Análisis y diseño de software

dit
UPM

# Recursividad /ejercicios

José A. Mañas

8.2.2015

ETSIT UPM

# índice

- ejercicios clásicos
- gráficos recursivos
- grafos

# ejercicio 1

## Función de Ackermann

- A(m, n) =

  n+1 si m = 0

  A(m-1, 1) si m > 0 & n = 0

  A(m-1, A(m, n-1)) si m > 0 & n > 0

- ejemplos

| **A(0, 0) = 1** | **A(0, 1) = 2** | **A(0, 2) = 3** | **A(0, 5) = 4** | **A(0, 4) = 5** |
|---|---|---|---|---|
| A(1, 0) = 2 | A(1, 1) = 3 | A(1, 2) = 4 | A(1, 3) = 5 | A(1, 4) = 6 |
| A(2, 0) = 3 | A(2, 1) = 5 | A(2, 2) = 7 | A(2, 3) = 9 | A(2, 4) = 11 |
| A(3, 0) = 5 | A(3, 1) = 13 | A(3, 2) = 29 | A (3, 3) = 61 | A(3, 4) = 125 |

# ejercicio 2 (Euclides)

GCD

- máximo común divisor
- dados enteros m, n > 0
- mcd(m, n) =
  - n  si m = n
  - mcd(m-n, n)  si m > n
  - mcd(m, n-m)  si m < n
- solución recursiva
- solución iterativa
- pruebas con JUnit

# ejercicio 3 (Euclides)

GCD

- máximo común divisor
- dados enteros m ≥ n > 0
- mcd(m, n) =
  - n  si m % n = 0
  - mcd(n, m%n)  si no
- solución recursiva
- solución iterativa
- pruebas con JUnit

# ejercicio 4

- combinaciones de n elementos tomados de k en k
- $\binom{n}{k} = \frac{n!}{k!(n-k)!}$
- $\binom{n}{0} = 1.$
  $\forall k > n : \binom{n}{k} = 0$
  $\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}.$

# ejercicio 5

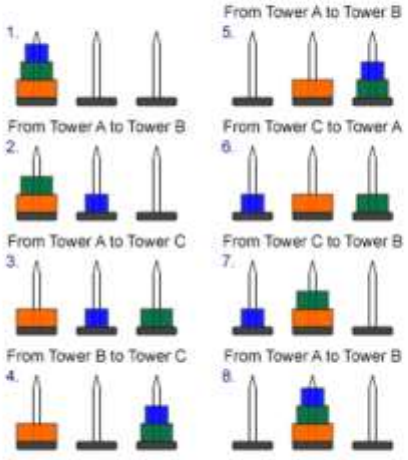
- dadas 2 listas ordenadas de String, combinarlas respetando el orden
- List<String> merge(List<String>, List<String> b)

# ejercicio 6.1

- torres de hanoi
- hay que mover N discos de la torre A a la torre B de forma que
  - sólo se puede mover un disco cada vez
  - nunca puede haber un disco de mayor tamaño sobre otro de menor tamaño
  - se puede usar una torre.

# ejercicio 6.2

- solución



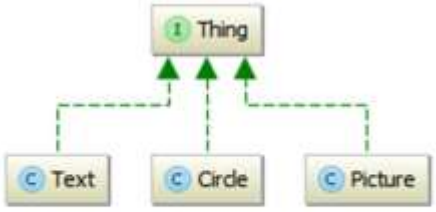
# ejercicio 6.3

- programar
  mueve(int n, int A, int B, int C)
- si n = 1, se mueve 1 de A a B
- si n > 1
  1. se mueven n-1 de A a C
  2. se mueve 1 de A a B
  3. se mueven n-1 de C a B

# índice

- ejercicios clásicos
- gráficos recursivos
- grafos

# entorno




- class Screen
- class Thing
  - class Circle implements Thing
  - class Picture implements Thing
  - class Text implements Thing
  - class … implements Thing
- …

http://www.dit.upm.es/~pepe/doc/adsw/tema1/exs/
http://www.dit.upm.es/~pepe/doc/adsw/tema2/exs/

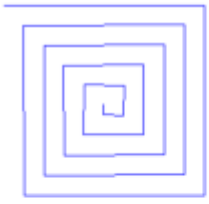
# recursión

- Usaremos una tortuga para pintar

```
public class Turtle {
public Turtle(Screen screen) {  }

   public void setColor(Color color) {  }

   public void move(double distance) {  }

   public void move(double dx, double dy) {  }

   public void rotate(double angle) {  }

   public void save() {  }

   public void back() {  }
}
```

## 51 – recursión espiral

- .../Spiral.mp4



java 14

1. posicionamos la tortuga en un punto y en una dirección
2. llamamos al método recursivo con un desplazamiento d:
   spiral(d)

el método recursivo spiral(d)

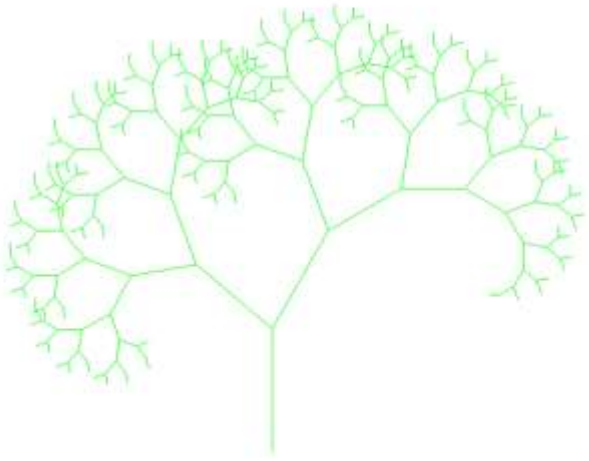
1. pinta un trazo de longitud d
2. gira 90º
3. llama a spiral(d-10)

condición de parada

- d <= 0

## 51 – recursión árbol

- .../Tree.mp4



1. posicionamos la tortuga en un punto y en una dirección
2. llamamos al método recursivo con un desplazamiento d:
   tree(d)

el método recursivo tree(d)

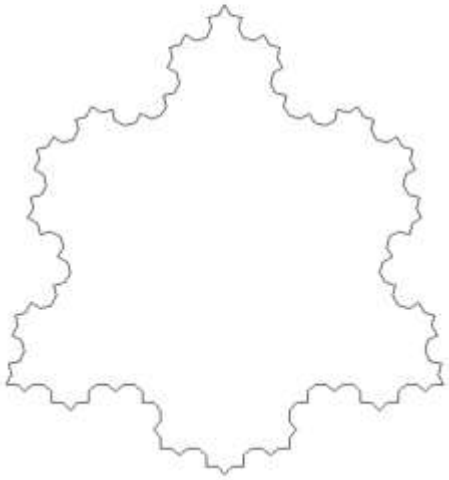
1. pinta un trazo de longitud d
2. guarda su posición
3. gira 30º
4. llama a tree(d * 0.75)
5. recupera la posición guardada
6. guarda su posición
7. gira -50º
8. llama a tree(d * 0.66)
9. recupera u posición

condición de parada

- d < 10

# 51 – recursión copo de nieve

- .../SnowFlake.mp4



## copo de nieve paso a paso

- la base es un triángulo equilátero

- cada lado se trocea recursivamente

para pintar cada tramo
paso(d):

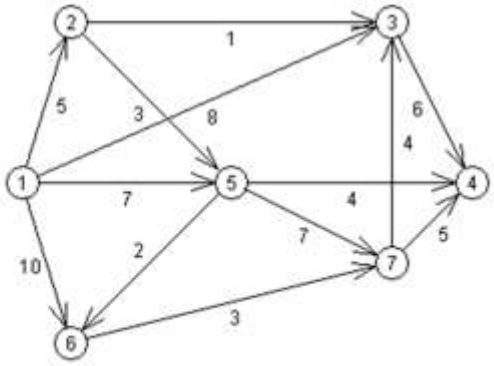
- si d < 20 → avanza d
- si no →
  - llama a paso(d/3)
  - gira 45º
  - llama a paso(d/3)
  - gira -90º
  - llama a paso(d/3)
  - gira 45º
  - llama a paso(d/3)

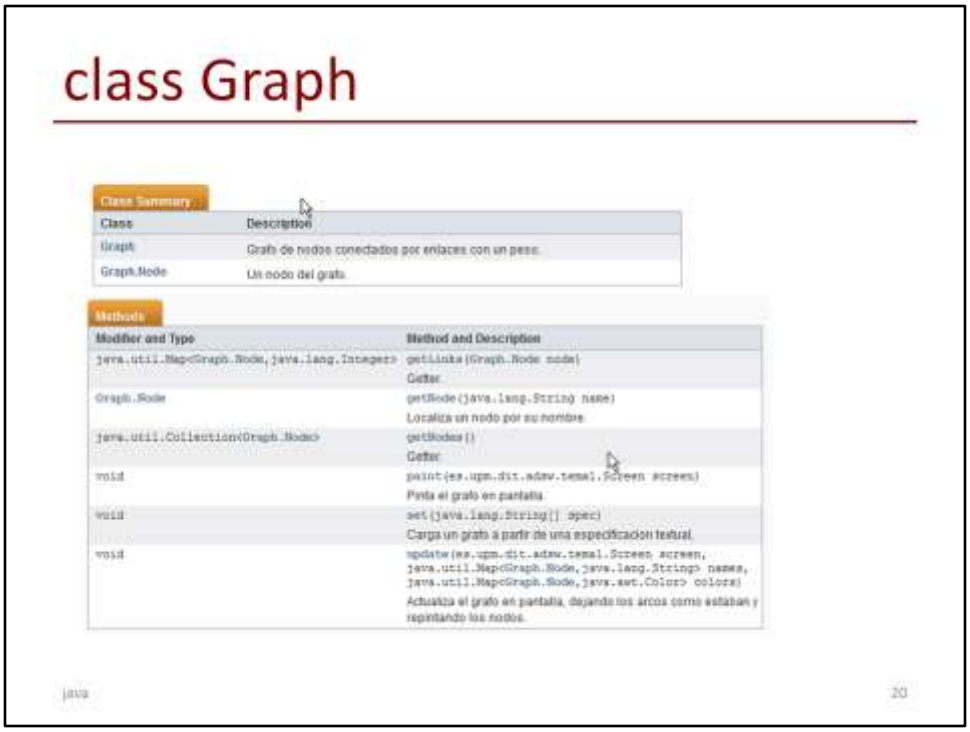
# índice

- ejercicios clásicos
- gráficos recursivos
- grafos

# grafos

- nodos
- con enlaces con un peso / distancia / ...





# class Graph

**Class Summary**

| Class | Description |
|---|---|
| Graph | Grafo de nodos conectados por enlaces con un peso. |
| Graph.Node | Un nodo del grafo. |

**Methods**

| Modifier and Type | Method and Description |
|---|---|
| java.util.Map<Graph.Node,java.lang.Integer> | getLinks(Graph.Node node)<br>Getter. |
| Graph.Node | getNode(java.lang.String name)<br>Localiza un nodo por su nombre. |
| java.util.Collection<Graph.Node> | getNodes()<br>Getter. |
| void | paint(es.upm.dit.adsw.temal.Screen screen)<br>Pinta el grafo en pantalla. |
| void | set(java.lang.String[] spec)<br>Carga un grafo a partir de una especificación textual. |
| void | update(es.upm.dit.adsw.temal.Screen screen, java.util.Map<Graph.Node,java.lang.String> names, java.util.Map<Graph.Node,java.awt.Color> colors)<br>Actualiza el grafo en pantalla, dejando los arcos como estaban y repintando los nodos. |

java 20

Map<Node node, Integer v) getLinks(Node node)

arcos o enlaces a partir de un nodo, indicando el valor v de cada arco

paint(Screen)

pinta los nodos con su nombre

pinta los arcos con su peso

update(Screen screen, Map<Node, String> names, Map<Node, Color> colors)

pinta los nodos con el texto indicado en “names”

pinta los nodos del color indicado en “colors”

# class Graph.Node

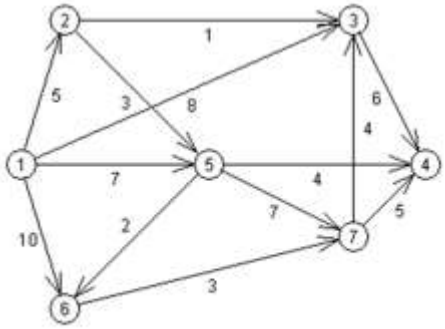
**Method Summary**

Methods

| Modifier and Type | Method and Description |
| --- | --- |
| java.lang.String | getName()<br>Getter |
| int | getX()<br>Getter |
| int | getY()<br>Getter |

# ejemplo de uso

```
public static void main(String[] args) {
    String[] spec = {
            "1 120,200 2:5 3:8 5:7 6:10",
            "2 150,100 3:1 5:3",
            "3 350,100 4:6",
            "4 400,200",
            "5 250,200 4:4 6:2 7:7",
            "6 150,300 7:3",
            "7 350,250 3:4 4:5",
    };

    graph = new Graph();
    graph.set(spec);

    screen = new Screen(450, 400);
    graph.paint(screen);
```

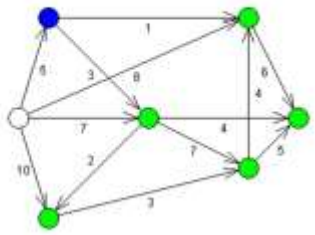



Ejemplo de uso: para recorrer los arcos que parten de un nodo

```
Graph.Node node1 = graph.getNode("2");
System.out.printf("node1: %s%n", node1.getName());
Map<Graph.Node, Integer> links = graph.getLinks(node1);
for (Graph.Node node2 : links.keySet()) {
   Integer v = links.get(node2);
   System.out.printf("  %s (%d)%n", node2.getName(), v);
}
```

## GraphEx1 - conectividad

- marcar todos los nodos accesibles dese uno inicial
  - Set<Graph.Node> visited
  - Graph.Node start = graph.getNode("2")
  - visited.add(start)
  - eval(start, visited)




Visitamos el nodo inicial. Visitamos (recursivamente) todos los enlaces que parten de ese nodo.

Para parar la recursividad, pasamos como argumento un conjunto de nodos visitados.

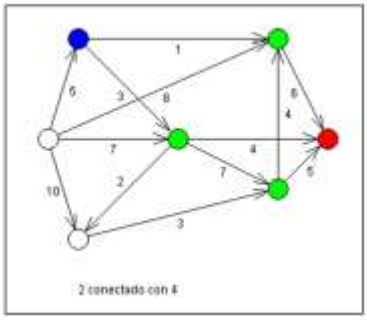
```
Set<Graph.Node> visited = new HashSet<Graph.Node>();
eval(inicial, visitados);
```

- Si al visitar un nodo ya esta en el conjunto, paramos la recursión.
- Cada ve que visitamos un nodo nuevo, lo metemos en el conjunto.

Al acabar, el conjunto tiene todos los nodos visitados que son todos los conectados directa o indirectamente con el nodo inicial.

# GraphEx2 - conectividad

- Queremos averiguar si un nodo origen está conectado con otro nodo destino
- Se usa el esquema del algoritmo anterior; pero si se llega a visitar el nodo destino terminamos la recursión

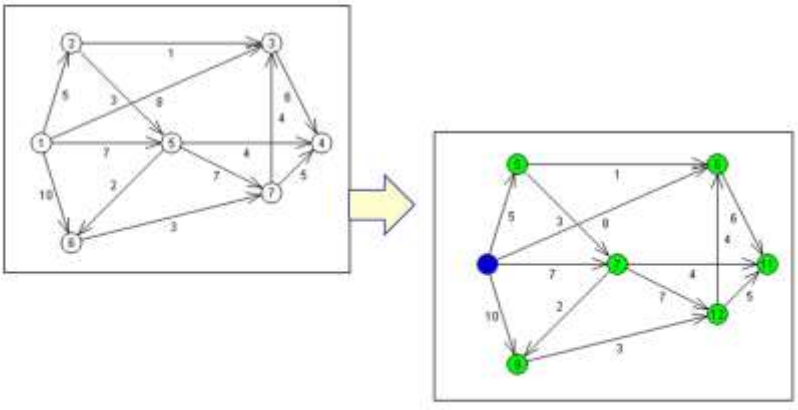



Igual que el algoritmo para localizar todo los conectados; pero le pasamos el nodo destino que buscamos y si lo visitamos paramos la recursión y devolvemos TRUE

boolean eval(Graph.Node from, Graph.Node dest, Set<Graph.Node> visited)

# GraphEx3 – distancia óptima

- Calcular la mínima distancia de un nodo origen a los nodos alcanzable



Se usa la función eval () recursiva; pero …

Apuntamos la mejor distancia para llegar a cada nodo

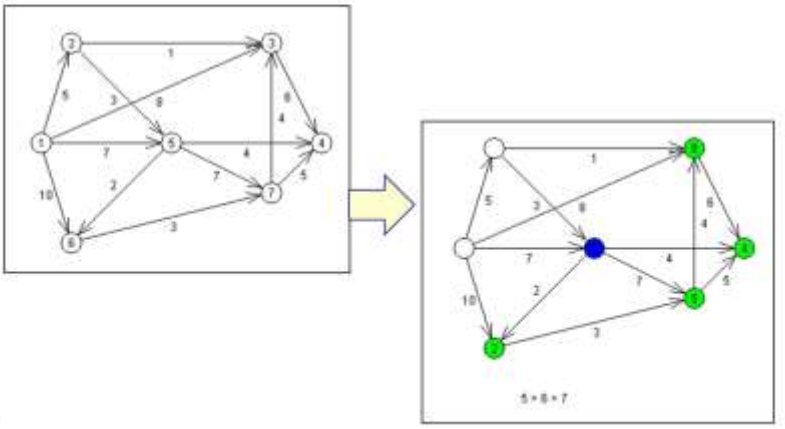
```
Map<Graph.Node, Integer> distances = new HashMap<Graph.Node,
Integer>();
```

Durante la evaluación miramos si la nueva distancia es menor que la que tenemos hasta el momento

```
eval(Node from, Map<Graph.Node, Integer> distances)
    Map<Graph.Node, Integer> links = graph.getLinks(from);
    for (Graph.Node node : links.keySet()) {
       int nd = distances.get(from) + links.get(node);
       if (distances.get(node) > dd) {
          llamamos de nuevo a eval para recalcular a partir de node
```

# GraphEx4 – ruta óptima

- Calcular la mínima distancia entre 2 nodos y la ruta óptima



Reusamos el algoritmo de calcular las distancias óptimas;

Como elemento nuevo, cada vez que actualizamos una distancia apuntamos en una memoria cual es el nodo previo desde el cual llego óptimamente.

O sea

Map<Node, Node>

En el ejemplo, al terminar tendremos esta memoria de caminos óptimos:

3 ← 7 – el mejor camino al 3 viene desde el 7
4 ← 5 – el mejor camino al 4 viene desde el 5
6 ← 5 – el mejor camino al 6 viene desde el 5
7 ← 6 – el mejor camino al 7 viene desde el 6

Por último hay que recorrer recursivamente la memoria de caminos para enlazarlos y concluir (en el ejemplo) que el mejor camino del 5 a 7 es

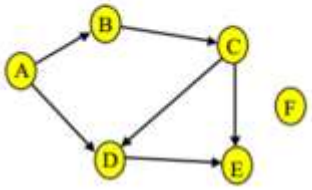
7 ← 6 ← 5

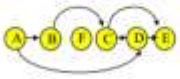
# GraphEx5 – orden topológico

http://courses.cs.washington.edu/courses/cse326/03wi/lectures/RaoLect20.pdf

- Se trata de crear una lista de nodos de forma que no haya arcos desde un nodo a nodos previos
  o que todos los arcos apunten a nodos posteriores
  - pueden haber varias soluciones

# GraphEx5 – ejemplo




- A B F C D E
- F A B C D E

# GraphEx5 – por partes …

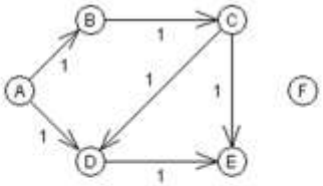
1. Escriba un método para validar que una lista de nodos es un ordenamiento topológico de un grafo

```
boolean verify(Screen screen,
  Graph graph,
  ArrayList<Graph.Node> sorted) {
```

# GraphEx5 – por partes …

2. Prepare un mapa de arcos de entrada
   - recuerde que
     getLinks() devuelve un mapa de arcos de salida

| arcos salida | |
|---|---|
| A | |
| | → B |
| | → D |
| B | |
| | → C |
| C | |
| | → D |
| | → E |
| D | |
| | → E |
| E | |
| F | |




| arcos entrada | |
|---|---|
| A | |
| B | |
| | ← A |
| C | |
| | ← B |
| D | |
| | ← A |
| | ← C |
| E | |
| | ← C |
| | ← D |
| F | |

# GraphEx5 – por partes …

3. Algoritmo

paso 0

- prepare una lista de nodos pendientes de visitar

paso1

- si no quedan nodos pendientes, hemos terminado
- seleccione un nodo sin arcos de entrada
- si no hay ninguno, fin: hay un bucle
- añádalo a la lista ordenada

paso 2

- elimine el nodo de la lista de pendientes
- elimine el nodo de los arcos de entrada
- repita el paso 1

# GraphEx5 – ejemplo con bucle