a_n = +\infty$ si $\forall\, M > 0.\, \exists n_0 \in \mathbb{N}.\, \forall n > n_0.\, a_n > M$.

Ejercicio 2

Determinar el orden de complejidad temporal de peor caso de los siguientes algoritmos, asumiendo que todas las operaciones sobre arreglos y matrices toman tiempo $O(1)$.
La complejidad se debe calcular en función de una medida de los parámetros de entrada, por ejemplo, la cantidad de elementos en el caso de los arreglos y matrices y el valor en el caso de parámetros naturales.

Sumatoria, que calcula la sumatoria de un arreglo de enteros:

```
1: function Sumatoria(arreglo A)
2:     int i, total;
3:     total := 0;
4:     for i := 0 ... Long(A) - 1 do
5:         total := total + A[i];
6:     end for
7: end function
```

SumatoriaLenta, que calcula la sumatoria de n, definida como la suma de todos los enteros entre 1 y n, de forma poco eficiente:

```
1: function SumatoriaLenta(natural N)
2:     int i, total;
3:     total := 0;
4:     for i := 1 ... n do
5:         for j := 1 ... i do
6:             total := total + 1;
7:         end for
8:     end for
9: end function
```

InsertionSort, que ordena un arreglo pasado como parámetro:

```
1: function InsertionSort(arreglo A)
2:     int i, j, valor;
3:     for i := 0 ... Long(A) - 1 do
4:         valor := A[i];
5:         j := i - 1;
6:         while j ≥ 0 ∧ a[j] > valor do
7:             A[j+1] := A[j];
8:             j := j - 1;
9:         end while
10:        A[j+1] := valor;
11:    end for
12: end function
```

BúsquedaBinaria, que determina si un elemento se encuentra en un arreglo, que debe estar ordenado:

```
1: function BúsquedaBinaria(arreglo A, elem valor)
2:     int izq := 0, der := Long(A) - 1;
3:     while izq < der do
4:         int medio := (izq + der) / 2;
5:         if valor < A[medio] then
6:             der := medio;
7:         else
8:             izq := medio;
9:         end if
10:    end while
11:    return A[izq] = valor;
12: end function
```

ProductoMat, que dadas dos matrices A (de $p \times q$) y B (de $q \times r$) devuelve su producto AB (de $p \times r$):

```
1: function ProductoMat(matriz A, matriz B)
2:     int fil, col, val, colAFilB;
3:     matriz res(Filas(A), Columnas(B));
4:     for fil := 0 ... Filas(A) - 1 do
5:         for col := 0 ... Columnas(B) - 1 do
6:             val := 0;
7:             for colAFilB := 0 ... Columnas(A) - 1 do
8:                 val := val + (A[fil][colAFilB] * B[colAFilB][col]);
9:             end for
10:            res[fil][col] := val;
11:        end for
12:    end for
13:    return res;
14: end function
```

Ejercicio 3

Para cada una de las siguientes afirmaciones, decida si son verdaderas o falsas y justifique su decisión.

- $O(n^2) \cap \Omega(n) = \Theta(n^2)$
- $\Theta(n) \cup \Theta(n \log n) = \Omega(n \log n) \cap O(n)$
- Existe una $f : \mathbb{N} \to \mathbb{N}$ tal que para toda $g : \mathbb{N} \to \mathbb{N}$ se cumple que $g \in O(f)$.
- Sean $f, g : \mathbb{N} \to \mathbb{N}$, entonces se cumple que $O(f) \subseteq O(g)$ o $O(g) \subseteq O(f)$ (es decir, el orden sobre funciones dado por la inclusión de la O es total).

2.3. Invariante de representación y función de abstracción

Ejercicio 1

Se propone la siguiente estructura para representar a los polinomios que tienen a lo sumo grado n (ver TAD en Práctica 1):

polinomio **se representa con** estr, donde estr es tupla

⟨*grado*: nat,
coef: array[0 . . . n] de nat⟩

Se pide:

1. Definir el invariante de representación y la función de abstracción.
2. Escribir la interface completa y el algoritmo para la función evaluar.

Ejercicio 2

Los palíndromos son aquellas palabras que pueden leerse al derecho o al revés. El siguiente TAD describe a los palíndromos:

TAD PALÍNDROMO(α)

observadores básicos
ver : palindromo(α) $\longrightarrow$ secu(α)

generadores
medio : α $\longrightarrow$ palindromo(α)
medioDoble : α $\longrightarrow$ palindromo(α)
agregar : $\alpha \times$ palindromo(α) $\longrightarrow$ palindromo(α)

axiomas
ver(medio(a)) $\equiv$ a • <>
ver(medioDoble(a)) $\equiv$ a • a • <>
ver(agregar(a,p)) $\equiv$ a • (ver(p) $\circ$ a)

Fin TAD

Se propone la siguiente estructura de representación:

palindromo **se representa con** estr, donde estr es tupla

⟨*long*: nat,
palabra: secu(α)⟩

dónde *palabra* representa el palíndromo completo.
Se pide:

- Definir el invariante de representación y la función de abstracción.
- Escribir la interface y el algoritmo para la función ver.
- Rehacer los ítems anteriores si el campo *palabra* en lugar de la palabra completa guardamos sólo la mitad inicial de la palabra (redondeando hacia arriba).

Ejercicio 3

El salón Alta Fiesta se encuentra últimamente en dificultades para coordinar el desarrollo de las reuniones que se realizan en sus facilidades. Hoy en día las fiestas se desarrollan de una manera muy particular. Los invitados llegan en grupos, por lo general numerosos, aunque también a veces de una sola persona. Que un invitado llegue sin un regalo está considerado una falta grave a las buenas costumbres. Tanto es así que a dichos individuos no se les permite el ingreso a las fiestas. Lo que sí se permite es que los invitados hagan regalos en grupo: los invitados que llegan en grupo traen un único regalo de parte de todos. Como es habitual, sólo se permite la entrada a la fiesta a aquellas personas que han sido invitadas.
Al ingresar un grupo de invitados a la fiesta, éste se identifica con un nombre: por ejemplo "Los amigos de la secundaria" o "La familia de la novia". Este nombre se usa por ejemplo para los juegos grupales que van a hacer los animadores durante la fiesta. Igualmente, dado que el comportamiento de las personas en masa no siempre es civilizado, se quiere poder saber de manera eficiente cuál es el nombre del grupo más numeroso para poder seguirle el rastro.
Además, se desea tener un registro de todos los regalos, junto con el grupo de personas que lo hicieron, y se desea conocer en todo momento qué personas se encuentran ya en la fiesta.
Se nos ha encomendado realizar un sistema que permite llevar el control del estado de la fiesta en un momento dado. La cátedra ha realizado la especificación del sistema y ha armado una estructura de representación para el diseño del mismo.

TAD AltaFiesta

observadores básicos

invitadosPendientes : AltaFiesta $\longrightarrow$ conj(Persona)
regalos : AltaFiesta $\longrightarrow$ conj(Regalo)
personasPorRegalo : AltaFiesta a × Regalo r $\longrightarrow$ conj(Persona) {r ∈ regalos(a)}
grupoMasNumeroso : AltaFiesta $\longrightarrow$ Grupo

generadores

iniciarFiesta : conj(Persona) *invitados* $\longrightarrow$ AltaFiesta
lleganInvitados : AltaFiesta a × conj(Persona) c × Nombre n × Regalo r $\longrightarrow$ AltaFiesta
{c ⊆ invitadosPendientes(a) ∧ n ∉ grupos(a) ∧ r ∉ regalos(a)}

axiomas

...

Fin TAD

altafiesta **se representa con** estr, donde estr es tupla

⟨*invitados*: conj(persona),
presentes: cola(persona),
grupoDe: dicc(grupo, conj(persona)),
regaloDeGrupo: dicc(grupo, regalo),
grupoMasNumeroso: grupo⟩

grupo, persona y regalo **son** string

Informalmente, esta representación cumple las siguiente propiedades:

- En *invitados* están todos los invitados a la fiesta, incluyendo también a aquellos que ya llegaron.
- En *presentes* están los invitados que ya llegaron a la fiesta.
- En *grupoDe* se encuentra, para cada identificador de grupo i, las personas que al llegar agrupadas se identificaron como i.
- En *regaloDeGrupo* se encuentra qué regalo trajo cada grupo.

- *grupoMasNumeroso* contiene al identificador del grupo de más personas. En caso de empate, contiene al lexicográficamente menor de los empatados (se asume que la función $<_{string}$ está definida).

Se pide:

1. Realizar el invariante de representación del módulo. Escribirlo en lenguaje formal y en castellano. Para que quede claro cuáles enunciados informales hacen referencia a cuáles enunciados formales se recomienda numerar cada punto del invariante para luego poder hacer referencia al mismo.
2. Escribir la función de abstracción. De considerarse necesario, explicarla en castellano.
3. Escribir una versión imperativa de la función llegaGrupo marcando claramente los puntos de su programa en que alguna parte del invariante de representación se rompe, indicando a qué parte se refiere, y también aquellos puntos donde éste se reestablece.

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. Complejidad

Ejercicio 1: funciones polinomiales vs. funciones exponenciales.
El objetivo de este ejercicio es demostrar que todas las funciones polinomiales de la forma x^p, donde el exponente $p \in \mathbb{N}$ está fijo, están acotadas por todas las funciones exponenciales de la forma b^x, donde la base $b \in \mathbb{R}$ está fija y $b > 1$. Es decir que se verifica $x^p \in O(b^x)$. Por ejemplo, tenemos que $x^5 \in O((\frac{3}{2})^x)$.

Para ello, demostraremos gradualmente varias desigualdades:

(a) Sea $b \in \mathbb{R}$ tal que $b \geq 2$. Demostrar que $n \leq b^n$ para todo $n \in \mathbb{N}$, por inducción en n.

(b) Sea como en el ítem anterior $b \in \mathbb{R}$ tal que $b \geq 2$. Demostrar que $x \leq b^{x+1}$ para todo $x \in \mathbb{R}_{\geq 0}$.

Notar que ahora $x \in \mathbb{R}_{\geq 0}$ es un número real no negativo. *Sugerencia:* acotar x por $\lfloor x \rfloor + 1$, donde $\lfloor x \rfloor$ denota la parte entera de x, y usar el ítem (a).

(c) Sea ahora $b \in \mathbb{R}$ tal que $b > 1$. Como $b > 1$, sabemos[3] que existe una constante $k \in \mathbb{N}$ tal que $b^k \geq 2$. Demostrar que $\frac{x}{k} \leq b^{x+k}$ para todo $x \in \mathbb{R}_{\geq 0}$.

Sugerencia: usar el ítem (b).

(d) Sean como en el ítem anterior $b \in \mathbb{R}$ tal que $b > 1$ y $k \in \mathbb{N}$ tales que $b^k \geq 2$. Demostrar que para todo $x \in \mathbb{R}_{\geq 0}$ y para todo $n, p \in \mathbb{N}$ vale la siguiente desigualdad:

$$\left(\frac{x}{pk}\right)^n \leq b^{n(\frac{x}{p}+k)}$$

Sugerencia: proceder por inducción en n y usar el ítem (c).

(e) Demostrar que, vistas como funciones de $x \in \mathbb{R}_{\geq 0}$, vale:

$$x^p \in O(b^x)$$

para toda base $b \in \mathbb{R}$ tal que $b > 1$ y para todo exponente $p \in \mathbb{N}$.

Sugerencia: Usar el ítem (d), tomando $n = p$ y despejando x^p para obtener una cota.

Nota: puede usar un ítem para resolver los siguientes a pesar de que no lo haya resuelto.

Ejercicio 2: Determinar las complejidades en mejor y peor caso del siguiente algoritmo `P`. Expresarlas en función del tamaño del arreglo de entrada:

```
function P(A : arreglo(nat)) -> B : arreglo(nat) {
    n := tam(A)
    M := 0
    for i := 0 to n - 1 {
        if (A[i] >= n) { A[i] := 0 } else { M := max(M, A[i]) }
    }
    B := nuevo arreglo(nat) indexado desde 0 hasta M inclusive, inicializado en 0
    for i := 0 to M {
        for j := i to M {
            B[A[i]] := 1 + B[A[i]] + B[A[j]]
        }
    }
    return B
}
```

[3] Basta con tomar cualquier $k \geq \log_b(2)$.

3.2. Invariante de representación y función de abstracción

Ejercicio 3: Un editor de texto tiene varias **pestañas**. Cada vez que se abre una pestaña, se le asigna un número único de manera incremental (0, 1, 2, ...). En cada pestaña se almacena un texto (lista de caracteres). Siempre hay por lo menos una pestaña en el editor, y exactamente una de ellas es la pestaña **seleccionada**, sobre la que se encuentra el cursor. El cursor está ubicado sobre alguna posición del texto de la pestaña seleccionada. Si la pestaña seleccionada tiene n letras, el cursor puede encontrarse entre las posiciones 0 y n inclusive—por ejemplo si el texto es "abc" las posiciones posibles son 0 ("|abc"), 1 ("a|bc"), 2 ("ab|c") y 3 ("abc|"). El usuario puede mover el cursor hacia la izquierda ($\Leftarrow$) o la derecha ($\Rightarrow$), siempre que no se encuentre en alguno de los bordes, y puede insertar una letra en la posición actual. La siguiente es una especificación (incompleta) del TAD que modela el comportamiento del editor:

TAD String **es** Secu(Char)

TAD Editor

observadores básicos

#pestañas	: ed $\longrightarrow$ nat		
seleccionada?	: ed e $\times$ nat p $\longrightarrow$ bool		$\{p < \text{pestañas}(e)\}$
texto	: ed e $\times$ nat p $\longrightarrow$ string		$\{p < \text{pestañas}(e)\}$
posiciónCursor	: ed $\longrightarrow$ nat		

generadores

abrir	: $\longrightarrow$ ed	
nuevaPestaña	: ed $\longrightarrow$ ed	
seleccionarPestaña	: ed e $\times$ nat p $\longrightarrow$ ed	$\{p < \text{pestañas}(e)\}$
moverCursor$\Rightarrow$	: ed $\longrightarrow$ ed	$\{\text{puedeMoverCursor}\Rightarrow?(e)\}$
moverCursor$\Leftarrow$	: ed $\longrightarrow$ ed	$\{\text{puedeMoverCursor}\Leftarrow?(e)\}$
insertarLetra	: ed $\times$ char $\longrightarrow$ ed	

otras operaciones

puedeMoverCursor$\Rightarrow$?	: ed $\longrightarrow$ bool	
puedeMoverCursor$\Leftarrow$?	: ed $\longrightarrow$ bool	

axiomas

...

Fin TAD

La estructura de representación elegida es la siguiente:

Editor **se representa con** estr

donde estr es tupla $\langle$ *inactivasVacías*: conj(nat),
inactivasNoVacías: conj(tupla$\langle$ *nro*: nat, *contenido*: string $\rangle$),
seleccionada: nat,
anteriores: string,
posteriores: string $\rangle$

En esta estructura:

- *inactivasVacías* contiene los números de todas las pestañas cuyo texto sea el string vacío, **exceptuando** a la pestaña seleccionada (que nunca debe estar en este conjunto, ya sea que esté vacía o no).
- *inactivasNoVacías* contiene la información de todas las pestañas cuyo texto **no** sea el string vacío, **exceptuando** a la pestaña seleccionada (que nunca debe estar en este conjunto, ya sea que esté vacía o no). Cada pestaña inactiva tiene un número que la identifica y su correspondiente texto.
- *seleccionada* indica el número de la pestaña seleccionada.
- *anteriores* es la secuencia de letras anteriores del cursor. Ej. en "ab|cd" tendríamos *anteriores* = "ab".
- *posteriores* es la secuencia de letras posteriores al cursor. Ej. en "ab|cd" tendríamos *posteriores* = "cd".

Teniendo en cuenta lo descripto arriba se pide:

a) Escribir en castellano el invariante de representación.

b) Escribir formalmente el invariante de representación.

c) Escribir formalmente la función de abstracción.

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 1.2

En esta entrega buscaremos agregar la siguiente funcionalidad a SimCity para partidas colaborativas: dos partidas de SimCity deben poder unirse en una nueva. Esta acción deberá hacer concoordar los rios y construcciones de ambas ciudades. Para que el mecanismo de unión sea constructivo y no destructivo, se deberá restringir la posibilidad de unir dos SimCitys a los casos donde no se solapen ríos con construcciones. Además, para evitar la depresión de quienes juegan, tampoco deberán solaparse las construcciones de nivel máximo de cualquiera de las partidas con otras construcciones.

Con estas consideraciones, deberán tomar desiciones respecto de cómo resolver conflictos entre construcciones de ambos SimCitys que ocupen el mismo casillero (casa vs. casa, comercio vs. comercio o casa vs. comercio). Las desiciones deberán estar documentadas debidamente aclarando las consideraciones tomadas, tanto con un texto como en la axiomatización. No son aceptables especificaciones que ignoren el conflicto y, por ejemplo, liberen el casillero.

Para conocer la popularidad de la partida, además es necesario poder conocer la cantidad de uniones que la conforman. Finalmente, nos interesa conocer la antigüedad de una partida como la cantidad de turnos que sucedieron en la misma. En el caso de unirse dos partidas, se considera la mayor antiguedad como la antiguedad de la unión.

Entrega

La entrega consistirá de un único documento digital con el modelado en TADs del enunciado presentado. Se recomienda el uso de los paquetes de LaTeX de la cátedra para lograr una mejor visualización del informe.
El PDF debe estar agregado, commiteado y pusheado en su repositorio de trabajo grupal en el directorio `tpg2`.

4.2. Taller: Lista Doblemente Enlazada

En este taller se debe implementar una lista doblemente enlazada para `int`. El enunciado corresponde al del taller de listas enlazadas presentado en L05.

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 `g2/taller`. Lo importante es que dentro del último (`g2/taller`) puedan encontrarse los archivos fuente del taller: `Lista.h` y `Lista.hpp`.