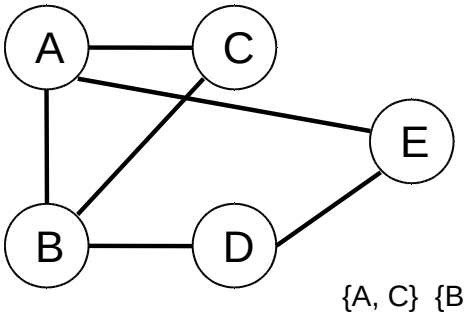
Grafoak

Koldo Gojenola eta Bertol Arrieta
Lengoaia eta sistema informatikoak
UPV-EHU

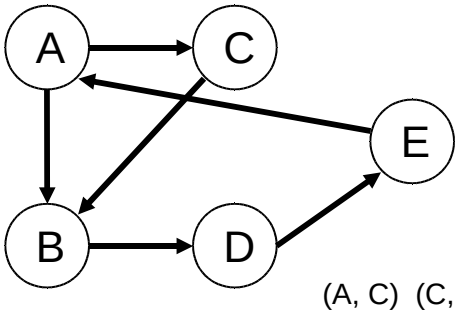
Zer da grafo bat?

- G=(N, A)
  - N adabegien multzo bat da (edo erpinak)
  - A arkuen multzoa (adabegi bikotez adieraziak)



EZ zuzendua

{A, C} {B, C}
{A, B} {D, B}
{A, E} {E, D}



zuzendua

(A, C) (C, B)
(A, B) (B, D)
(E, A) (D, E)

Zenbait kontzeptu

- Auzokidetasuna: bi adabegien artean auzokidetasun erlazioa dagoela esango dugu, baldin eta bi adabegi horiek lotzen dituen arkurik badago.
- Ibilbidea: grafoko bi adabegi lotzen dituen arkuen sekuentzia bat da.
- Zikloa: hasten den adabegi berean bukatzen den ibilbidea da, non ez baita arkurik errepikatzen.

GraphADT (interface)

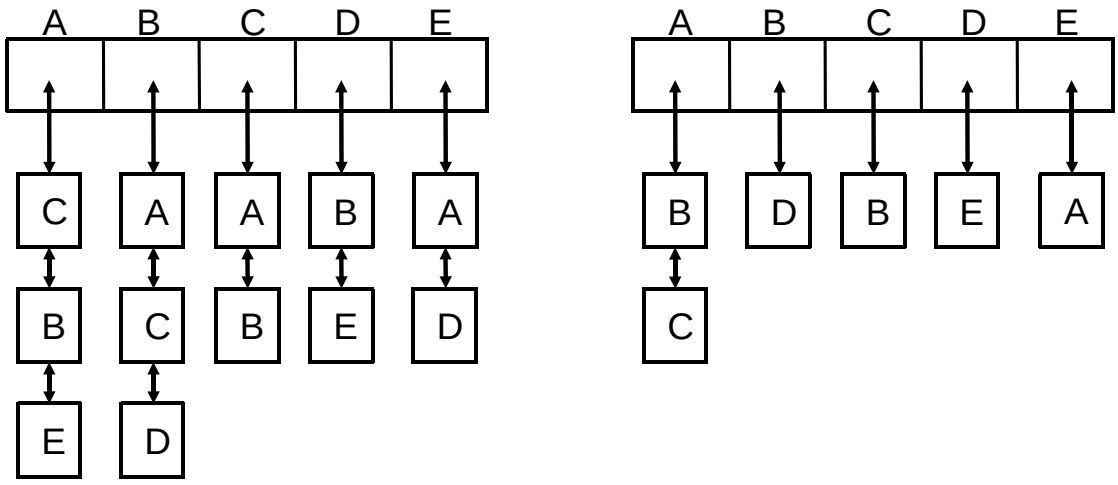
void addVertex (T vertex)	Adds a vertex to this graph
void removeVertex (T vertex)	Removes a single vertex with the given value from this graph
void addEdge (T vertex1, T vertex2)	Inserts an edge between two vertices of this graph
void removeEdge (T vertex1, T vertex2)	Removes an edge between two vertices of this graph
Iterator iteratorBFS(T startVertex)	Returns a breadth first iterator starting with the given vertex
Iterator iteratorDFS(T startVertex)	Returns a depth first iterator starting with the given vertex
boolean isConnected()	Returns true if this graph is connected, false otherwise

Zenbait kontzeptu

- Grafoak vs zuhaitzak: grafoa zuhaitza baino kontzeptu orokorragoa da, ez baita kontuan hartzen zuhaitzen murriztapen hau:
  - Adabegi bakoitzak aita bakarra du (erroak izan ezik, zeinak ez baitu aitarik)
- Grafoetan ez dago errorik, eta adabegi bakoitza konektatua egon daiteke gainontzeko n-1 adabegietara (gehienez).
- Grafo bat osoa dela esaten da adabegiak konektatzen dituzten arkuen kopurua maximoa baldin bada.

Grafoen adierazpideak

- Auzokidetasun-zerrendak



Grafoen adierazpideak

- Auzokidetasun-zerrendak

```
public class GraphAL<T> implements GraphADT<T>
{ protected final int DEFAULT_CAPACITY = 10;
   protected int numVertices;   // number of vertices in the graph
   protected LinkedList<Integer>[] adjList;   // adjacency list
   protected T[] vertices;   // values of vertices

  public GraphAL()  {  // Eraikitzailea
```

```
  }
```


Grafoen adierazpideak

- Auzokidetasun-matrizea

	A	B	C	D	E
A	0	1	1	0	1
B	1	0	1	1	0
C	1	1	0	0	0
D	0	1	0	0	1
E	1	0	0	1	0

	A	B	C	D	E
A	0	1	1	0	0
B	0	0	0	1	0
C	0	1	0	0	0
D	0	0	0	0	1
E	1	0	0	0	0

Grafoen adierazpideak

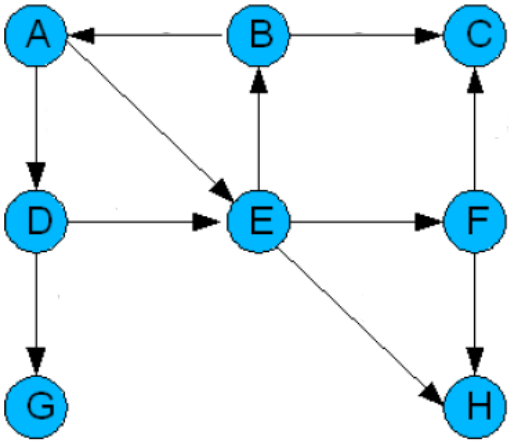
- Auzokidetasun-matrizea

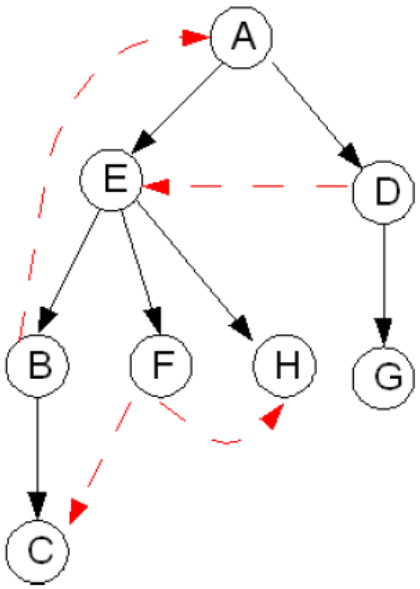
```
public class Graph<T> implements GraphADT<T>
{ protected final int DEFAULT_CAPACITY = 10;
   protected int numVertices;   // number of vertices in the graph
   protected boolean[][] adjMatrix;   // adjacency matrix
   protected T[] vertices;   // values of vertices

public Graph()  {  // Eraikitzailea
```

```
 }
```


Zabalerako ibilbidea





Zuhaitzen mailakako ibilbidearen tankerakoa

Zabalerako ibilbidea

```
proc RecorridoEnAnchura (G = (N, A))
  for cada v ∈ N loop marca(v)← falso end loop
  for cada v ∈ N loop
    if ¬ marca(v) then MarcaAncho(v)

proc MarcaAncho (v)
  C ← new Cola
  marca(v) ← verdadero
  C.insert(v)
  while ¬ C.is_empty() loop
    u ← C.remove_first()
    for cada w ∈ N adyacente de u loop
      if ¬ marca(w) then
        marca(w) ← verdadero
        C.insert(w)
  end while
```


Afaria, biltzarrean (1)

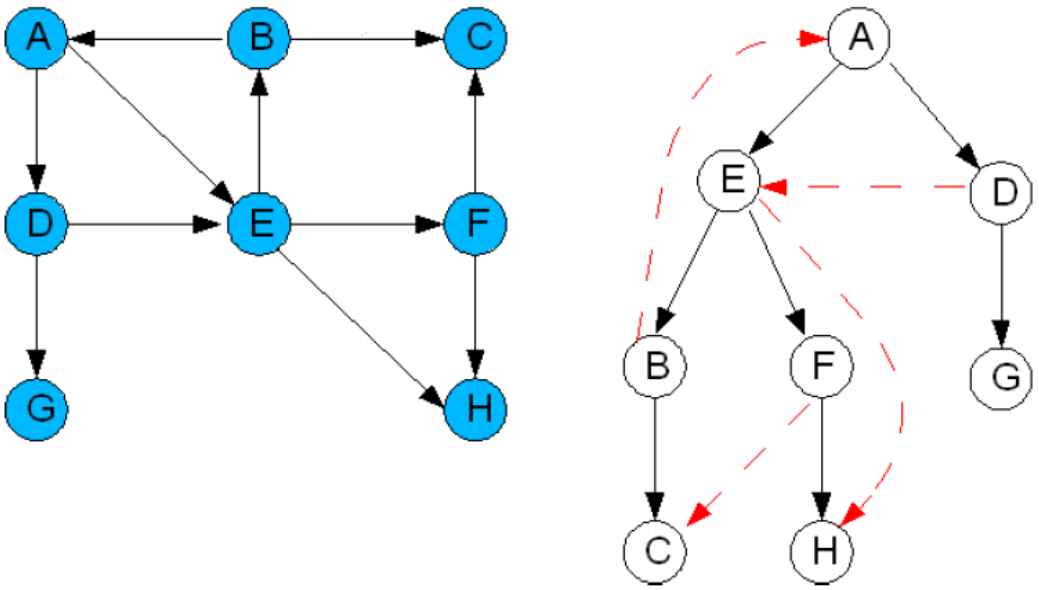
```
func Convención (G = (N, A)) return boolean
  es_posible ← true {valor inicial}
  for cada v ∈ N loop marca(v) ← false end for
  for cada v ∈ N loop
    if not marca(v) then Marca_Comedor(v, true, es_posible) end if
    if not es_posible then return false end if
  end for
  return true
```

Adibidea: afaria, biltzarrean. *Biltzar batean, afari bat antolatu behar dute, eta bi jangela dituzte horretarako. Partehartzaile asko haserretuta daude, eta ez dute jangela berean egon nahi. Posible al da partehartzaileak bi jangela horietan banatzea, haserre daudenak jangela desberdinetan jarrita?*

Afaria, biltzarrean (2)

```
proc MARCA_COMEDOR(v, opción, resultado)
  C ← Cola_vacía()
  marca(v) ← true
  comedor(v) ← opción
  C.añadir(v)
  while not C.vacía() loop
      u ← C.retirar_primero()
      for cada w adyacente de u loop
        if not marca(w) then
          marca(w) ← true
          comedor(w) ← not comedor(u)   {llevar a w a otro comedor}
          C.añadir(w)
        else   {w está marcado, y por tanto asignado a un comedor.}
               {Si es el mismo que el de u entonces no es posible repartirlos}
          if comedor(u) = comedor(w) then
            resultado ← false
            return
```


Sakonerako ibilbidea



Sakonerako ibilbidea

```
proc RecorridoEnProfundidad (G = (N, A))
  for cada v ∈ N loop marca(v)← falso end loop
  for cada v ∈ N loop
    if ¬ marca(v) then MarcaProf(v)

proc MarcaProf (v)
  marca(v)← verdadero
  for cada w ∈ N adyacente de v loop
    if ¬ marca(w) then MarcaProf(w)
```

Ariketa: 11. gardenkiko algoritmoan zein aldaketa SOTIL egingo zenuke, sakonerako ibilbide hau lortzeko

Sakonerako ibilbidea (eskema orokorra)

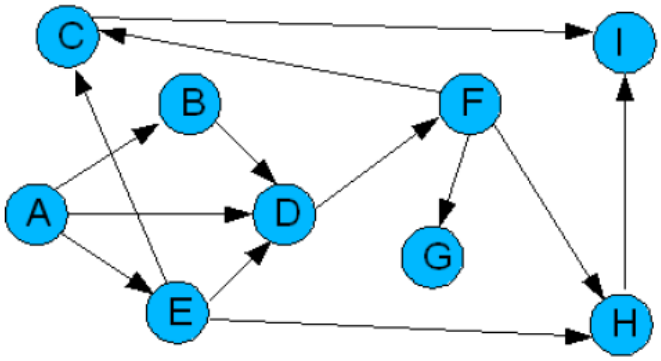
```
proc Marca_Prof_Gen(v)
  [Procesar v en primera visita]
  [Como en preorden]
  marca(v) ← cierto
  for cada w ∈ N adyacente de v loop
    [Procesar arista (v, w)]
    if ¬ marca(w) then
      [Procesar arista (v, w) del árbol]
      Marca_Prof_Gen(w)
      [Procesar v al regreso de procesar w]
      [Como en inorden]
    else
      [Procesar arista (v, w). NO es del árbol]
  end for
  [Procesar v al abandonarlo]
  [Como en postorden]
```

Zabalerako eta sakonerako ibilbideen analisia

- Izanik *n* grafoko adabegien kopurua eta *a* arkuen kopurua:
  - O(*n*+*a*) baldin eta grafoa auzokidetasun listekin adierazten bada.
  - O(n^2) baldin eta grafoa auzokidetasun matrizeekin adierazten bada.

Ordenazio topologikoa

G= (N,A) zuzendutako grafo ez-zikliko baten ordenazio topologikoak berekin dakar grafoko adabegien lista bat, non A-ko (u,v) arku bakoitzerako u adabegia v adabegiaren aurretik agertuko den emaitzako listan.



Adibideko grafoaren zenbait ordenazio topologiko:
A B E D F C H I G
A E B D F G H C I

Ordenazio topologikoa

```
func ORDENACIÓN_TOPOLÓGICA (G = (N, A)) return Lista_de_nodos
  L ← new Lista
  for cada v ∈ N loop marca(v) ← falso end loop
  for cada v ∈ N loop
    if ¬ marca(v) then ORDENTOPO(v, L)
  end for
  return L

proc ORDENTOPO (v, L)
  marca(v) ← verdadero
  for cada w ∈ N adyacente de v loop
    if ¬ marca(w) then ORDENTOPO(w, L)
  end for
  L.insert_first(v)
```


Irakurgaiak

- Barne-txostena:
  - "Recorridos de grafos: Teoría y aplicaciones" Jesús Bermúdez de Andrés
- [Lewis, Chase 2010]
  - 13. kapitulua