Trabajo Práctico Nro.1 - Tipos Abstractos de Datos y Complejidad

Fecha de Presentación: 03/09/2015
Fecha de Entrega: 17/09/2015

Ejercicio1) La odisea[1]

La Odisea es un poema épico atribuido al poeta griego Homero. Narra la vuelta a casa, tras la Guerra de Troya, del héroe griego Odiseo .

Además de haber estado diez años fuera luchando, Odiseo tarda otros diez años en regresar a la isla de Ítaca, donde poseía el título de rey, período durante el cual su hijo Telémaco y su esposa Penélope han de tolerar en su palacio a los pretendientes que buscan desposarla (pues ya creían muerto a Odiseo), al mismo tiempo que consumen los bienes de la familia.

La mejor arma de Odiseo es su astucia. Gracias a su inteligencia —además de la ayuda provista por Palas Atenea, hija de Zeus Cronida— es capaz de escapar de los continuos problemas a los que ha de enfrentarse por designio de los dioses. Para esto, planea diversas artimañas, bien sean físicas —como pueden ser disfraces— o con audaces y engañosos discursos de los que se vale para conseguir sus objetivos.

Se desea modelar el regreso de Odiseo desde Troya hasta Ítaca, pasando por n ciudades intermedias.

a) Odiseo desea llegar a Ítaca recorriendo el camino más corto que sea posible. Para ello sabe que primero tiene que calcular todos los caminos posibles y así, considerar el camino más corto.

Lamentablemente para n ciudades existen O(n!) caminos posibles
Se desea ayudar a Odiseo a calcular el camino mínimo que le permita volver de Troya a Ítaca con una estrategia que no explore todos los caminos.

Para ello la catedra otorgara el TAD Mapa (que modela cualquier mapa).
Dentro de dicho TAD implementar el método

ArrayList<Ciudad> caminoMinimoProbable(Nodo origen, Nodo destino)

Que devuelve el camino más corto probable asumiendo la siguiente estrategia
-Cuando estoy parado en un Nodo k, ir al Nodo más cercano (k+1) hasta encontrar el Nodo destino.
-Marcar los nodos que ya fueron visitados durante el proceso.
-Asumir que se conocen las distancias de todos los nodos entre sí.

b) Calcular la complejidad de caminoMinimoProbable

-Nombrar las variables que definen en tamaño del problema

-Expresar la complejidad en dichas variables

[1] https://es.wikipedia.org/wiki/Odisea

Test1

```
public class Test {

	public static void main(String[] args) {
		Mapa m1 = new Mapa(3);

		m1.asignarNombre(0, "Troya");
		m1.asignarNombre(1, "Otra");
		m1.asignarNombre(2, "Itaca");

		m1.agDist(0, 1, 10, true);
				// distancia desde 0 hasta 1 es igual a 10
		m1.agDist(0, 2, 10, true);
		m1.agDist(1, 2, 10, true);

		System.out.println(m1.dist(2, 1));
		System.out.println(m1.caminoMinimoProbable(0,0));


	}

}
Consola
10
[0 Troya]
```

Test2

```
public class Test {

	public static void main(String[] args) {
		Mapa m1 = new Mapa(4);

		m1.asignarNombre(0, "Troya");
		m1.asignarNombre(1, "Otra1");
		m1.asignarNombre(2, "Itaca");
		m1.asignarNombre(3, "Otra2");

		m1.agDist(0, 1, 10, true);
		m1.agDist(0, 2, 15, true);
		m1.agDist(0, 3, 10, true);
		m1.agDist(1, 2, 10, true);
		m1.agDist(1, 3, 15, true);
		m1.agDist(2, 3, 10, true);

		System.out.println(m1.caminoMinimoProbable(0,2));


	}

}
```

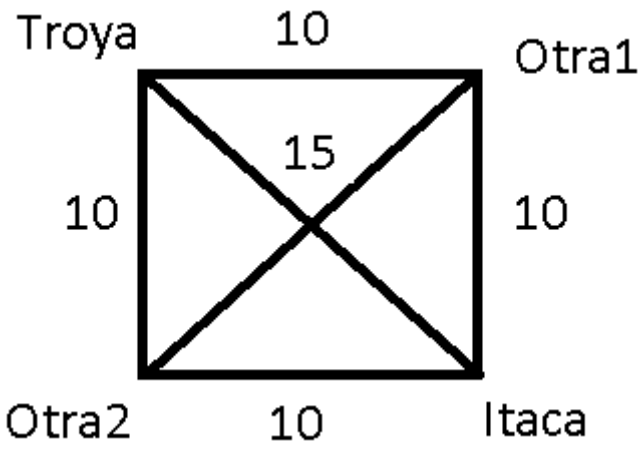


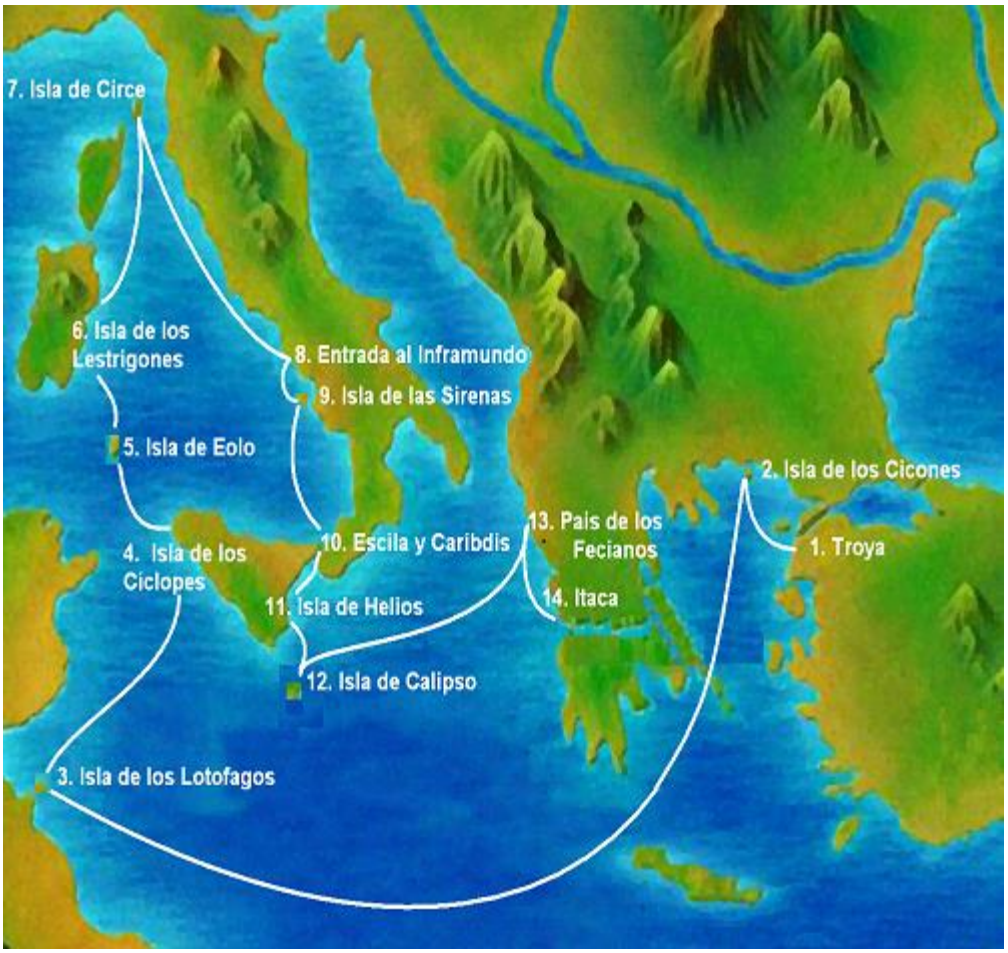
Grafico1: Distancias de Test2

Consola

```
[0 Troya, 1 Otra1, 2 Itaca]   //distancia 20
```

Test3

Modelar el viaje real de Odiseo asumiendo
-Que se conocen las distancias de todas las ciudades a todas las ciudades. Asumir valores aleatorios para dichas distancias



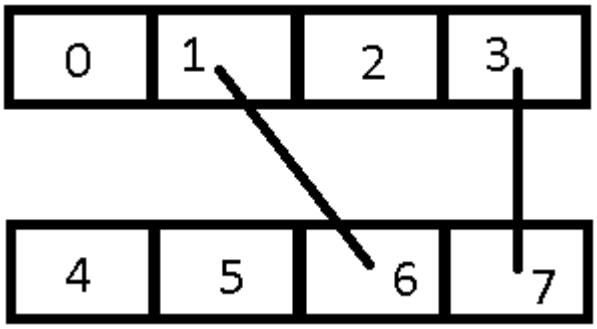
Comparar la distancia del viaje real, respecto del viaje calculado por el algoritmo.

Incluir ambas distancias en el informe

Ejercicio2) TAD Juego: El hormiguero

La hormiga sale de la posición 0 y se mueve por el hormiguero hasta llegar a la posición n
En el medio puede pasar por atajos los cuales está obligada a tomar aunque tenga que retroceder.

Hormiguero para n = 8 (hay 8 casillas)



Atajos: Desde 6 voy a 1 y desde 3 voy a 7, la casilla final.

Se desea modelar el hormiguero, pero con la cantidad de casilleros definida por el usuario.

Sistema de juego simplificado:

Se pide por única vez el tamaño del tablero(n), y los atajos
En cada turno la hormiga tira un dado (un número de 1 a 5).
Cuando la hormiga avanza, si cae un atajo, debe quedar en la posición final del atajo.
Si me paso de la posición final se considera que se terminó el juego.

Al final del juego devolver la cantidad de veces que se tuvo que tirar el dado para finalizar.
Si se tira más de 1000 veces terminar el juego.

Enunciado:

a) Diseñar e implementar el TAD Juego de “Hormiguero”

Es obligatorio que el TAD utilice al menos otro TAD como TAD soporte. Es decir, que Hormiguero no podrá estar implementado únicamente sobre los tipos primitivos de Java (ni java.util).

Interfaz:

```
Hormiguero (n)          //tamaño del hormiguero, donde n > 1
Int cantJugadas()       //devuelve el nro de jugadas hasta el momento
Void agregarAtajo (int desde, int hasta) //Agrega un atajo
void jugar()            // tira los dos dados
String ver()            // muestra el estado del tablero y de la hormiga
Int ultimaCasilla()     // casilla actual de la hormiga
```

Ejemplo del código principal:

```
Hormiguero m = new Hormiguero(8);         // Instancia: n == 8

m.agregarAtajo(6,1);//atajo desde 6 hasta 1
m.agregarCasilla(3,7);
while m.ultimaCasilla() != 8{       //como máximo hacerlo 1000 veces
      m.jugar();              // La hormiga "tira" el dado
      System.out.println(m.ver())     //Se  muestra  un  resumen     del
hormiguero
}
System.out.println(m.cantJugadas ());
```

Test1

Modelar la instancia del enunciado.

b) Escribir el invariante de representación del Hormiguero.
Describir que Hormigueros son instancias validas del Hormiguero que se quiere representar

Entrega y condiciones para aprobar el TP

Los grupos deberán ser de 1 o 2 personas a lo sumo.

Antes de la entrega deberán informar por mail los integrantes del grupo, a lo que se responderá con el nro. de grupo.

El día de la entrega(o antes) se enviara un mail por grupo con el subjet 2015C2TP1 seguido de los apellidos de los integrantes, por ejemplo "2015C2TP1 Perez - Gutierrez"
Y como adjunto, un archivo .zip con el código fuente y el informe.
El código fuente puede estar exportado como .**jar,** .java, .txt, .doc, .docx o .pdf
El informe puede estar en formato .doc, .docx o .**pdf**

Es importante entregar si es posible, antes de la fecha límite, de manera que puedan tener recomendaciones antes de la entrega final.

El testeo debe ser positivo, es decir:

El algoritmo debe funcionar bien para cualquier Mapa, además de test1,2 y 3.

Lo más importante del ejercicio es el diseño. Es decir, las clases deben tener solo la responsabilidad que les corresponde. Ni más ni menos.

El código fuente deberá ser entregado por mail antes del **17/09/2015**
La entrega del informe(por mail) se realizará a más tardar al comienzo de la clase del 17/09/2015

El informe deberá contener:
- Los nombres de los participantes.
- El código fuente de las funciones más importantes.
- Un informe explicando
    - **El cálculo del orden los algoritmos del ejercicio 1**.
    - Como eligieron la responsabilidad de cada clase.
    - Las decisiones tomadas
    - Lo especificado en el Apéndice I

Apéndice I: Contenido mínimo del informe

Ejercicio1

La justificación de la complejidad y las funciones más importantes del código fuente.

Ejercicio2 TAD Juego

-Para cada clase que lo compone

Clase1: Responsabilidad
…
ClaseN: Responsabilidad

-Decisiones de diseño tomadas

-Escribir el invariante de representación del Hormiguero.

Apéndice II: Código fuente otorgado por la cátedra. Tad Mapa

TAD Mapa

```
import java.util.ArrayList;

public class Mapa {
private Ciudad [] ciudades;
private int [][] ady;

private int n;

public Mapa(int tamano){
	n= tamano;
	ciudades = new Ciudad [n];
	ady = new int [n][n];

	for (int i=0; i<n; i++){
		ciudades [i] = new Ciudad(i,"Ciudad" + i);
		for (int j=0; j<n; j++){
			ady[i][j] = 0;	//distancias default en 0
		}
	}

}


public int tamano(){
	return n;
}

public void asignarNombre(Integer ciudadID, String nombre){
	ciudades[ciudadID].Nombre = nombre;
}

public void agDist(int i, int j, int dist, boolean simetrico){
	ady[i][j] = dist;

	if (simetrico){
		ady[j][i] = dist;

	}
}

// distancia en tiempos nominales
public int dist(int i, int j){ // si es adyacente tomamos ady[i][j]

return ady[i][j];
}
```

```
ArrayList<Ciudad> caminoMinimoProbable(Integer ciudadOrigen, Integer
ciudadDestino){
	ArrayList<Ciudad> ret = new ArrayList<Ciudad>();

	ret.add(ciudades[0]);

	//implementar!

	return ret;
}

Int distancia(ArrayList<Ciudad> ciudades){

	//implementar!

	Return 0;
}
```

TAD Ciudad

```
public class Ciudad {
	Integer id;
	String Nombre;

public Ciudad(Integer id, String nombre){
	this.id = id;
	this.Nombre = nombre;
}

@Override
public String toString() {
	return id + " " + Nombre;
}

}
```