4 Recorridos de grafos

4 Recorridos de grafos: Introducción

- Justificación de la necesidad:

  Programas que precisan aplicar sistemáticamente un tratamiento a todos los vértices de un grafo (*visitar* todos los vértices).

- Consideraciones generales:

2 casos

- No importa el orden de visita ⇒
  ⇒ **para todo** v **en** vért **hacer**...
- El orden de visita depende de las aristas existentes ⇒
  ⇒ diversas estrategias de recorrido

  - recorrido en profundidad
  - recorrido en anchura
  - recorrido por ordenación topológica

Característica común: uso de conjuntos para almacenar los vértices visitados
(a diferencia de en los árboles, a un mismo nodo puede accederse por distintos caminos, y hay que evitar visitas reiteradas).

4 *Recorridos de grafos: Recorrido en profundidad*

J.E. Hopcroft y R.E. Tarjan:
"Efficient algorithms for graph manipulation",
Communications of the ACM, 16(6), pp. 372-37

- Para grafos dirigidos

  Para no dirigidos: considerar cada arista no dirigida como un par de aristas dirigidas.

- Es una generalización del recorrido preorden de un árbol:
  - se comienza visitando un nodo cualquiera;
  - se recorre en profundidad el componente conexo que cuelga de cada sucesor
    (i.e., se examinan los caminos hasta que se llega a nodos ya visitados o sin sucesores);
  - si después de haber visitado todos los sucesores transitivos del primer nodo (i.e., él mismo, sus sucesores, los sucesores de sus sucesores, …) todavía quedan nodos por visitar, se repite el proceso a partir de cualquiera de estos nodos no visitados.

4 Recorridos de grafos: Recorrido en profundidad

- Terminología:
  - Sucesores transitivos de un nodo: **descendientes**.
  - Predecesores transitivos: **antecesores**.

- No hay un único recorrido en profundidad de un grafo sino un conjunto de ellos.

- Aplicaciones:
  - analizar la robustez de una red de computadores representada por un grafo no dirigido [AHU88, pp. 243-245]
  - examinar si un grafo dirigido tiene ciclos, antes de aplicar sobre él cualquier algoritmo que exija que sea acíclico

4 Recorridos de grafos: Recorrido en profundidad

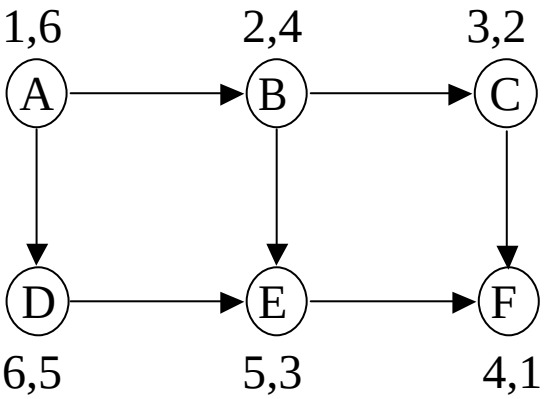
```
{Pre: g es un grafo dirigido y no etiquetado}
función recorEnProf(g:grafo)
                devuelve listaVért
variables S:cjtVért; {conjunto de vért.}
          v:vért;
          l:listaVért
principio
  S:=∅;
  creaVacía(l);
  para todo v en vért hacer
    si v∉S
      entonces visitaComponente(g,v,S,l)
    fsi
  fpara
  devuelve l
fin
{Post: esRecorEnProf(l,g)}
```

4 Recorridos de grafos: Recorrido en profundidad

```
{Pre: v∉S 3 esRecorEnProf(l,subgrafo(g,S)) 3
   3 ∀u∈vért:
    (u∈l⇔u∈S)3(u∈l⇒∀w∈descendientes(g,u):w∈l)}
algoritmo visitaComponente(ent g:grafo;
                           ent v:vért;
                           ent/sal S:cjtVért;
                           ent/sal l:listaVért)
variable w:vért
principio
  S:=S∪{v};
  inserta(l,v); {inserta v en la lista l}
  para todo w en suc(g,v) hacer
    si w∉S
      entonces visitaComponente(g,w,S,l)
    fsi
  fpara
fin
{Post: v∈S 3 esRecorEnProf(l,subgrafo(g,S)) 3
   3 ∀u∈vért:
   (u∈l⇔u∈S)3(u∈l⇒∀w∈descendientes(g,u):w∈l)}
```

4 Recorridos de grafos: Recorrido en profundidad

- Existe una versión simétrica:
  - no visitar un nodo hasta haber visitado todos sus descendientes (generalización del recorrido postorden de un árbol)
  - se habla de *recorrido en profundidad hacia atrás* frente al *recorrido en profundidad hacia adelante*
  - en el algoritmo anterior basta con mover la inserción de un nodo en la lista detrás del bucle



Orden de visita a los vértices de un grafo partiendo desde A con un recorrido en profundidad hacia adelante (izquierda) y hacia atrás (derecha)

4 Recorridos de grafos: Recorrido en profundidad

- Coste temporal:
  depende exclusivamente de la implementación elegida para el grafo
  - listas (múltiples o no) de adyacencia: $\Theta(a+n)$
    (dado un vértice siempre se consultan las aristas que parten de él, aunque no se visite más que una vez; el factor n aparece por si el grafo tiene menos aristas que vértices)
  - matriz de adyacencia: $\Theta(n^2)$
    (por el tiempo lineal de la operación suc)

  → si el grafo es disperso, la representación por listas es más eficiente

- Espacio adicional: lineal en el número de nodos

4 *Recorridos de grafos: Recorrido en anchura*

E.F. Moore: “The shortest path through a maze”
Proceedings of the International Symposium on
Theory of Switching, Harvard Univ. Press, 1959

- Es una generalización del recorrido por niveles de un árbol:
  - después de visitar un vértice se visitan los sucesores,
  - después los sucesores de los sucesores,
  - después los sucesores de los sucesores de los sucesores, etc.,
  - si después de visitar todos los descendientes del primer nodo todavía quedan más nodos por visitar, se repite el proceso.
- Aplicación típica:
  - problemas de planificación (ej.: atravesar un laberinto)
- Eficiencia: igual al recorrido en profundidad

4 Recorridos de grafos: Recorrido en anchura

```
{Pre: g es un grafo dirigido y no etiquetado}
función recorEnAnch(g:grafo)
                devuelve listaVért
variables S:cjtVért; {conjunto de vért.}
          v:vért;
          l:listaVért
principio
  S:=∅;
  creaVacía(l);
  para todo v en vért hacer
    si v∉S
      entonces visitaComponente(g,v,S,l)
    fsi
  fpara
  devuelve l
fin
{Post: esRecorEnAnch(l,g)}
```

4 Recorridos de grafos: Recorrido en anchura

```
{Pre: v∉S 3 esRecorEnAnch(l,subgrafo(g,S)) 3
   3 ∀u∈vért:
    (u∈l⇔u∈S)3(u∈l⇒∀w∈descendientes(g,u):w∈l)}
algoritmo visitaComponente(ent g:grafo;
  ent v:vért; ent/sal S:cjtVért;
  ent/sal l:listaVért)
variables c:colaVért; u,w:vért
principio
  S:=S∪{v}; creaVacía(c); añadir(c,v);
  mq not esVacía(c) hacer
    w:=primero(c);eliminar(c);inserta(l,w);
    para todo u en suc(g,w) hacer
    si u∉S entonces S:=S∪{u};añadir(c,u) fsi
    fpara
  fmq
fin
{Post: v∈S 3 esRecorEnAnch(l,subgrafo(g,S)) 3
   3 ∀u∈vért:
   (u∈l⇔u∈S)3(u∈l⇒∀w∈descendientes(g,u):w∈l)}
```

4 Recorridos de grafos: Ordenación topológica

- Aplicable sólo a grafos dirigidos acíclicos
- Un vértice sólo se visita si han sido visitados todos sus predecesores
  - En particular, al comenzar sólo se pueden visitar los nodos que no tienen ningún predecesor
- Aplicación:
  - Representación de las fases de un proyecto mediante un grafo dirigido acíclico:
    los vértices representan tareas y las aristas relaciones temporales entre ellas.
  - Evaluación de atributos en la fase semántica de un compilador.

4 Recorridos de grafos: Ordenación topológica

```
{Pre: g es un grafo dirigido,no etiquetado y
     acíclico}
función ordTopológica(g:grafo)
                  devuelve listaVért
variable l:listaVért; v,w:vért
principio
  creaVacía(l);
  mq queden nodos por visitar hacer
    escoger v∉l t.q. todos sus pred.
                       estén en l;
    inserta(l,v)
  fmq
  devuelve l
fin
{Post: esOrdTopológica(l,g)}
```

- Problema: es costoso buscar vértices sin predecesores repetidas veces.
- Una solución: calcular previamente cuántos predecesores tiene cada vértice, almacenar el resultado en un vector y actualizarlo siempre que se incorpore un nuevo vértice a la solución.

4 Recorridos de grafos: Ordenación topológica

```
{Pre: g es un grafo dirigido,no etiquetado y
     acíclico}
función ordTopológica(g:grafo)
                    devuelve listaVért
variables l:listaVért; v,w:vért;
  ceros:cjtVért; númPred:vector[vért]de nat
principio
  {inicializ. de las estructuras en dos pasos}
  ceros:=∅;
  para todo v en vért hacer
    númPred[v]:=0; ceros:=ceros∪{v}
  fpara;
  para todo v en vért hacer
    para todo w en suc(g,v) hacer
      númPred[w]:=númPred[w]+1;
      ceros:=ceros-{w}
    fpara
  fpara;
  ...
```

4 Recorridos de grafos: Ordenación topológica

```
...
{visita a los vértices del grafo}
creaVacía(l);
mq ceros?∅ hacer
  <ceros,v>:=escogerUnoCualquiera(ceros);                    (*)
  inserta(l,v);
  para todo w en suc(g,v) hacer
    númPred[w]:=númPred[w]-1;
    si númPred[w]=0
      entonces ceros:=ceros∪{w}
    fsi
  fpara
fmq;
devuelve l
fin
{Post: esOrdTopológica(l,g)}
```

(*) Devuelve un elemento cualquiera del conjunto y lo borra.

4 Recorridos de grafos: Ordenación topológica

- Coste temporal:
  - inicialización:
    - primer bucle: $\Theta(n)$
    - segundo bucle: examen de todas las aristas
      $\Theta(a+n)$ para listas de adyacencia
      $\Theta(n^2)$ para matriz de adyacencia
  - bucle principal:
    $\Theta(a+n)$ para listas de adyacencia
    $\Theta(n^2)$ para matriz de adyacencia