# Exámen 2012, preguntas y respuestas

Estructuras de datos y algoritmos (Universitat Politecnica de Valencia)

## Resolución del Tercer Parcial de EDA (6/6/2012)

**1.-** Sea la siguiente clase abstracta la raíz de la jerarquía que implementa un Grafo en Java:

```
public abstract class Grafo {
  protected int visitados[]; //Para marcar los vértices visitados en un DFS o BFS
  protected int ordenVisita; //Orden de visita de un vértice en un DFS o BFS
  // Otros atributos para la implementación de Caminos Mínimos
  …
  public abstract int numVertices();
  public abstract int numAristas();
  public abstract ListaConPI<Adyacente> adyacentesDe(int i);
  //Resto de métodos de la clase
  …
}
```

donde la clase *Adyacente* se define como sigue:

```
public class Adyacente {
  protected int destino; protected double peso;
  public Adyacente(int v, double peso){ destino = v; this.peso = peso; }
  public int getDestino(){ return this.destino; }
  public double getPeso(){ return this.peso; }
  public String toString(){ return destino + "("+ peso+ ") "; }
}
```

**Se pide** diseñar en la clase *Grafo* un método *enCiclo* que compruebe si el vértice *v* forma parte de un ciclo de un Grafo. Indicar también el coste Temporal del método diseñado, justificándolo adecuadamente. **(3 puntos)**

```
public boolean enCiclo(int v){
        visitados = new int[numVertices()];
        return enCicloDFS(v, v);
}
protected boolean enCicloDFS(int v, int v){
        boolean res = false; visitados[v] = 1;
        ListaConPI<Adyacente> l = adyacentesDe(v);
        for ( l.inicio(); !l.esFin() && !res; l.siguiente() ){
           int w = l.recuperar().getDestino();
           if ( visitados[w]==0 ) res = enCicloDFS(w, v);
           else if ( w==v ) res = true;
        }
        return res;
}
```

Análisis del coste Temporal del método. Sea x el tamaño de un Grafo *Simple*, x=f(|V|, |E|):

**En el Mejor de los Casos** *v* no tiene adyacentes, o existe un ciclo de longitud 3 formado por *v*, su 1er adyacente y el 1er adyacente a éste. Por tanto, $\mathbf{T^{M}_{enCiclo}(x) \in \Theta(1)}$ **y** $\mathbf{T_{enCiclo}(x) \in \Omega(1)}$.

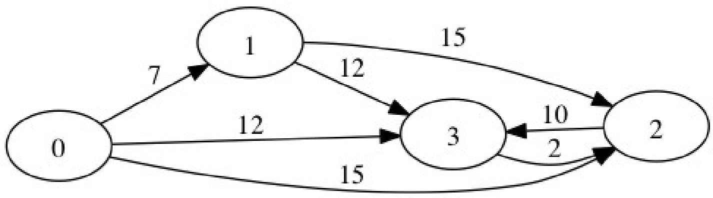
**En el Peor de los Casos** el Grafo es Conexo y Acíclico, por lo que en el DFS de *v* se visitan todas sus Aristas. Por tanto, $\mathbf{T^{P}_{enCiclo}(x) \in \Theta(|E|)}$ **y** $\mathbf{T_{enCiclo}(x) \in O(|E|)}$.

**2.-** El peso de entrada de un vértice *v* es la suma del peso de todas las Aristas de un Grafo que inciden en *v*. En la clase *Grafo* de la pregunta anterior, **se pide** diseñar un método *pesoEntrada* que devuelva el peso de entrada del vértice *v* de un Grafo. **(2 puntos)**

```
public double pesoEntrada(int v){
        double res = 0.0;
        for ( int i=0; i<numVertices(); i++ ){
              ListaConPI<Adyacente> l = adyacentesDe(i);
              for ( l.inicio(); !l.esFin(); l.siguiente() ){
                    Adyacente a = l.recuperar();
                    if ( a.getDestino()==v ) res += a.getPeso();
              }
        }
        return res;
}
```

Análisis del coste Temporal del método. Sea x=f(|V|, |E|); en *pesoEntrada* se efectúa un Recorrido -sin instancias- de todas las Aristas del Grafo, por lo que $\mathbf{T_{pesoEntrada}(x) \in \Theta(|E|)}$.

**3.- Se pide** realizar una traza del algoritmo de Dijkstra desde el vértice **0** a todos los demás del siguiente Grafo. **(2 puntos)**




| Vértice | Adyacentes |
|---|---|
| 0 | (1,7) (2,15) (3,12) |
| 1 | (2,15) (3,12) |
| 2 | (3,10) |
| 3 | (2,2) |

| | distanciaMin | | | | caminoMin | | | | |
|---|---|---|---|---|---|---|---|---|---|
| v | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 3 | qPrioridad |
| | 0 | ∞ | ∞ | ∞ | -1 | -1 | -1 | -1 | (0,0) |
| 0 | | 7 | 15 | 12 | | 0 | 0 | 0 | (1,**7**) (2,15) (3,12) |
| 1 | | | | | | | | | (2,15) (3,**12**) |
| 3 | | | **14** | | | | **3** | | (2,15) (2,**14**) |
| 2 | | | | | | | | | (2,15) |
| 2 | Ya visitado | | | | | | | | Vacía |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |

**4-** Sea la siguiente clase Java una implementación de la interfaz *MFSet* en la que el identificador de cada subconjunto es la raíz del Árbol que lo representa (*elArray[i]=i*).

```
public class ForestMFSetNormal implements MFSet {

  protected int talla;   // Nº de elementos
  protected int n_particiones; // Nº de árboles
  protected int elArray[];

  public ForestMFSetNormal(int n){…}
  public int find(int x){…}
  public void merge(int x, int y){…}
}
```

**Se pide** diseñar en esta clase *ForestMFSetNormal* un método *cambiar* que asigne *x* al subconjunto al que pertenece *y,* siempre que *x* no sea el identificador de su subconjunto. Indicar también el coste Temporal del método diseñado, justificándolo adecuadamente. **(3 puntos)**

```
public void cambiar(int x,int y){
   if ( elArray[x]!=x ){
       int raizX = find(x); int raizY = find(y);
       for ( int i=0; i<talla; i++ )
            if ( elArray[i]==x ) elArray[i] = raizX;
       elArray[x] = raizY;
   }
}
```

Análisis del coste Temporal del método. Sea x=*talla*.

**En el Mejor de los Casos** *x* es el identificador de su subconjunto (*elArray[x]==x*), por lo que no se realiza el cambio. Así, $\mathbf{T^{M}_{cambiar}(x) \in \Theta(1)}$ **y** $\mathbf{T_{cambiar}(x) \in \Omega(1)}$.

**En el Peor de los Casos** *x* se cambia al subconjunto de *y*, manteniendo el resto del conjunto al que pertenecía *x* agrupado adecuadamente en Θ(x) (bucle *for*); por ello, aunque el coste de los *find* que se realizan para obtener *raizX* y *raizY* fuera O(x), $\mathbf{T^{P}_{cambiar}(x) \in \Theta(x)}$ **y** $\mathbf{T_{cambiar}(x) \in O(x)}$.