Guía 4

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
22 de junio	23 de junio	24 de junio	25 de junio	26 de junio Quiz 2.1.1 Empezar 2.2	27 de junio	28 de junio
29 de junio Empezar 2.3	30 de junio	1 de julio Quiz 2.1.2	2 de julio	3 de julio Corrección 2.2 Corrección 2.3	4 de julio	5 de julio
6 de julio	7 de julio	8 de julio Entrega del ej. grupal	9 de julio	10 de julio	11 de julio Subida del ej. individual	12 de julio
13 de julio	14 de julio Entrega del ej. individual	15 de julio	16 de julio	17 de julio	18 de julio	19 de julio

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

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

disponibilidades establecidas anteriormente. En particular, *trataremos* de que haya docentes disponibles durante las siguientes franjas horarias:

- **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 — Entrega TP3 Grupal
    - 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
    - Mié 22/07 — L12: Sorting (ex taller de sorting)[2] ..... **Presentación del recuperatorio**
    - Vie 24/07 — Consultas
    - Dom 26/07 — **Entrega del recuperatorio**
16. **Semana 16**
    - Lun 27/07 — Consultas
    - 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 T09: Sorting

- Exhibir arreglos de entrada con los que se alcance el mejor caso de *selection sort* e *insertion sort*, que son $\Theta(n)$ en mejor caso. Exhibir arreglos de entrada con los que se alcance el peor caso de *quicksort*, que es $\Theta(n^2)$ en peor caso, suponiendo que se elige como pivote siempre el primer elemento del arreglo.
- Sabemos que *mergesort* y *heapsort* son $O(n \log n)$ en peor caso y *quicksort* es $O(n \log n)$ en caso promedio, si se trabaja sobre arreglos. ¿Cómo se verían afectadas las complejidades si se cambian los arreglos por listas enlazadas?
- Podemos ordenar una secuencia de 0s y 1s usando *counting sort*. Por ejemplo, dada la secuencia 01110110, podemos notar que hay tres 0s y cinco 1s, para producir a continuación 00011111 como salida. En general, el costo de ordenar con este método es $O(n)$ en peor caso. ¿Por qué esto no contradice la cota inferior de $\Omega(n \log n)$ en peor caso vista en la teórica?
- ¿Cuáles de los algoritmos de ordenamiento conocidos se pueden hacer *in-place*, es decir, sobreescribiendo el arreglo de entrada y sin usar memoria adicional (salvo variables locales)?
- Recordar que un algoritmo de ordenamiento se dice *estable* si los elementos que "empatan" se ubican en la salida en el mismo orden en el que se encontraban en la entrada. ¿Cuáles de los algoritmos de ordenamiento conocidos se pueden hacer estables?

2.1.2. Clase T10: Divide & Conquer

- Completar la siguiente tabla para los siguientes algoritmos que reciben como entrada un arreglo de tamaño n y proceden haciendo Divide & Conquer.

	Número de llamados recursivos	Costo de dividir	Costo de combinar	Costo total
Búsqueda binaria	1	...	...	...
Mergesort	...	$O(1)$	...	...
Quicksort	...	...	$O(1)$	...

-

2.2. Ordenamiento

Ejercicio 1:
Se tiene un arreglo de n números naturales que se quiere ordenar por frecuencia, y en caso de igual frecuencia, por su valor. Por ejemplo, a partir del arreglo [1, 3, 1, 7, 2, 7, 1, 7, 3] se quiere obtener [1, 1, 1, 7, 7, 7, 3, 3, 2]. Describa un algoritmo que realice el ordenamiento descrito, utilizando las estructuras de datos intermedias que considere necesarias. Calcule el orden de complejidad temporal del algoritmo propuesto.

Ejercicio 2:
Se desea ordenar los datos generados por un sensor industrial que monitorea la presencia de una sustancia en

un proceso químico. Cada una de estas mediciones es un número entero positivo. Dada la naturaleza del proceso se sabe que, dada una secuencia de n mediciones, a lo sumo $\lfloor \sqrt{n} \rfloor$ valores están fuera del rango $[20, 40]$.
Proponer un algoritmo $O(n)$ que permita ordenar ascendentemente una secuencia de mediciones y justificar la complejidad del algoritmo propuesto.

Ejercicio 3:
Se tiene un arreglo $A[1 \dots n]$ de T, donde T son tuplas $\langle c_1 : nat \times c_2 : string[\ell] \rangle$ y los $string[\ell]$ son *strings* de longitud máxima ℓ. Si bien la comparación de dos nat toma $O(1)$, la comparación de dos $string[\ell]$ toma $O(\ell)$. Se desea ordenar A en base a la segunda componente y luego la primera.

1. Escriba un algoritmo que tenga complejidad temporal $O(n\ell + n\log(n))$ en el peor caso. Justifique la complejidad de su algoritmo.
2. Suponiendo que los naturales de la primer componente están acotados, adapte su algoritmo para que tenga complejidad temporal $O(n\ell)$ en el peor caso. Justifique la complejidad de su algoritmo.

Ejercicio 4:
Se tiene un arreglo de enteros no repetidos $A[1..n]$, tal que se sabe que para todo i hay a lo sumo i elementos mas chicos que $A[i]$ en todo el arreglo. Dar un algoritmo que ordene el arreglo en $O(n)$.

Ejercicio 5:

1. Dar un algoritmo que ordene un arreglo de n enteros positivos menores que n en tiempo $O(n)$.
2. Dar un algoritmo que ordene un arreglo de n enteros positivos menores que n^2 en tiempo $O(n)$. Pista: Usar varias llamadas al ítem anterior.
3. Dar un algoritmo que ordene un arreglo de n enteros positivos menores que n^k para una constante arbitraria pero fija k.
4. ¿Qué complejidad se obtiene si se generaliza la idea para arreglos de enteros no acotados?

2.3. Dividir y conquistar

Ejercicio 1:
Escriba un algoritmo con dividir y conquistar que determine si un arreglo de tamaño potencia de 2 es *más a la izquierda*, donde "más a la izquierda" significa que:

- La suma de los elementos de la mitad izquierda superan los de la mitad derecha.
- Cada una de las mitades es a su vez "más a la izquierda".

Por ejemplo, el arreglo [8, 6, 7, 4, 5, 1, 3, 2] es "más a la izquierda", pero [8, 4, 7, 6, 5, 1, 3, 2] no lo es.
Intente que su solución aproveche la técnica de modo que complejidad del algoritmo sea estrictamente menor a $O(n^2)$.

Ejercicio 2:
Suponga que se tiene un método *potencia* que, dada un matriz cuadrada A de orden 4×4 y un número n, computa la matriz A^n. Dada una matriz cuadrada A de orden 4×4 y un número natural n que es potencia de 2 (i.e., $n = 2^k$ para algun $k \geq 1$), desarrollar, utilizando la técnica de dividir y conquistar y el método *potencia*, un algoritmo que permita calcular

$$A^1 + A^2 + \ldots + A^n.$$

Calcule el número de veces que el algoritmo propuesto aplica el método *potencia*. Si no es estrictamente menor que $O(n)$, resuelva el ejercicio nuevamente.

Ejercicio 3:
Dado un árbol binario cualquiera, diseñar un algoritmo de dividir y conquistar que devuelva la máxima distancia entre dos nodos (es decir, máxima cantidad de ejes a atravesar). El algoritmo no debe hacer recorridos innecesarios sobre el árbol.

Ejercicio 4:
La cantidad de parejas en desorden de un arreglo $A[1 \dots n]$ es la cantidad de parejas de posiciones $1 \leq i <$

$j \leq n$ tales que $A[i] > A[j]$. Dar un algoritmo que calcule la cantidad de parejas en desorden de un arreglo y cuya complejidad temporal sea estrictamente mejor que $O(n^2)$ en el peor caso. **Hint:** Considerar hacer una modificación de un algoritmo de sorting.

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.

Nota (★): en los ejercicios que siguen, trabajaremos con conjuntos de naturales, dados por un módulo ConjuntoDeNaturales. Para recorrer los conjuntos, se puede suponer que el módulo ConjuntoDeNaturales cuenta con un iterador que visita los elementos en orden estrictamente creciente, con tiempo constante de creación, acceso y avance. Por ejemplo, si C es un conjunto de cardinal n, el siguiente `for` muestra los elementos del conjunto C ordenados de menor a mayor y tiene costo lineal en peor caso, es decir, cuesta $O(n)$:

```
for x in C {
    mostrar(x)
}
```

Se puede escribir más explícitamente con un `while`:

```
it := CrearIt(C)              // CrearIt       cuesta O(1)
while HaySiguiente?(it) {     // HaySiguiente? cuesta O(1)
    mostrar(Siguiente(it))    // Siguiente     cuesta O(1)
    Avanzar(it)               // Avanzar       cuesta O(1)
}
```

No se pueden asumir otras operaciones en la interfaz del módulo ConjuntoDeNaturales. En particular, dicho módulo **no** provee una operación para calcular la intersección.

3.1. Ordenamiento

Ejercicio 1: ordenamiento

Se tiene un arreglo A de N conjuntos $A[1], \ldots, A[N]$. El cardinal de cada conjunto es **a lo sumo** K, es decir $\#(A[i]) \leq K$ para todo $1 \leq i \leq N$. Se desea ordenar el arreglo A para obtener como resultado un arreglo de conjuntos $B[1], \ldots, B[N]$ de tal modo que se cumpla la siguiente condición:

$$B[i] \subseteq B[j] \;\Rightarrow\; i \leq j \qquad \text{para todo } i, j \text{ en el rango } 1..N$$

es decir, si un conjunto está incluido en otro, debe aparecer antes en el arreglo ordenado.

Notar que el problema no tiene única solución. Por ejemplo si A es el siguiente arreglo de conjuntos:

$$A \;=\; [\{1,2,3\}, \{1,2\}, \{1,3\}, \{1,2,4\}]$$

Una posible respuesta podría ser el siguiente arreglo B:

$$B \;=\; [\{1,3\}, \{1,2\}, \{1,2,3\}, \{1,2,4\}]$$

y otra posible respuesta podría ser el siguiente arreglo B':

$$B' \;=\; [\{1,2\}, \{1,2,4\}, \{1,3\}, \{1,2,3\}]$$

Proponer un algoritmo para resolver este problema con complejidad temporal en peor caso $O(NK)$, teniendo en cuenta que no hay conjuntos repetidos. Justificar la complejidad temporal obtenida. Para manipular los conjuntos tener en cuenta la **nota** (★) de arriba.

3.2. Dividir y conquistar

Ejercicio 2: dividir y conquistar

Se tiene un arreglo C de N conjuntos $C[1], \ldots, C[N]$. El cardinal de cada conjunto es **exactamente** M, es decir $\#(C[i]) = M$ para todo $1 \leq i \leq N$.

1. Proponer un algoritmo para determinar si los conjuntos $C[1], \ldots, C[N]$ son **disjuntos dos a dos**, es decir si para todo $i \neq j$ en el rango $1..N$ se tiene que $C[i] \cap C[j] = \emptyset$. La complejidad temporal en peor caso del algoritmo debe ser $O(MN \log N)$. Justificar la complejidad temporal obtenida.

2. (**optativo**) Suponiendo que los elementos de todos los conjuntos están en el rango comprendido entre 1 y el producto $M \cdot N$, proponer un algoritmo para resolver el mismo problema con complejidad temporal en peor caso $O(MN)$. Notar que este algoritmo sirve para determinar si los conjuntos $C[1], \ldots, C[N]$ constituyen una **partición** del conjunto $\{1, 2, \ldots, MN - 1, MN\}$.
   Justificar la complejidad temporal obtenida.

Para manipular los conjuntos tener en cuenta la **nota** (★) de arriba.

Aclaración: el ejercicio que define la aprobación es el 2.1, el que está marcado como optativo define si, de estar aprobado el anterior, se sube la nota para que quede una P en lugar de una A.
De estar mal resuelto el 2.1, la nota final sería de una R o I independientemente del 2.2.

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 3

4.1.1. Enunciado

El objetivo de este TP es implementar en C++ todos los módulos correspondientes al diseño presentado en el TP2.
El código debería respetar el diseño propuesto en el TP 2 de la manera más fiel posible. Obviamente se permite y se espera que corrijan todos los potenciales *bugs* que puedan llegar a encontrar en el diseño así como adaptaciones requeridas durante la corrección del TP 2.
Las implementaciones deben cumplir con las complejidades definidas en su solución del TP 2, incluyendo las restricciones de complejidad establecidas por el enunciado del mismo.

4.1.2. Código producido

La resolución debe tener un archivo `.h` y `.cpp` por cada módulo del TP 2. Los mismos han de ubicarse en el directorio `src`. En caso de que su solución del TP 2 implique la extensión de módulos existentes, consultar la mejor manera de llevar a cabo esta implementación.

4.1.3. Código de la cátedra y Tests

Como parte del enunciado, la cátedra provee un conjunto de tests que deberán ser pasados con éxito por la solución presentada. El código de los mismos se encuentra en la página de la materia (link). El descargable posee la misma estructura que los talleres, con un directorio `src` para los archivos fuente y un directorio `tests` para el código de los tests. Para la entrega pueden incorporar ambas carpetas junto con el `CMakeLists.txt` a su repositorio de entregas grupales en la carpeta `tpg4`. Luego, pueden agregar el código producido para la solución del TP en la carpeta `src`.
En la carpeta `src` del descargable se encuentran los archivos `aed2_Mapa.{h,cpp}` y `aed2_SimCity.{h,cpp}`. Estos archivos tienen la interfaz para los módulos Mapa y SimCity que se utilizan en los tests. Deben completar los mismos agregando instancias de sus implementaciones en las partes privadas de cada clase e implementando sus métodos de forma que utilicen la interfaz provista por sus módulos. Además, la clase `aed2_Mapa` cuenta con dos funciones para convertir desde su clase a la de la cátedra y viceversa:

- `aed2_Mapa::aed2_Mapa(Mapa m);`
- `Mapa aed2_Mapa::mapa();`

De la misma forma, función `SimCity aed2_SimCity::simCity()` permite obtener su implementación de SimCity a partir de la clase de la cátedra.
Finalmente, el código provisto por la cátedra posee un archivo auxiliar `src/Tipos.h` que define algunos renombres de tipos.
La adaptación de la interfaz de sus módulos a los requeridos en `aed2_SimCity` y `aed2_Mapa` pueden conllevar operaciones con un costo no inmediato (e.g.: copiar un conjunto a una lista, recorrer un diccionario). Los requisitos de complejidad a cumplir son solo para las funciones de la interfaz de los módulos. Los costos asociados a la traducción de su interfaz a la nuestra no tienen restricciones.

4.1.4. Módulos básicos

Pueden utilizar las siguientes clases de la STL de C++ para los respectivos módulos básicos:

Módulo	Clase
Lista Enlazada	`std::list`
Pila	`std::stack`
Cola	`std::queue`
Vector	`std::vector`
Diccionario Lineal	`std::map`
Conjunto Lineal	`std::set`

4.1.5. Entrega

La entrega consistirá en publicar un commit con los archivos fuente generados en su repositorio de entregas grupales, dentro del directorio `tpga4`.
El commit debe estar sincronizado con el repositorio remoto en `gitlab.com`.
La fecha de entrega es hasta el Miércoles 8 de Julio a las 23:59. La devolución del mismo será para el Miércoles 15 de Julio. La fecha de reentrega es hasta el Domingo 26 de Julio a las 23:59.

4.2. Taller: Diccionario sobre Trie

En este taller se debe implementar un diccionario sobre Trie para un tipo paramétrico `T`. El enunciado corresponde al del taller de diccionario sobre Trie en L08.

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 `g4/taller`. Lo importante es que dentro del último (`g4/taller`) puedan encontrarse los archivos fuente del taller: `string_map.h` y `string_map.hpp`.