

1. Condiciones del Trabajo Practico Obligatorio

1.1. Condiciones de aprobación

- Tener al menos 5 de los ejercicios en condiciones de aprobación.
- El código debe estar en GitHub.

El grupo debe indicar quién realizó cada ejercicio para la evaluación individual.

1.2. Condiciones de entrega

- El nombre del repositorio será *TPO_P2_DT_grupo_XX*, donde *DT* es día y turno. Por ejemplo: *LN* es Lunes Noche. *XX* es el número de grupo. En el caso de hacer el TPO solo, el nombre del repoo debe ser *TPO_P2_DT_LU*.
- Deben ponerme como colaborador, el usuario es *nicolas-monzon*.
- Pueden crear los branches que necesiten, pero el branch que voy a revisar es el branch *develop*. Recomendando, aunque no es necesario, usar la metodología GitFlow.

1.3. Aclaraciones

- Las buenas prácticas del código suma puntos.
- Usar recursividad en ejercicios que no lo piden, suma puntos.
- Está permitido usar cualquier LLM.
- Solo serán considerados los commits incluidos hasta la fecha de entrega inclusive (ver cronograma).

2. Ejercicios

2.1. Ejercicio 1: Matrices

Dada una cola de n pilas, y cada pila de n elementos, esta tomará un aspecto matricial. Llamaremos a estructura **QueueOfStacks**. Por ejemplo, una posible matriz de $n \times n$ para $n = 3$ es:

$$\begin{array}{|c|c|c|} \hline 1 & 2 & 3 \\ \hline 4 & 5 & 6 \\ \hline 7 & 8 & 9 \\ \hline \end{array} \rightarrow$$

1. Desarrolle una función que reciba una instancia de **QueueOfStacks**, y calcule su traza.
2. Desarrolle una función que reciba una instancia de **QueueOfStacks**, y devuelva su traspuesta.
3. Desarrolle una función que reciba dos instancias de **QueueOfStacks** y devuelva una nueva instancia de **QueueOfStacks** que represente la suma matricial.

Calcule la complejidad computacional de los algoritmos anteriores.

2.2. Ejercicio 2: Programación genérica

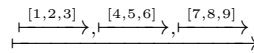
Desarrolle **Stack**, **Queue**, **QueueWithPriority** y **Set** con programación genérica. Luego:

1. Desarrolle una función que reciba un **Stack** genérico e invierta sus elementos.
2. Desarrolle una función que copie un **Set** genérico.

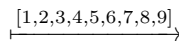
2.3. Ejercicio 3: Propagación

Desarrolle una `QueueOfQueue` y agregue los siguientes métodos a su interfaz e implementación:

1. **concatenate**: Recibe n instancias de `QueueOfQueue` y genera una nueva instancia de `QueueOfQueue` con todos los elementos de las instancias anteriores manteniendo el orden en que se leyeron los valores de estas instancias.
2. **flat**: Crear una instancia de `Queue` a partir de la instancia de `QueueOfQueue` con los mismos elementos. Ejemplo:



Se convierte en



3. **reverseWithDepth**: Inverte la instancia de `QueueOfQueue` pero también cada `Queue` dentro de esta.

2.4. Ejercicio 4: Flexibilidad de las estructuras

2.4.1. Builder

Modifique la estructura `Set` y la estructura `Stack` para que tengan métodos `add` y `addAll` y además de un constructor, también se tenga el patrón *Builder* para poder instanciar:

```
Set set = Set.builder().add(1).add(2).add(3).build();
Set set2 = Set.builder().add(0).addAll(set).add(4).build();
```

2.4.2. Constructor

Modifique la estructura `Queue` para que tenga un constructor con cantidad de parámetros variables. Estos deben actuar como los valores por defecto de la estructura.

2.5. Ejercicio 5: Algoritmos

Desarrolle uno de los siguientes algoritmos y calcule su complejidad computacional:

1. Dada una pila de elementos desordenados, generar una nueva pila sin elementos repetidos y ordenados.
2. Cree un diccionario que tenga como clave cada letra del alfabeto español, y como valor cada frecuencia asociada. Cree una String en español de mas de 500 caracteres, y que tenga aplicado un cifrado César. Desarrolle un algoritmo que descifre la String en base a la frecuencia de sus caracteres.
3. Modifique el algoritmo de *paréntesis balanceados* para que, al tener uno de estos paréntesis comillas (por ejemplo, '(' o '}', entonces lo ignore.

Implemente el **Método de Montecarlo** de la siguiente forma.

1. Cree un TDA que represente una coordenada en el plano.
2. Defina una región cuadrada en el primer cuadrante.
3. Cree un TDA `Montecarlo` que tenga un método que reciba una coordenada y la almacene dentro de su estructura solo si esta presente en la región delimitada por el cuadrado.
4. Cree un conjunto de coordenadas y agregue elementos hasta que su cardinal sea 1000, de forma aleatoria.
5. Llene la estructura `Montecarlo` con estos elementos y utilícelos para aproximar π .

2.6. Ejercicio 6: Modificación a los TDAs

A partir de los TDAs de la cursada, cree los siguientes TDAs:

1. Pila dinámica con capacidad limitada. Debe recibir la capacidad máxima por constructor.
2. Superconjunto (o conjunto universal). A parte de las operaciones de un conjunto normal, debe contar con un método que reciba una instancia del `Set` y devuelva `true` si es subconjunto. Debe contar con un segundo método que permite calcular el conjunto complemento del recibido por parámetro. Debe ser precondition que no se puede calcular el complemento de un conjunto que no es subconjunto del superconjunto.
3. Cola dinámica cíclica doblemente enlazada. Los métodos que posee deben usar esta propiedad para disminuir la complejidad computacional.
4. Conjunto con repetidos. Es un conjunto de duplas, donde cada elemento tiene asociada una cantidad.

2.7. Ejercicio 7: Grafos

Desarrollar una función que reciba un grafo dirigido, y devuelva un conjunto de grafos dirigidos tales que, cada uno de estos grafos, es fuertemente conexo. Explique que sucede si el grafo de entrada no es dirigido. *Tip: Se puede hacer un conjunto con todos los subgrafos posibles y luego filtrar aquellos que son fuertemente conexos.*

3. Restricciones sobre el desarrollo

- Se deberán usar las estructuras desarrolladas en clase, de forma obligatoria, en los ejercicios 1, 3 y 7.
- Cada ejercicio deberá tener su propio *package*.
- Cada package no deberá contener contenido no relacionado al ejercicio.