Trabajo práctico 1: Diseño

# JuegoDePalabras

## Normativa

**Límite de entrega**: Viernes 21 de octubre *hasta las 23:59 hs.* Ver "Instructivo para entregas" en el sitio Web de la materia.

**Normas de entrega**: Ver "Información sobre la cursada" en el sitio Web de la materia. (`http://campus.exactas.uba.ar`)

## 1. Enunciado

Este TP consiste en diseñar módulos para implementar un servidor para el *juego de palabras*, conforme a la especificación ya presentada en el curso y disponible en el sitio de la materia.

Se deben proveer las siguientes operaciones, con las complejidades temporales en **peor caso** indicadas. Usamos las siguientes variables:

- $N$ — tamaño del tablero.
- $K$ — cantidad de jugadores.
- $|\Sigma|$ — cantidad de letras en el alfabeto.
- $F$ — cantidad de fichas por jugador.
- $L_{\text{máx}}$ — longitud de la palabra legítima más larga definida por la variante del juego de la que se trate.

### 1.1. Juego

1. Iniciar un nuevo juego. $O(N^2 + |\Sigma|K + FK)$.
2. Determinar si una jugada es válida. $O(L_{\text{máx}}^2)$, donde la jugada podría tener $m$ fichas. Observar que la cota no depende de $m$ ni de $N$.
3. Ubicar un conjunto de fichas en el juego. $O(m)$, donde $m$ es el número de fichas que se ubican.
4. Conocer la información general del juego:
   - Obtener información sobre la variante del juego. $O(1)$.
   - Obtener el número del jugador a quien le toca jugar actualmente. $O(1)$.
   - Obtener el puntaje de un jugador. $O(1 + m \cdot L_{\text{máx}})$, donde $m$ es la cantidad de fichas que ubicó el jugador desde la última vez que se invocó a esta operación.
   - Obtener el contenido del tablero en una coordenada $(i, j)$. $O(1)$.
   - Dada una cierta letra $x$ del alfabeto, conocer cuántas fichas tiene un jugador de dicha letra. $O(1)$.

### 1.2. Servidor

1. Inicializar un servidor. $O(N^2 + |\Sigma|K + FK)$.
2. Conectar un cliente a un servidor. $O(1)$.
3. Consultar la cola de notificaciones de un cliente (lo cual vacía dicha cola). $O(n)$, donde $n$ es la cantidad de mensajes en la cola de dicho cliente.
4. Recibir un mensaje de un cliente. No se impone una cota explícita. La complejidad no debe depender de $N$ ni de $K$. Puede depender de $|\Sigma|$, $F$, $L_{\text{máx}}$ y del número de fichas que el jugador pretenda ubicar al enviar este mensaje.
5. Conocer la información general del servidor:
   5.1. Obtener el número de clientes esperados. $O(1)$.
   5.2. Obtener el número de clientes conectados. $O(1)$.
   5.3. Obtener el juego que se está jugando en el servidor. $O(1)$.

## 2. Documentación a entregar

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

1. **Interfaz.**
   - 1.1. *Tipo abstracto ("se explica con ...").* Género (TAD) que sirve para explicar las instancias del módulo, escrito en el lenguaje de especificación formal de la materia. Pueden utilizar la especificación que se incluye en el apéndice.
   - 1.2. *Signatura.* Listado de todas las funciones públicas que provee el módulo.
   - 1.3. *Contrato.* Precondición y postcondición de todas las funciones públicas. Las precondiciones de las funciones de la interfaz deben estar expresadas formalmente en lógica de primer orden.[1]
   - 1.4. *Complejidades.* Complejidades de todas las funciones públicas, cuando corresponda.
   - 1.5. *Aspectos de* aliasing. De ser necesario, aclarar cuáles son los parámetros y resultados de los métodos que se pasan por copia y cuáles por referencia, y si hay *aliasing* entre algunas de las estructuras.
2. **Implementación.**
   - 2.1. *Representación ("se representa con ...").* Módulo con el que se representan las instancias del módulo actual.
   - 2.2. *Invariante de representación.* Puede estar expresado formalmente en lógica de primer orden o en lenguaje natural.
   - 2.3. *Función de abstracción.* Puede estar expresada formalmente en lógica de primer orden o en lenguaje natural.
   - 2.4. *Algoritmos.* Pueden estar expresados en pseudocódigo, usando si es necesario la notación del lenguaje de módulos de la materia o notación tipo C++. Las pre y postcondiciones de las funciones auxiliares pueden estar expresadas en lenguaje natural (no es necesario que sean formales). Indicar de qué manera los algoritmos cumplen con el contrato declarado en la interfaz y con las complejidades pedidas. No se espera una demostración formal, pero sí una justificación adecuada.
3. **Servicios usados.** Módulos que se utilizan, detallando las complejidades, *aliasing* y otros aspectos que dichos módulos deben proveer para que el módulo actual pueda cumplir con su interfaz.

**Sobre el uso de lenguaje natural y formal.**

Las precondiciones y poscondiciones de las funciones auxiliares, pueden estar expresados en lenguaje natural. Los algoritmos pueden estar expresados en pseudocódigo. Se recomienda aplicar el sentido común para priorizar la **claridad** y **legibilidad**. Por ejemplo:

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

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