Estructuras de Datos

Algoritmos sobre Grafos: Recorridos.

Contenidos

- Recorrido en profundidad.
- Recorrido en anchura.
- Ordenación topológica.

Motivación

- Los grafos en general contendrán ciclos.
- Si queremos procesar el grafo, tendremos que evitar los ciclos.

Recorridos de Grafos

- Consideraciones generales:
  - Cada vértice, para evitar los ciclos, tiene una propiedad “visitado” que se activa para indicar que ya ha sido procesado.
  - Además, es posible que se le añada a los vértices un índice que indique el orden de procesamiento.
  - Al eliminar los ciclos, los recorridos darán como resultado un árbol abarcador.
  - Si el grafo no es conexo, para cubrir todo el grafo, se necesitarán más de un árbol abarcador.
  - Los grafos no imponen un orden de vértices ni de lados**.**
  - **Pero nosotros sí**: en los ejemplos mostrados:
    - Los vértices se recorren de “**menor a mayor**” según su etiqueta.
    - Los lados incidentes en un vértice se recorren de “**menor a mayor**” según la etiqueta del otro extremo.

Recorridos de Grafos

- Recorrido primero en profundidad.

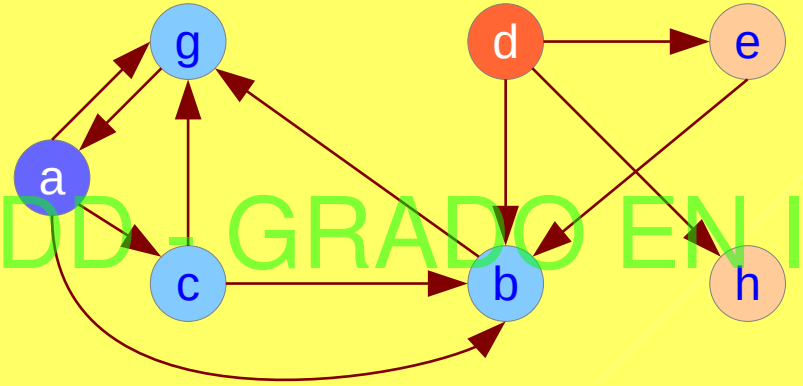
Claves:
- Los nodos y tienen un campo “visitado” que usaremos para evitar ciclos.
- El recorrido genera un **árbol abarcador en profundidad (spanning tree)** avanzando a partir de un nodo todo lo que se pueda.


```
dfScan(VAR g:Graph[V,E], f:Functor)
Var
  u: VertexIterator[V]
Begin
  g.reset(False)
  u := g.beginVertex()
  While u<>g.endVertex() Do
    If Not u.get().isVisited() Then
      dfScan(g, u, f)
    u.gotoNext()
  End-While
End.
```

```
dfScan(VAR g:Graph[V,E],
   u:VertexIterator[V]; f:Functor)
Local:
   e : EdgeIterator[V,E]
   v : Vertex[V]
Begin
  f(u.get())
  u.get().setVisited(True)
  e := g.beginEdge(u)
  While e<>g.endEdge(u) Do
    v := e.get().other(u.get())
    If Not v.isVisited() Then
      dfScan(g, g.getIterator(v), f)
    e.goToNext()
  End-While
End.
```

Recorridos de grafos

- Recorrido profundidad: ejemplo.

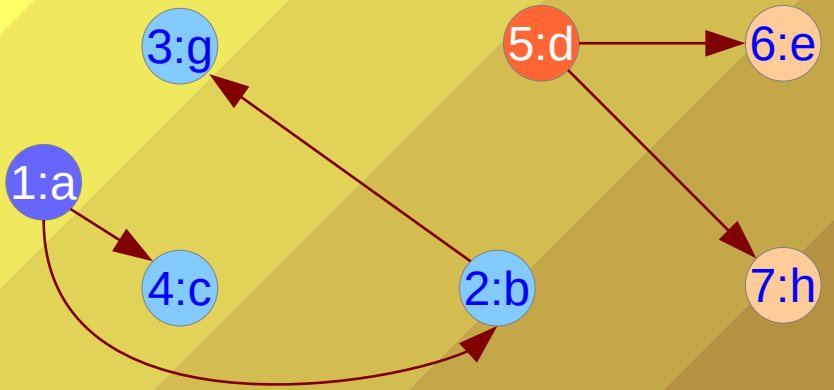


1	2	3	4	-	5	6	7
a:a	b:a c:a g:a	g:b c:a g:a	c:a g:a	g:a	d:d	e:d h:d	h:d

g llegando desde **a**

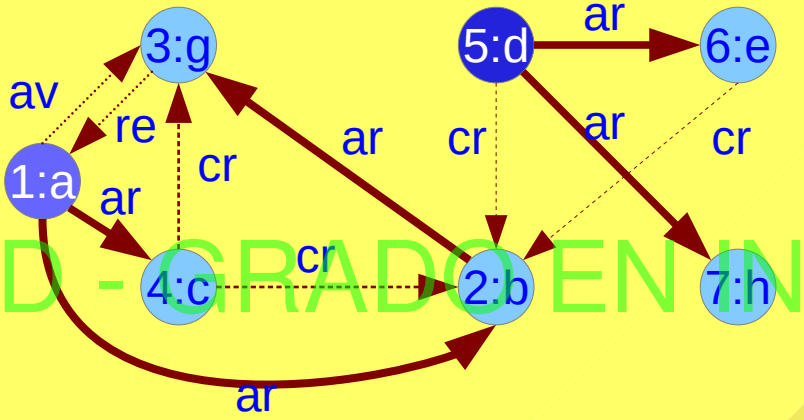
Si el proceso fuera "print(vertex)":

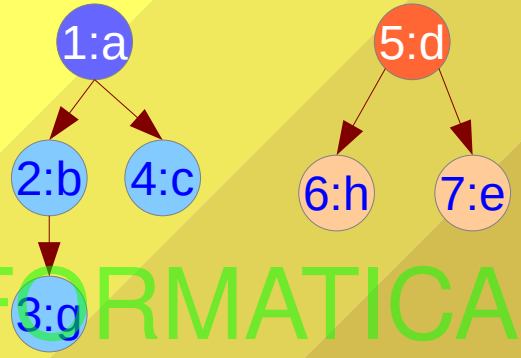
"abgcdeh"



Recorridos de grafos

- Recorrido en profundidad: clasificación de los lados.





Bosque abarcador en profundidad.

Propiedad:
Los n vértices descendientes en el árbol abarcador con raíz n1 cumplen:

$$o(n1)<o(n)<=o(n1) + d(n1)$$

con:
o(n)= orden de n.
d(n)= número de descendientes de n.

Tipos de lados para grafos dirigidos:
- ar: del árbol abarcador.
- av: (n1,n2) es de *avance* si: o(n1)<o(n2)<=o(n1)+d(n1)
- re: (n2,n1) es de *retroceso* si: o(n1)<o(n2)<=o(n1)+d(n1)
- cr: (n1,n2) es *cruzado* si: o(n1)>o(n2)+d(n2).

Tipos de lados para grafos no dirigidos:
- ar: del árbol abarcador.
- ot: de avance/retroceso.
- no hay lados cruzados.

Recorridos de grafos

- Recorrido en amplitud.

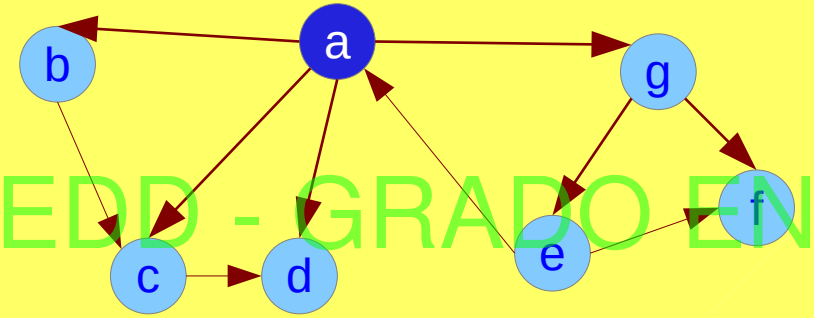
Claves:
- Partimos de un nodo s.
- Recorremos el grafo utilizando una **cola** para guiar el recorrido.
- El recorrido genera un **árbol abarcador en amplitud** donde cada nivel del árbol representa un salto en la distancia de los nodos respecto al nodo inicial.


```
bfScan(Var g:Graph[V,E], f:Functor)
Var u:VertexIterator[V]
Begin
  g.reset(False)
  u := g.beginVertex()
  While u<>g.endVertex() Do
    If Not u.get().isVisited() Then
      bfScan2(g, u, f)
    u.gotoNext()
  End-While
End.
```

```
bfScan2(Var g:Graph[V,E], u: VertexIterator[V],
f:Functor)
Var
  q: Queue[Vertex[V]]
  e: EdgeIterator[V,E]
  u,v: Vertex[V]
Begin
  q.enque(u.get())
  While Not q.isEmpty() Do
    u := q.front()
    q.deque()
    If Not u.isVisited() Then
      f(u)
      u.setVisited(True)
      e := g.beginEdge(g.getIterator(u))
      While e <> g.endEdge(g.getIterator(u)) Do
        v := e.get().other(u)
        If Not v.isVisited() Then
           s.enque(v)
        e.gotoNext()
      End-While
    End-If
  End-While
End.
```

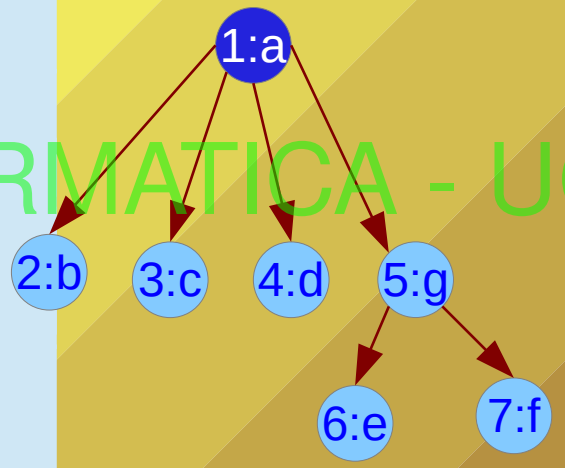
Recorridos de grafos

- Recorrido primero en amplitud: ejemplo.




```
1 a:a|
2 b:a|c:a d:a g:a
3 c:a|d:a g:a c:b
4 d:a|g:a c:b d:c
5 g:a|c:b d:c
- c:b|d:c e:g f:g
- d:c|e:g f:g
6 e:g|f:g
7 f:g|f:e
- f:e|
```

f llegando desde **e**



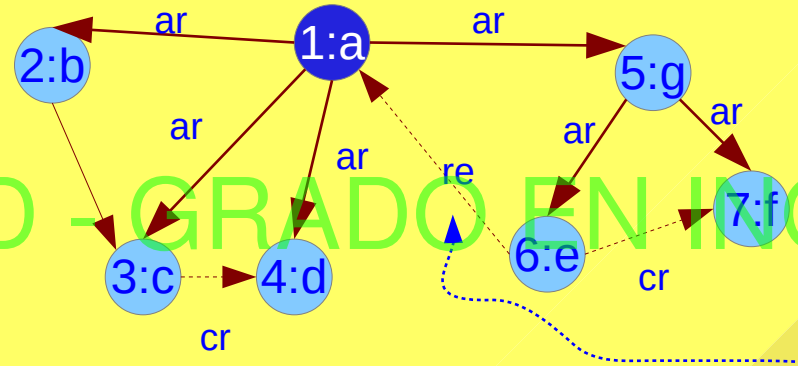
Árbol abarcador en amplitud.

Si el proceso fuera "print(vertex)":

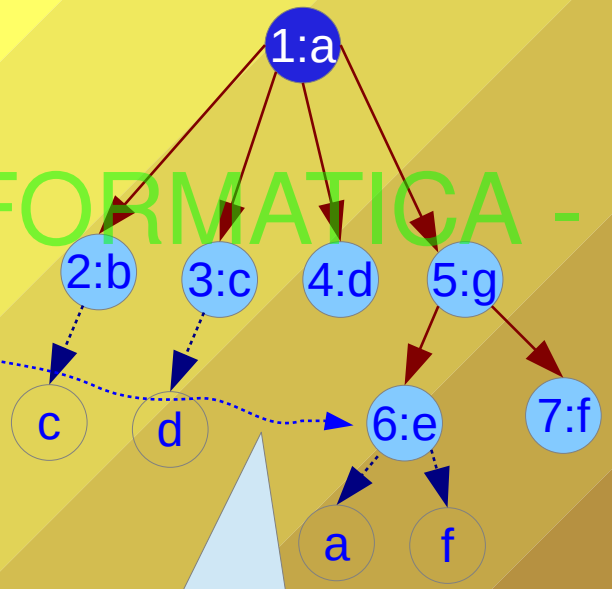
"abcdgef"

Recorridos de grafos

- Recorrido primero en amplitud: clasificación de los lados.



Árbol abarcador en anchura.



Tipos de lados para grafos dirigidos:

- ar: *Del árbol abarcador*.
- re: (n2,n1) es *de retroceso* si n2 es un descendiente de n1 en el árbol abarcador.
- cr: el resto son cruzados ya que no puede haber lados de avance.

Tipos de lados para grafos no dirigidos:

- Del árbol.
- Cruzados.

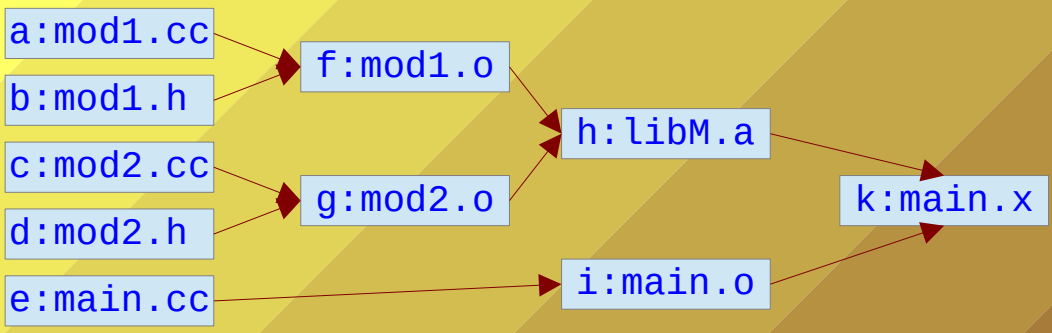
Lado de retroceso porque "e" es descendiente de "a" en el árbol abarcador

Recorridos de grafos

- Ordenación topológica.
  - Se aplica a Grafos Dirigidos Acíclicos (**GDA**).
  - La ordenación topológica **crea una ordenación lineal** de los vértices que cumple que para todo lado u->v, u aparece antes que v en la ordenación.

Makefile:

```
main.x: main.o libM.a
libM.a: mod1.o mod2.o
mod1.o: mod1.cc mod1.h
mod2.o: mod2.cc mod2.h
main.o: main.cc
```



Recorridos de grafos

- Ordenación topológica: algoritmo.

```
TopologicalSorting(Var
g:Graph):List[Vertex[V]]
//PRE-C: g is a DAG.
Var:
  list: List[Vertex[v]]
  u: VertexIterator[V]
Begin
  g.reset()
  u := g.beginVertex()
  While u <> g.endVertex() Do
    If not u.get().isVisited() Then
      TSScan(g, u, list)
    u.gotoNext()
  End-While
  Return list
End.
```

```
TSScan2(Var g:Graph,
        u:VertexIterator[V],
        Var List[Vertex[V]])
Var
  e: EdgeIterator[V,E]
  v: Vertex
Begin
  e := g.edgeBegin(u)
  While e<>g.endEdge(u) Do
    v := g.get().other(u.get())
    If Not v.isVisited() Then
      TSScan2(g, g.getIterator(v), list)
    e.gotoNext()
  End-While
  u.SetVisited(True) //Proceso postfijo
  list.pushFront(u.get())
End.
```

Recorridos de grafos

- Ordenación topológica: ejemplo.

Makefile:

```
main.x: main.o libM.a
libM.a: mod1.o mod2.o
mod1.o: mod1.cc mod1.h
mod2.o: mod2.cc mod2.h
main.o: main.cc
```


Grafo

Proceso

Ordenación topológica

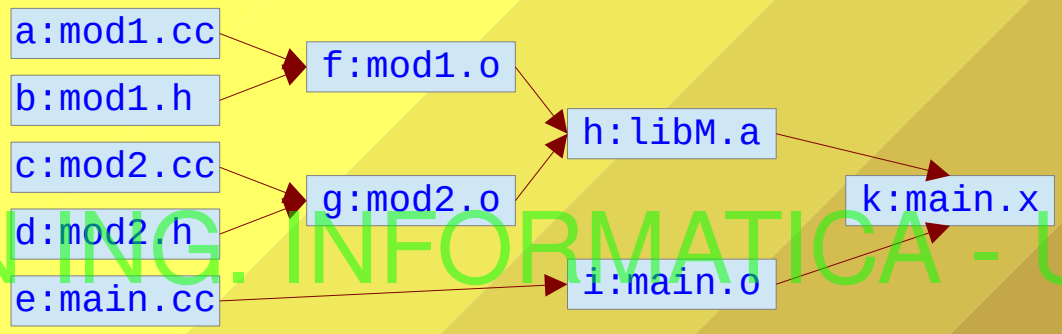
Ordenación

Recorridos de grafos

- Ordenación topológica: ejemplo.

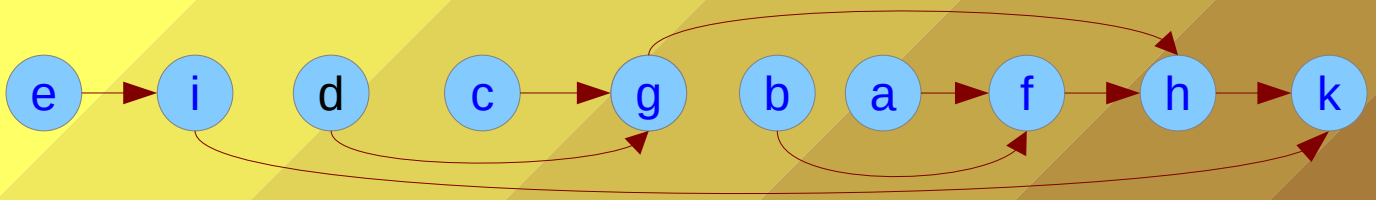
Makefile:

```
main.x: main.o libM.a
libM.a: mod1.o mod2.o
mod1.o: mod1.cc mod1.h
mod2.o: mod2.cc mod2.h
main.o: main.cc
```



-	-	-	1	2	3	4	5	-	6	7	8	-	9	10
a	f	h	k	h	f	a	b	c	g	c	d	e	i	e
	a	f	h	f	a				c				e	
		a	f	a										
			a											

Ordenación topológica



Resumen

- Los grafos tienen ciclos y tenemos que evitarlos cuando procesamos el grafo.
- Los dos recorridos básicos para procesar son:
  - En profundidad (usamos una Pila)
  - En anchura (usamos una Cola)
- Con los recorridos podemos definir un árbol abarcador y clasificar los lados del grafo: del árbol, de avance/retroceso y cruzados.
- La ordenación topológica es una forma de recorrer un grafo dirigido acíclico.

Referencias

- Lecturas recomendadas.
  - Caps. 14, 15 y 16 de “Estructuras de Datos”, A. Carmona y otros. U. de Córdoba. 1999.
  - Wikipedia:
    - Ordenación topológica: https://en.wikipedia.org/wiki/Topological_sorting
    - Recorrido primero en amplitud: https://en.wikipedia.org/wiki/Breadth-first_search
    - Recorrido primero en profundidad: https://en.wikipedia.org/wiki/Depth-first_search