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

Fecha de Presentación: 05/04/2016
Fecha de Entrega: 19/04/2016

**Ejercicio1) El rayo de calor de Arquímedes ¿mito o realidad?**[1]

Arquímedes de Siracusa (287 a. C. – ibídem, ca. 212 a. C.) fue un físico, ingeniero, inventor, astrónomo y matemático griego. Aunque se conocen pocos detalles de su vida, es considerado uno de los científicos más importantes de la Antigüedad clásica.

Según la tradición, dentro de sus trabajos en la defensa de Siracusa, Arquímedes podría haber creado un sistema de espejos ustorios que reflejaban la luz solar concentrándola en los barcos enemigos y con la finalidad de incendiarlos.

El artefacto, que en ocasiones es denominado como el "rayo de calor de Arquímedes", habría servido para enfocar la luz solar en los barcos que se acercaban, haciendo que estos ardieran.

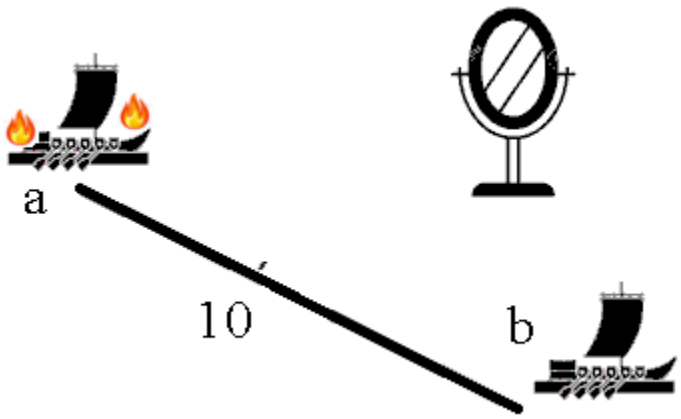
Estampa que reproduce el uso de espejos ustorios en la defensa de la ciudad de Siracusa durante el asedio romano

[1] https://es.wikipedia.org/wiki/Arqu%C3%ADmedes

Para quemar un barco, se tiene que girar el espejo, de un barco a otro.

Asumiremos que demora un segundo girar el espejo por cada metro de distancia entre dos barcos.

Por ejemplo, si se está quemando un barco situado en a, para quemar otro barco en b hay que girar el espejo, lo que demora 10 segundos, que es la distancia entre los barcos.




Distancia (a,b) = 10mts

En un primer planteo Arquímedes quería calcular todas las formas posibles de quemar los n barcos.

Para n barcos existen n! estrategias posibles para quemar los barcos.

Si plantear una estrategia de quemar los barcos demorase 1 segundo,

demoraríamos n! segundos para tomar esa decisión. Para ese entonces, Siracusa ya estaría invadida.

Se desea ayudar a Arquímedes con una estrategia que aunque no sea la óptima (la que mueve lo menos posible el espejo), dé alguna forma inteligente de quemar los barcos.

Aunque no siempre plantean la solución óptima (la menor cantidad de tiempo posible), se plantean dos alternativas:

a) Quemar el barco que esté más cerca del siguiente barco sin quemar.
   a. El 1er barco es el más cercano al espejo
b) Quemar el siguiente barco sin quemar que esté más cerca de la costa (del espejo)

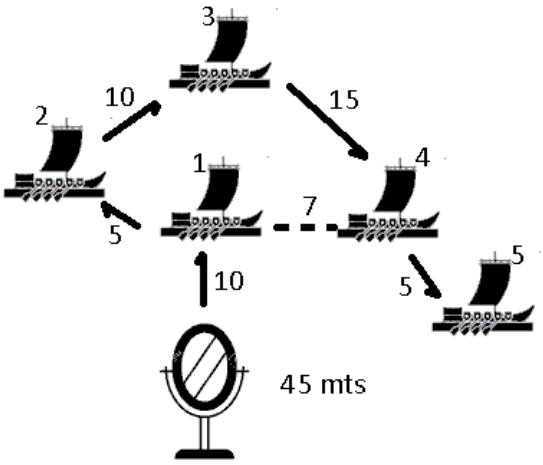


Figura 1a. Estrategia a.

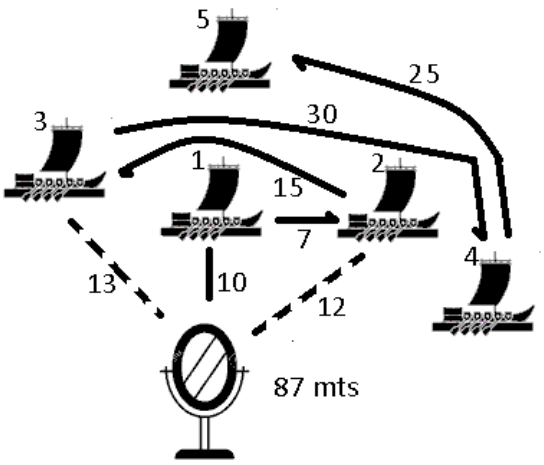


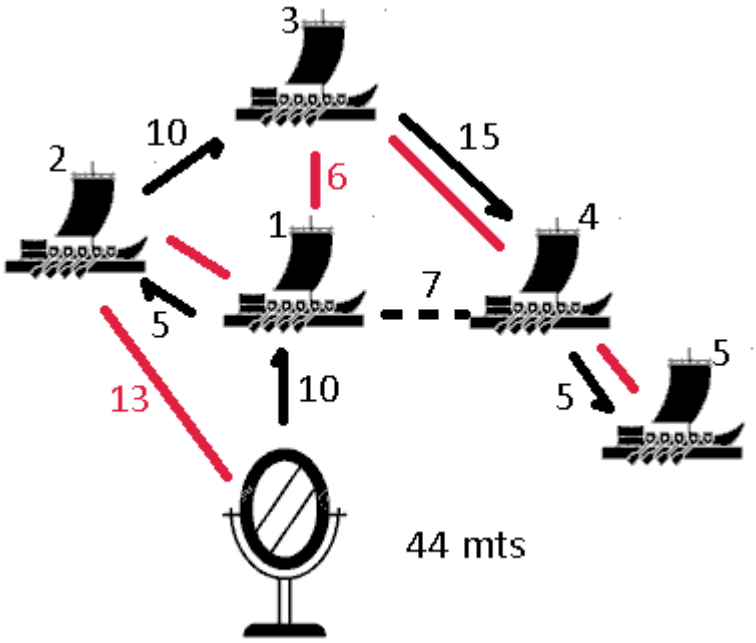
Figura 1b. Estrategia b.

Ambas figuras muestran el orden en que se eligieron los barcos para cada estrategia. En la Figura 1A el barco 2.º está a la menor distancia del 1.º (5 < 7), y el 3.º a la menor distancia del 2.º (10 < 15 < 30).

En la Figura 1B el barco 1.º está a la menor distancia del espejo (10 < 12 < 13), y el barco 2.º a la 2.ª menor distancia (12 < 13).

Para este ejemplo, la estrategia1 funciono mejor, aunque no necesariamente es siempre así.

El tiempo óptimo sin embargo era 44 mts, pero requeriría explorar todas las posibilidades!




Para ello la catedra otorgara el TAD Mapa (que modela los barcos y el espejo).
Dentro de dicho TAD implementar el método

ArrayList<Barco> caminoMinimoProbable()

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

i) Implementar los algoritmos a y b en dos códigos fuentes separados.
Es decir, las dos versiones de caminoMinimoProbable

ii) Calcular la complejidad de caminoMinimoProbable para cada código fuente (hacerlo en el informe. En el código se puede comentar brevemente).

-Nombrar las variables que definen el tamaño del problema

-Expresar la complejidad en dichas variables

**Test1: Implementa estratégia a**

```
public class Test1 {

public static void main(String[] args) {
Mapa m1 = new Mapa(5);
m1.asignarNombre(0, "B01");
m1.asignarNombre(1, "B02");
m1.asignarNombre(2, "B03");
m1.asignarNombre(3, "B04");
m1.asignarNombre(4, "B05");
m1.setDistEsp(0, 10 );
m1.setDistEsp(1, 13 );
m1.setDistEsp(2, 21 );
m1.setDistEsp(3, 12 );
m1.setDistEsp(4, 12 );
m1.setDistIJ(0, 1, 5, true);
m1.setDistIJ(0, 2, 6, true);
m1.setDistIJ(0, 3, 7, true);
m1.setDistIJ(0, 4, 9, true);
m1.setDistIJ(1, 2, 10, true);
m1.setDistIJ(1, 3, 15, true);
m1.setDistIJ(1, 4, 30, true);
m1.setDistIJ(2, 3, 15, true);
m1.setDistIJ(2, 4, 25, true);
m1.setDistIJ(3, 4, 5, true);

System.out.println(m1.distancia(m1.caminoMinimoProbable()));
System.out.println(m1.caminoMinimoProbable());

}
```

Consola

```
45
[0 B01, 1 B02, 2 B03, 3 B04, 4 B05]
```

**Ejercicio2) TAD Juego: Perros, gatos y gallinas (PGG)**

Una empresa de naipes nos pidió modelar un modelo computacional de un juego de naipes para chicos.
Y evaluar su viabilidad antes de sacarlo a la venta.

Una baraja tiene n naipes, según la cantidad que se desee comprar.
Cada naipe puede ser un Perro, un Gato o una Gallina.

Las reglas son las siguientes
-El Perro le gana al Gato
-El Gato le gana a la Gallina
-La Gallina le gana al Perro

El juego tiene dos jugadores que se reparten las cartas en partes iguales (asumir que n es par).
Cada jugador saca un naipe al azar, los naipes "compiten" y el ganador incluye ambos en su baraja.

El juego termina cuando un jugador se queda sin naipes o luego de 1000 rondas el jugador que tenga más naipes.

Se desea modelar TAD PGG, pero con la cantidad de naipes definida por el usuario.

**Enunciado:**

a) Diseñar e implementar el TAD PGG

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

Interfaz:

```
PGG (n)            // n > 1
Void agregarNaipe(…)    // Define un naipe de la baraja.

-Definir la interfaz. Y además decidir qué hacer si no se definen los n naipes
```

Test1: Ejemplo del código principal

```
PGG baraja = new PGG(8);            // Instancia: n == 8

baraja.agregarNaipe(…); // Agrego un Perro
baraja.agregarNaipe(…); // Agrego un Gato

while baraja.ganador() = "" && baraja.cantManosJugadas() < 1000{
      //como máximo hacerlo 1000 veces
      baraja.jugar();               // Se juega una mano
      System.out.println(baraja.ver())  //Se muestra un resumen del juego
}
System.out.println(baraja.cantManosJugadas());
```

b) Escribir el invariante de representación del juego:
Describir qué barajas son instancias válidas del PGG que se quiere representar
Que sucede en los casos extremos, como por ejemplo, si dos barajas se ganan mutuamente.

**Entrega y condiciones para aprobar el TP**

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

**Es condición excluyente para la aprobación entregar el Test1 de cada ejercicio funcionando.**

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 2016C1TP1 seguido de los apellidos de los integrantes, por ejemplo "2016C1TP1 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 **19/04/2016**
La entrega del informe (por mail) se realizará a más tardar al comienzo de la clase del 19/04/2016

**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**.
    - Cómo 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

-¿El diseño actual soporta agregar naipes de diferente tipo, además de los mencionados en el enunciado?
-Escribir el invariante de representación del PGG.

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

### TAD Mapa

```
import java.util.ArrayList;

public class Mapa {
private Barco [] barcos;
private int [][] distancias;  // distancia de i a j
private int [] distanciaEspejo; // distancia al espejo

private int n;

public Mapa(int tamano){
	n= tamano;
	barcos = new Barco [n];
	distancias = new int [n][n];
	distanciaEspejo = new int [n];

	for (int i=0; i<n; i++){
		barcos [i] = new Barco(i,"Barco" + i);
		distanciaEspejo[i] = 0;
		for (int j=0; j<n; j++){
			distancias[i][j] = 0;   //distancias default en 0

		}
	}

}


public int tamano(){
	return n;
}


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

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

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

public void setDistEsp(int i, int dist){
	distanciaEspejo[i] = dist;

}

// distancias nominales
public int getDistIJ(int i, int j){
```

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



ArrayList<Barco> caminoMinimoProbable(){
	ArrayList<Barco> ret = new ArrayList<Barco>();

//…

	return ret;
}

private Barco barcoMasCercano(Integer barcoOrigen, int [] quemado){

//…

}

Integer  distancia(ArrayList<Barco> camino){

	Integer _distancia = 0;

	for (int i=0;i< camino.size()-1;i++){
		_distancia = _distancia +
distancias[camino.get(i).id][camino.get(i+1).id];
	}

	return _distancia;

}
```

TAD Barco

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

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

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

}
```