Programación II, Algoritmos y Estructuras de Datos II

Examen Final Previo - 05/07/2025

Docente: Monzón, Nicolás Alberto .....

Apellido y Nombre: .....
L.U.: ..... **Firma:** .....

La firma de este documento implica la aceptación de las condiciones de aprobación abajo numeradas. Favor de verificar que cumpla con todas, en particular con los nombres de carpetas.

Requisitos para aprobar el examen:

1. Por actos de deshonestidad académica será sancionado.
2. Para poder aprobar este examen Ud. deberá estar administrativamente en condiciones de poder rendir el examen. En caso de no estarlo y rendir el examen, este no va a ser corregido y quedará anulado.
3. Deberá comprimir la carpeta `src` de su proyecto. El archivo comprimido deberá tener como nombre su APELLIDO y NOMBRE en ese orden. Deberá colocarlo en la unidad compartida de la computadora de UADE Labs y verificar con el docente que pudo ser recuperado.
4. Deberá desarrollar un set de prueba para demostrar que funciona su código. No es necesario escribir test unitarios, pero sí alguna prueba en el método `main` del proyecto.
5. Deberá escribir la estrategia de cada ejercicio y métodos que haga, y colocarla dentro de la carpeta `strategies` en formato `txt`, `MD` o `pdf`, cualquier otro formato no va a ser corregido.
6. En la firma de cada método escribir las precondiciones.
7. Solo podrá utilizar técnicas vistas en este curso. Esta prohibido el uso de librerías y genéricos, incluyendo estructuras que vienen con la JRE. No respetar este punto es suficiente para desaprobar el examen.
8. Este examen consta de 6 paginas.

9. El examen deberá ser entregado en el horario estipulado.

10. CONDICIONES PARA APROBAR: obtener al menos 80 puntos de los 140 puntos y cumplir con todos los puntos anteriores. En caso de no cumplir con alguno de los puntos anteriores, el examen queda desaprobado sin excepción.

Si el examen utiliza predicados, use la siguiente definición:

$$\mathcal{P}(n) = \text{Su LU, módulo 3, es } n$$

Ejercicios

1. **[40 p]** Relación de equivalencia. Dado un grafo, una *relación de equivalencia* se define como una relación reflexiva, simétrica y transitiva. Un grafo es reflexivo bajo la relación $\rightarrow$, si para todo vértive v, se tiene que $v \rightarrow v$. De la misma forma, un grafo es simétrico si para todo vértive v_1 y para todo vértice v_2, si $v_1 \rightarrow v_2$, entonces se cumple que $v_2 \rightarrow v_1$. Por último, un grafo es transitivo, si para todo vértice v_1, v_2 y v_3, si se cumple que $v_1 \rightarrow v_2$ y $v_2 \rightarrow v_3$, entonces se cumple que $v_1 \rightarrow v_3$.

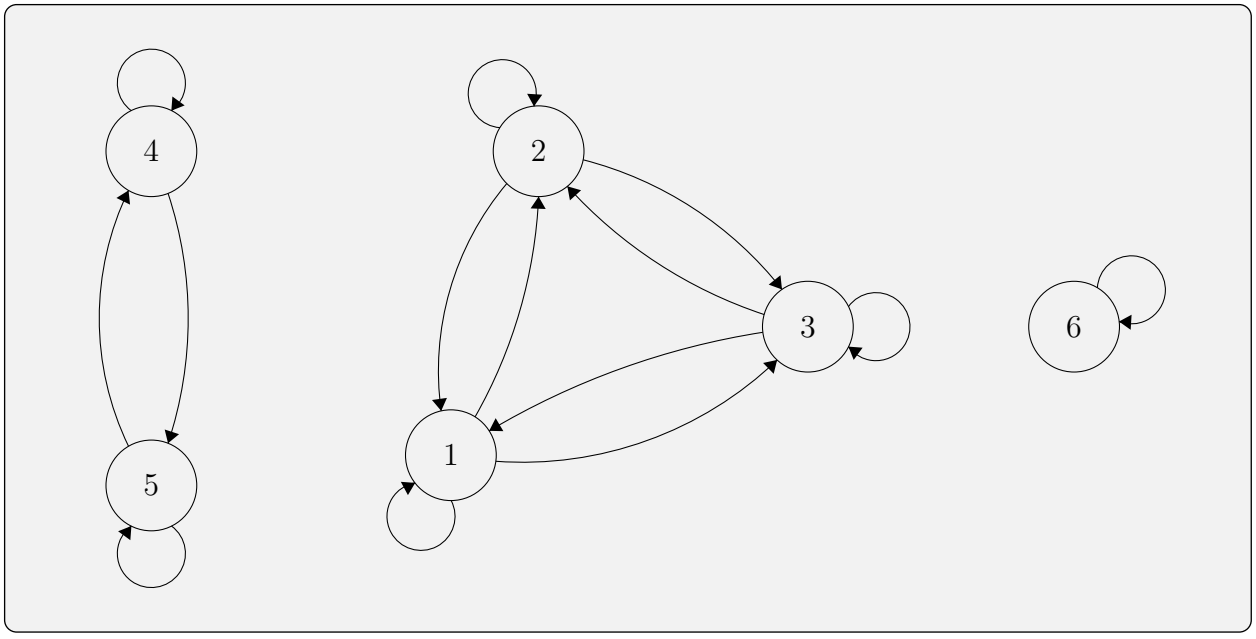

Figura 1: Ejemplo de un grafo que representa una relación de equivalencia.

Por ejemplo, en la Figura 1, dado que $4 \rightarrow 5$, tenemos que pedir que $5 \rightarrow 4$ (lo cual es cierto) para que el grafo sea simétrico,

   a) Describa brevemente (en no más de cinco líneas) la estrategia para determinar, dado un grafo G de n vértices, si G representa una clase de equivalencia.

   b) Escriba un método en Java que realice la estrategia del punto anterior.

   c) ¿Cómo se puede usar la información obtenida en el método anterior para saber si no existe un camino hamiltoniano?

2. **[20 p]** Tenemos cuatro números α, β, γ y δ, tales que $\alpha < \beta < \gamma < \delta$, ¿Cuántos árboles binarios distintos pueden formarse? ¿Cuántos podrían ser SBT? Dibujar los árboles AVL que se pueden formar.

3. **[40 p]** Se tiene un objeto de tipo `BinaryTree`. Tenemos como precondición que existe una rama más larga. Queremos un método que, dado un número que está en el árbol, nos devuelva `true` si pertenece a la rama más larga.

   a) Describa brevemente (en no más de cinco líneas) la estrategia que utilizará para resolver este ejercicio. El algoritmo debe ser recursivo.

   b) Escriba el método correspondiente en Java.

   c) Pruebe su algoritmo con el árbol de la Figura 2 para los valores **4** y **3** (debe devolver `false`) y luego para los valores **2** y **6** (debe devolver `true`).

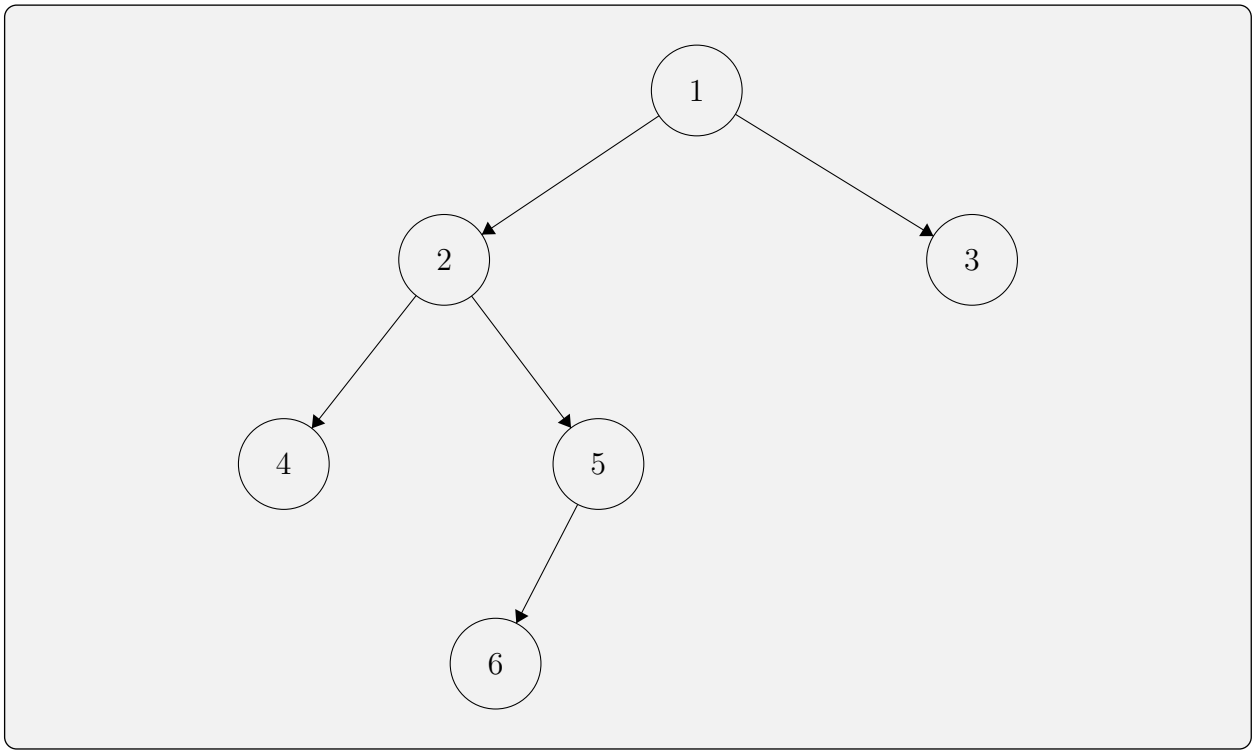

Figura 2: Ejemplo de árbol binario con una rama más larga que las demás

4. **[40 p]** Ejercicio de la guía. A partir del TDA `Set`, determinar si los elementos de una `Stack` s son los mismos que los de una `Queue` q. No interesa el orden ni si están repetidos o no.

Apéndice

Se entregará un archivo `zip` con el código necesario. Si una implementación de TDA no es proporcionada, pero su definición (interfaz) sí, asuma que debe crear el TDA por usted mismo.

A continuación se detallan los TDAs involucrados:

```
public interface BinaryTree {

    int getRoot();
    BinaryTree getLeft();
    BinaryTree getRight();
    void removeLeft();
    void removeRight();
    void addLeft(int a);
    void addRight(int a);

}
```

```
public interface Graph {

    void addNode(int node);
    void removeNode(int node);
    Set getNodes();
    void addEdge(int from, int to, int weight);
    void removeEdge(int from, int to);
    boolean edgeExists(int from, int to);
    int weight(int from, int to);

}
```

```
public interface Queue {

    void add(int a);
    void remove();
    boolean isEmpty();
    int getFirst();
```

```
}
```

```
public interface SearchBinaryTree {

    int getRoot();
    SearchBinaryTree getLeft();
    SearchBinaryTree getRight();
    void removeLeft();
    void removeRight();
    void add(int a);

}
```

```
public interface Set {

    void add(int a);
    void remove(int a);
    int choose();
    boolean isEmpty();

}
```

```
public interface Stack {

    void add(int a);
    void remove();
    int getTop();
    boolean isEmpty();

}
```