Agenda - Grafos

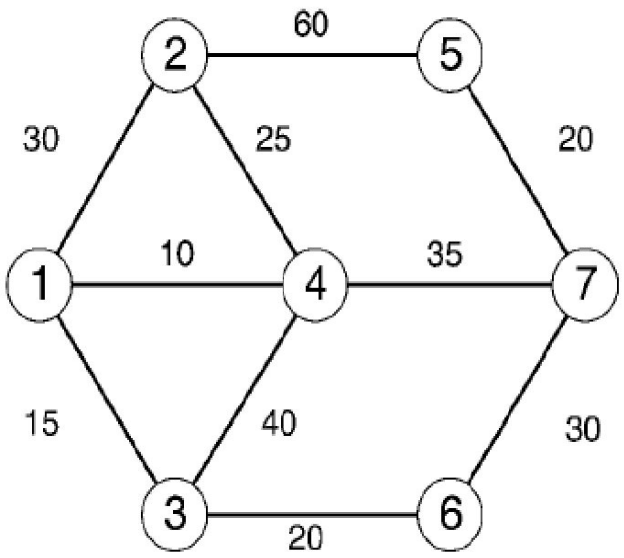
1. Recorridos
2. Aplicaciones de los recorridos

Agenda - Grafos

- Recorridos
  - en profundidad: DFS (Depth First Search)
  - en amplitud: BFS (Breath First Search)
  - Bosque de expansión DFS

Problema: El Guía de Turismo

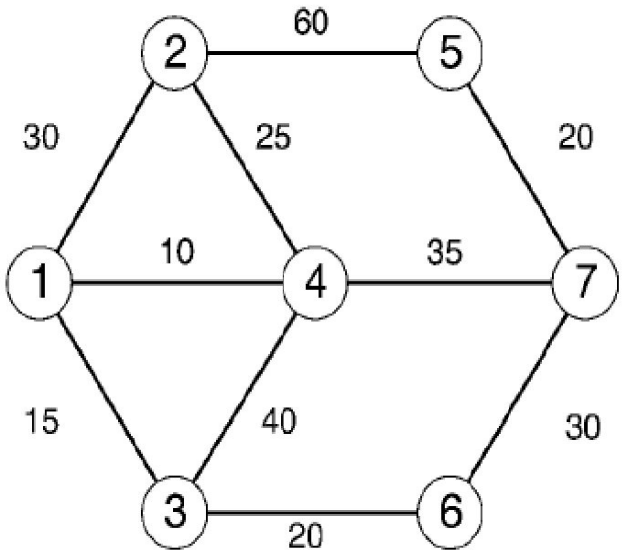
El señor H es un guía de turismo de la ciudad de Buenos Aires. Su trabajo consiste en mostrar a grupos de turistas diferentes **puntos de interés** de la ciudad.
Estos puntos de interés están **conectados por rutas en ambos sentidos**. Dos puntos de interés vecinos tienen un servicio de bus que los conecta, con una limitación en el **número máximo de pasajeros** que puede transportar. No es siempre posible para el señor H transportar de una única vez a todos los turistas a un destino en particular.



Por ejemplo, consideremos el siguiente mapa con 7 puntos de interés, donde las aristas representan las rutas y el peso de ellas representa el límite máximo de pasajeros a transportar por el servicio de bus. Su misión es indicarle al Sr. H cuál es el menor número de viajes que deberá realizar para llevar al grupo de turistas de un origen a un destino.

Problema: El Guía de Turismo

El señor H es un guía de turismo de la ciudad de Buenos Aires. Su trabajo consiste en mostrar a grupos de turistas diferentes **puntos de interés** de la ciudad.
Estos puntos de interés están **conectados por rutas en ambos sentidos**. Dos puntos de interés vecinos tienen un servicio de bus que los conecta, con una limitación en el **número máximo de pasajeros** que puede transportar. No es siempre posible para el señor H transportar de una única vez a todos los turistas a un destino en particular.

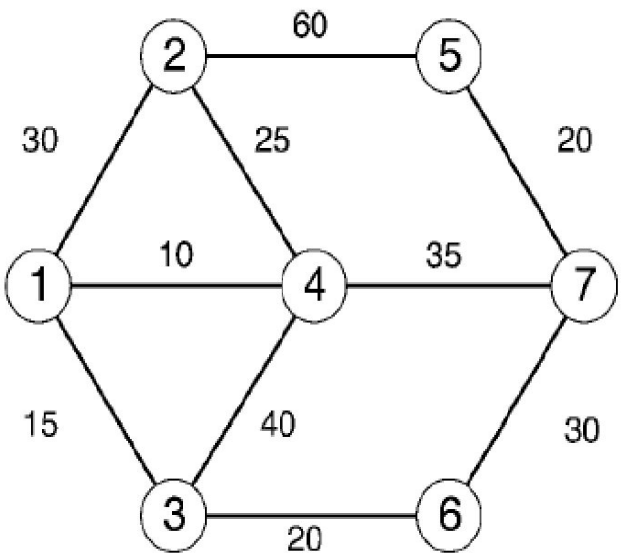


Por ejemplo, consideremos el siguiente mapa con 7 puntos de interés, donde las aristas representan las rutas y el peso de ellas representa el límite máximo de pasajeros a transportar por el servicio de bus. Su misión es indicarle al Sr. H cuál es el menor número de viajes que deberá realizar para llevar al grupo de turistas de un origen a un destino.

*En este ejemplo, el señor H debe transportar a **99 turistas** del punto **1** al punto **7**.*

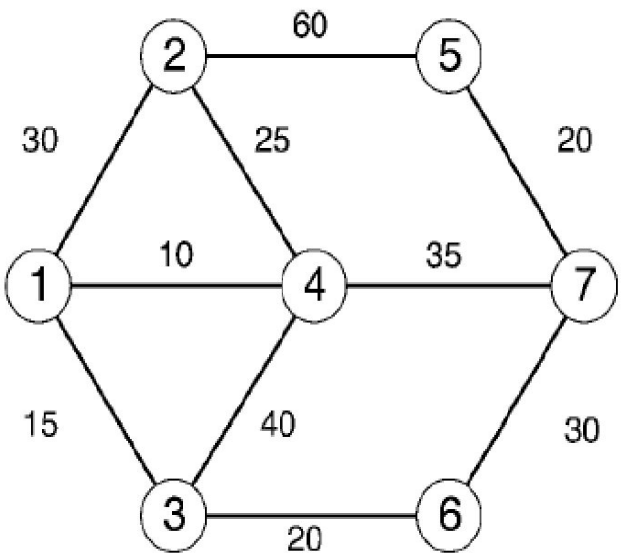
Veamos cuáles son los recorridos posibles y elijamos el que implique realizar el menor número de viajes.

Problema: El Guía de Turismo



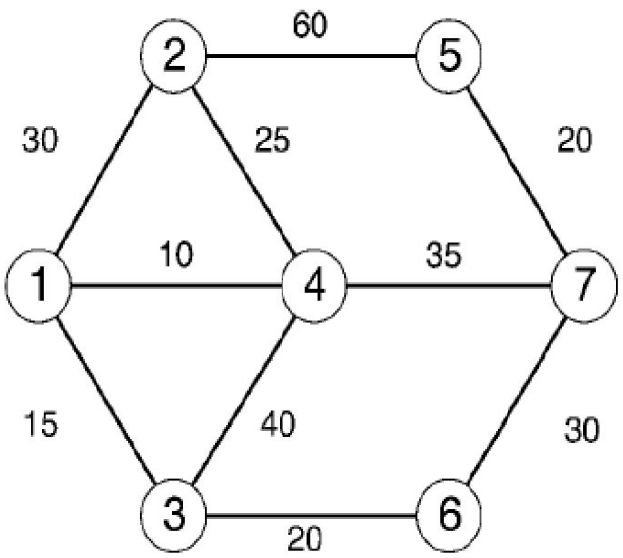
vértices del recorrido	cant. turistas/viaje	cant. de viajes
1 2 4 3 6 7		
1 2 4 7		
1 2 5 7		
1 3 4 2 5 7		
1 3 4 7		
1 3 6 7		
1 4 2 5 7		
1 4 3 6 7		
1 4 7		

Problema: El Guía de Turismo



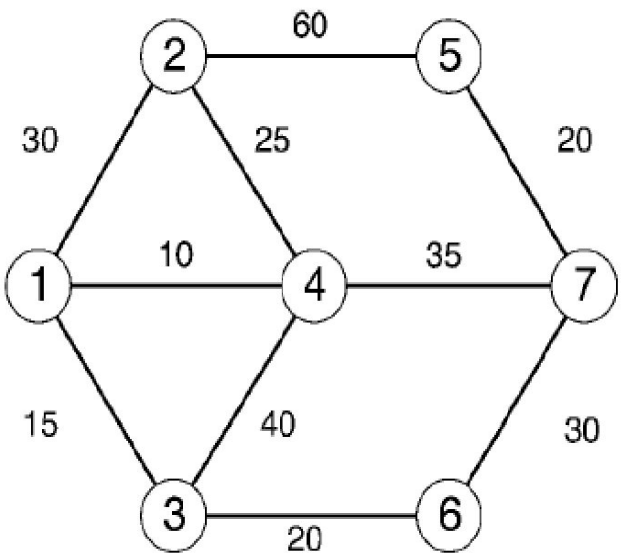
vértices del recorrido	cant. turistas/viaje	cant. de viajes
1 2 4 3 6 7	20	
1 2 4 7	25	
1 2 5 7	20	
1 3 4 2 5 7	15	
1 3 4 7	15	
1 3 6 7	15	
1 4 2 5 7	10	
1 4 3 6 7	10	
1 4 7	10	

Problema: El Guía de Turismo



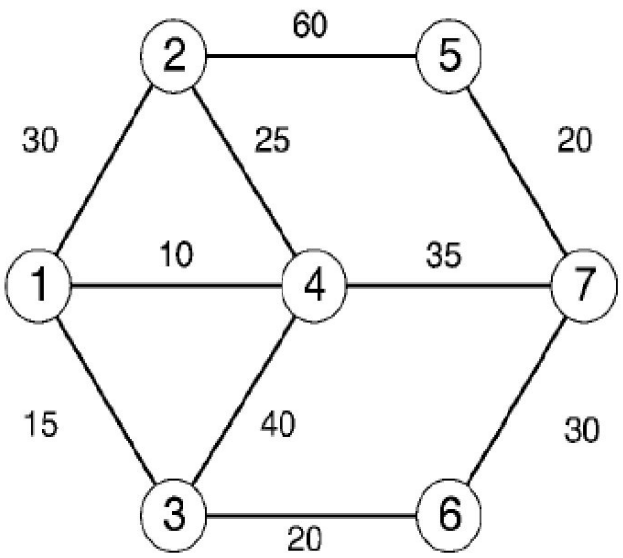
vértices del recorrido	cant. turistas/viaje	cant. de viajes
1 2 4 3 6 7	20	6
1 2 4 7	25	5
1 2 5 7	20	6
1 3 4 2 5 7	15	8
1 3 4 7	15	8
1 3 6 7	15	8
1 4 2 5 7	10	11
1 4 3 6 7	10	11
1 4 7	10	11

Problema: El Guía de Turismo



vértices del recorrido	cant. turistas/viaje	cant. de viajes
1 2 4 3 6 7	20	6
1 2 4 7	**25**	**5**
1 2 5 7	20	6
1 3 4 2 5 7	15	8
1 3 4 7	15	8
1 3 6 7	15	8
1 4 2 5 7	10	11
1 4 3 6 7	10	11
1 4 7	10	11

Problema: El Guía de Turismo

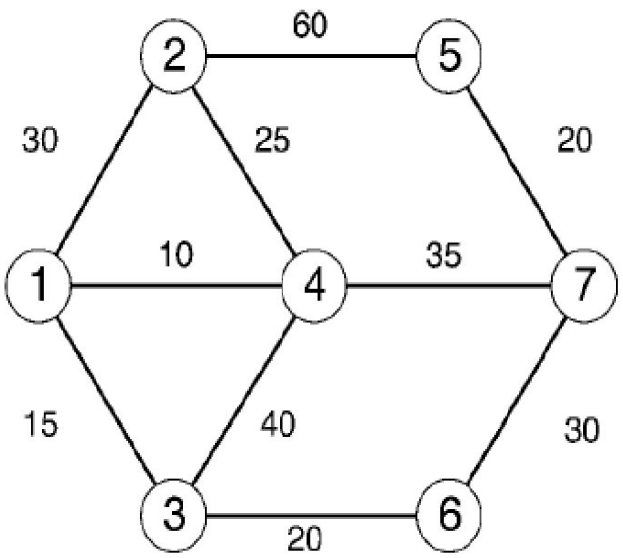


vértices del recorrido	cant. turistas/viaje	cant. de viajes
1 2 4 3 6 7	20	6
1 2 4 7	**25**	**5**
1 2 5 7	20	6
1 3 4 2 5 7	15	8
1 3 4 7	15	8
1 3 6 7	15	8
1 4 2 5 7	10	11
1 4 3 6 7	10	11
1 4 7	10	11

*Entonces, para transportar a los **99 turistas** del punto 1 al punto 7, se necesitarán 5 viajes, eligiendo la ruta:*

1 - 2 - 4 - 7

Problema: El Guía de Turismo



vértices del recorrido	cant. turistas/viaje	cant. de viajes
1 2 4 3 6 7	20	6
1 2 4 7	**25**	**5**
1 2 5 7	20	6
1 3 4 2 5 7	15	8
1 3 4 7	15	8
1 3 6 7	15	8
1 4 2 5 7	10	11
1 4 3 6 7	10	11
1 4 7	10	11

*Entonces, para transportar a los **99 turistas** del punto 1 al punto 7, se necesitarán 5 viajes, eligiendo la ruta:*

1 - 2 - 4 - 7

En cada viaje el servicio de bus puede transportar como máximo a 25 pasajeros, 24 turistas + al señor H, en los cuatro primeros viajes transporta a 96 turistas y en el último a los restantes 3.

Recorrido en profundidad

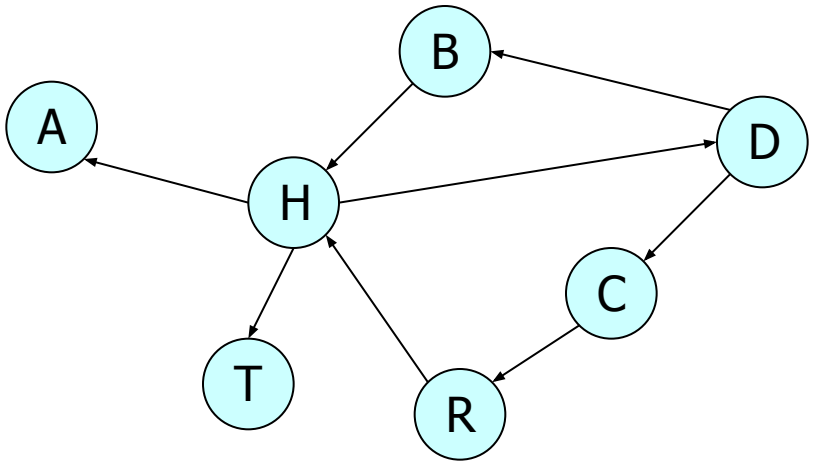
Depth-First Search (DFS)

→ Generalización del recorrido preorden de un árbol.

***Estrategia**:*

- *Partir de un vértice determinado **v**.*
- *Cuando se visita un nuevo vértice, explorar cada camino que salga de él.*
- *Hasta que no se haya finalizado de explorar uno de los caminos no se comienza con el siguiente.*
- *Un camino deja de explorarse cuando se llega a un vértice ya visitado.*
- *Si existían vértices no alcanzables desde **v** el recorrido queda incompleto; entonces, se debe seleccionar algún vértice como nuevo vértice de partida, y repetir el proceso.*

Recorrido en profundidad: DFS



Si tomamos como vértice de partida a D

Se Muestra:

D C R H T A B

Recorrido en profundidad: DFS

***Esquema recursivo**: dado G = (V , E)*

1. *Marcar todos los vértices como no visitados.*

2. *Elegir vértice **u** como punto de partida.*

3. *Marcar **u** como visitado.*

4. *$\forall$ **v** adyacente a **u**,**(u,v)** $\in$ E, si **v** no ha sido visitado, repetir recursivamente (3) y (4) para **v**.*

- *Finalizar cuando se hayan visitado todos los nodos alcanzables desde u.*

- *Si desde **u** no fueran alcanzables todos los nodos del grafo: volver a (2), elegir un nuevo vértice de partida **v** no visitado, y repetir el proceso hasta que se hayan recorrido todos los vértices.*

Recorrido en profundidad: DFS

```
dfs (v: vértice)
  marca[v]:= visitado;
  para cada nodo w adyacente a v
            si w no está visitado
            dfs(w);


main:dfs (grafo)
  inicializar marca en false (arreglo de booleanos);
  para cada vértice v del grafo
          si v no está visitado
           dfs(v);
```

Recorrido DFS: Tiempo de ejecución

- *G (V, E) se representa mediante listas de adyacencia.*
- *El método **dfs(v)** se aplica únicamente sobre vértices no visitados → sólo una vez sobre cada vértice.*
- ***dfs(v)** depende del número de vértices adyacentes que tenga (longitud de la lista de adyacencia).*

  *→ el tiempo de todas las llamadas a **dfs(v)**: $O(|E|)$*
- *añadir el tiempo asociado al bucle de main_dfs(grafo): $O(|V|)$).*

  ⇒ Tiempo del recorrido en profundidad es $O(|V|+|E|)$.

Recorrido en amplitud: BFS

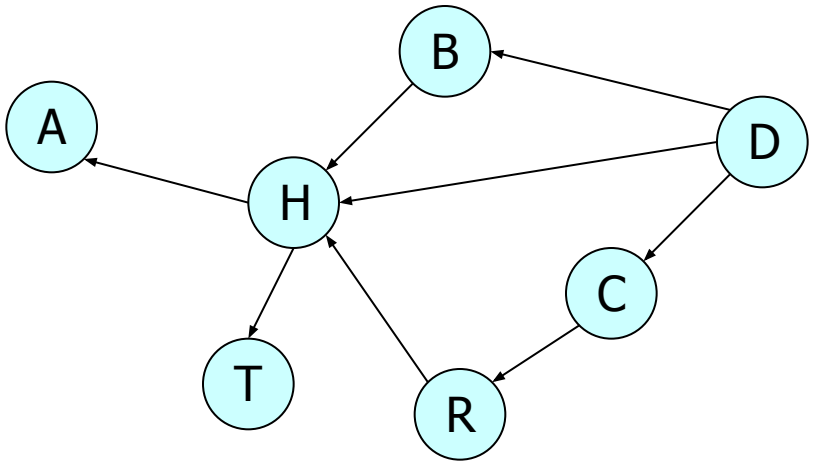
→ *Generalización del recorrido por niveles de un árbol.*

***Estrategia**:*

- *Partir de algún vértice **u**, visitar **u** y, después, visitar cada uno de los vértices adyacentes a **u**.*

- *Repetir el proceso para cada nodo adyacente a **u**, siguiendo el orden en que fueron visitados.*

Recorrido en amplitud: BFS



Si tomamos como vértice de partida a D

Se Muestra:

D C H B R T A

Cola:

D C H B R T A

Recorrido en amplitud: BFS

***Esquema iterativo**: dado G = (V , E)*

*1. Encolar el vértice origen **u**.*

*2. Marcar el vértice **u** como visitado.*

3. Procesar la cola.

*4. Desencolar **u** de la cola*

*5. ∀ adyacente a **u**,**(u,v)** ∈ E,*

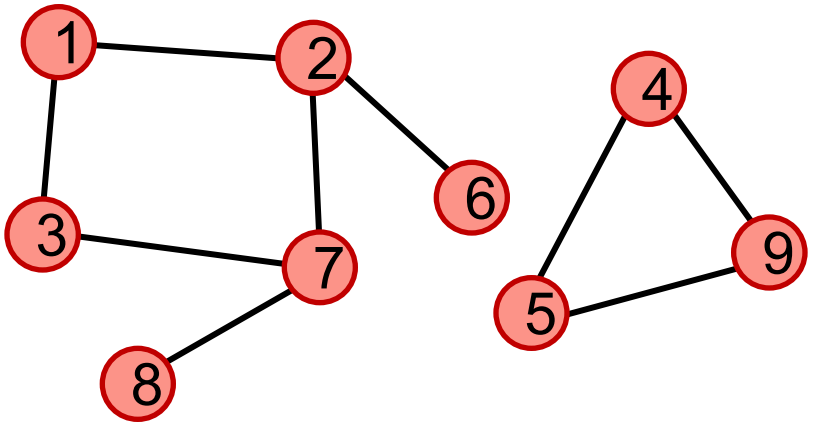
*6. si **v** no ha sido visitado*

*7. encolar y visitar **v***

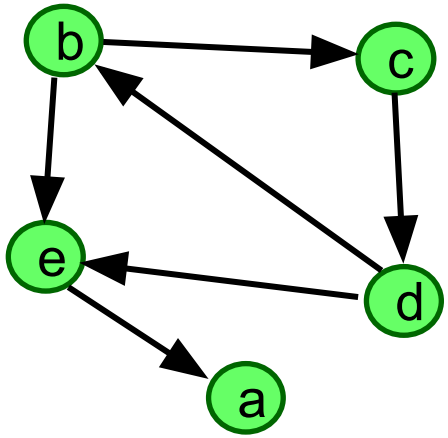
- *Si desde **u** no fueran alcanzables todos los nodos del grafo: volver a (1), elegir un nuevo vértice de partida no visitado, y repetir el proceso hasta que se hayan recorrido todos los vértices*
  - *Costo T(|V|,|E|) es de O(|V|+|E|)*

Bosque de expansión del DFS

- *El recorrido **no es único**: depende del nodo inicial y del orden de visita de los adyacentes.*
- *El orden de visita de unos nodos a partir de otros puede ser visto como un árbol: **árbol de expansión (o abarcador) en profundidad asociado al grafo.***
- *Si aparecen varios árboles: **bosque de expansión (o abarcador) en profundidad**.*

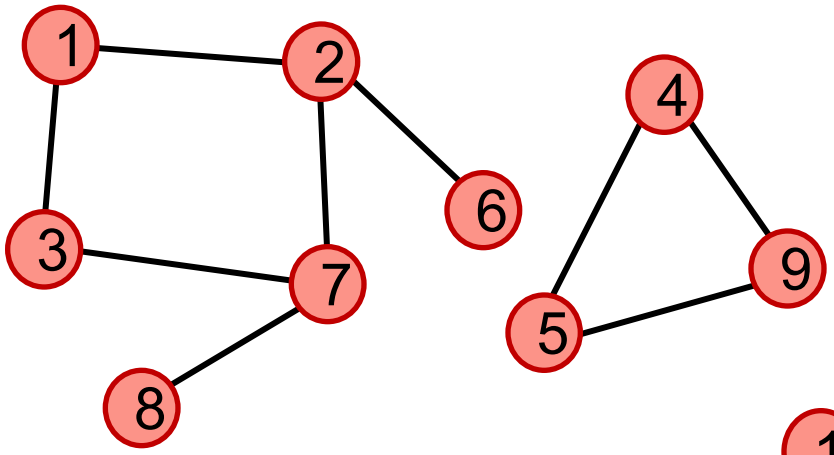


Grafo no dirigido y no Conexo



Grafo dirigido y no fuertemente Conexo

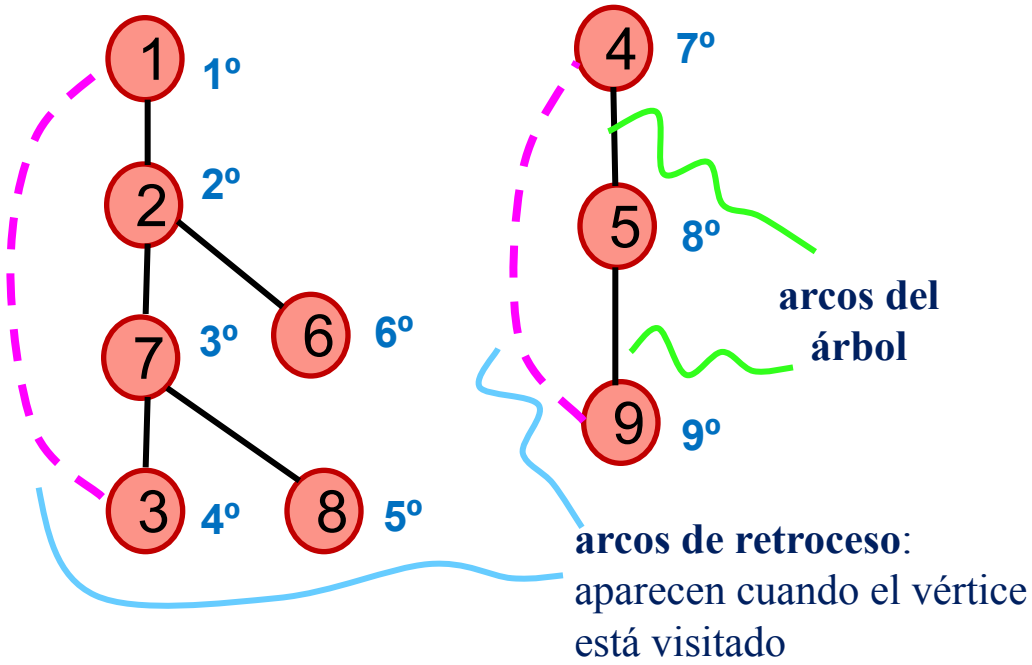
Bosque de expansión del DFS



Grafo no dirigido y no Conexo

En grafos no dirigidos, existen dos tipos de arcos en DFS:

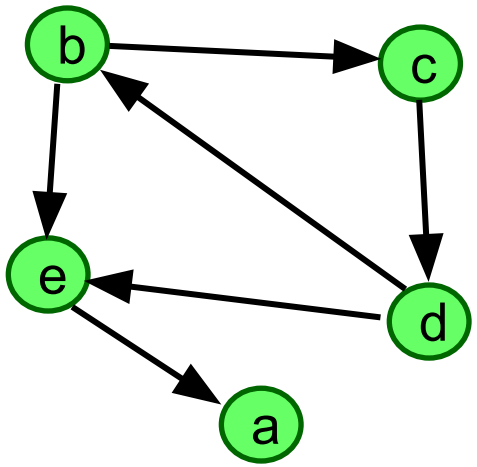
- Tree edges
- Back edges



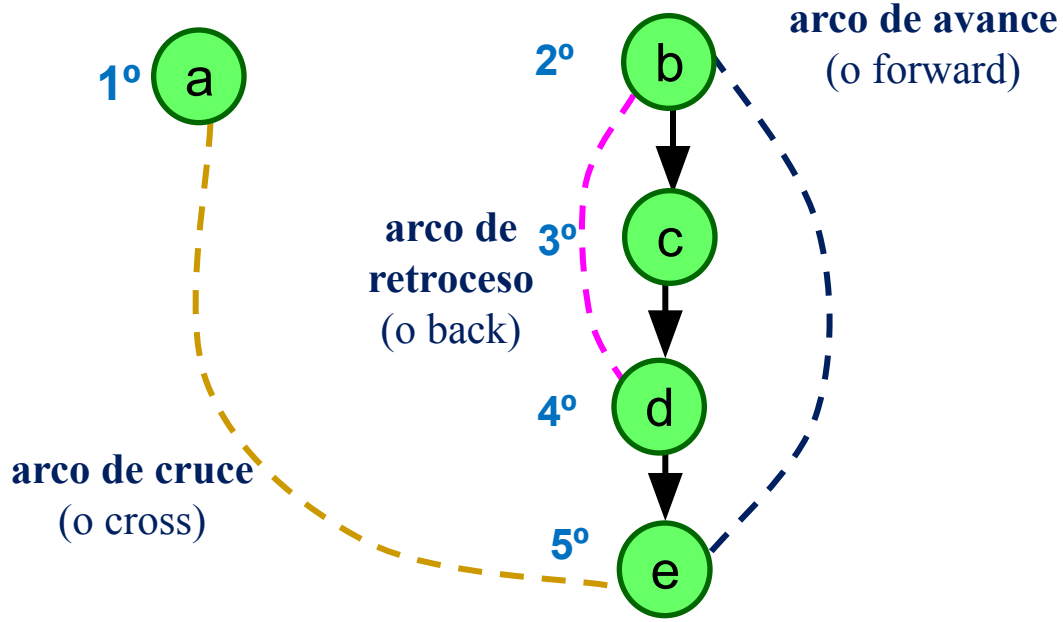
Bosque de expansión del DFS

En grafos dirigidos, existen cuatro tipos de aristas en DFS:

- Tree edges
- Back edges
- Cross edges
- Forward edges



Grafo dirigido y no fuertemente Conexo



Bosque de expansión, empezando el recorrido en el vértice **a**

Bosque de expansión del DFS

Clasificación de los arcos de un grafo dirigido en el bosque de expansión de un DFS.

- *Arcos **tree** (del **árbol**): son los arcos en el bosque depth-first-search, arcos que conducen a vértices no visitados durante la búsqueda.*

- *Arcos **forward**: son los arcos u→v que no están en el bosque, donde v es descendiente, pero no es hijo en el árbol.*

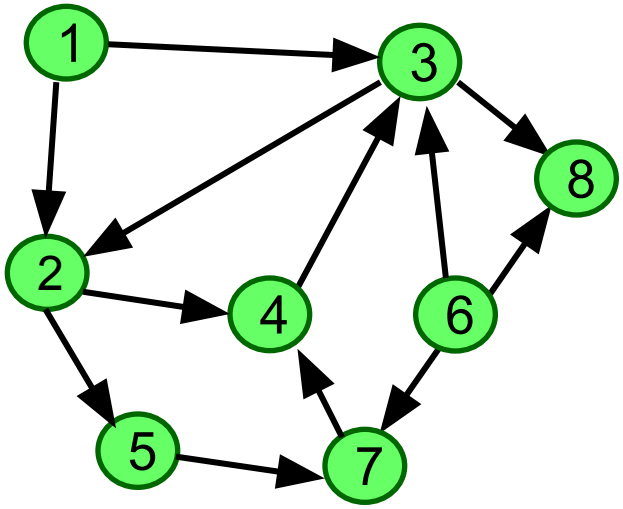
- *Arcos **backward**: son los arcos u→v, donde v es antecesor en el árbol. Un arco de un vértice a si mismo es considerado un arco back.*

- *Arcos **cross**: son todos los otros arcos u→v, donde v no es ni antecesor ni descendiente de u. Son arcos que pueden ir entre vértices del mismo árbol o entre vértices de diferentes árboles en el bosque depth-first-search.*

Bosque de expansión del DFS

Ejercicio 1:

Dado el siguiente grafo dirigido, en el bosque abarcador del DFS realizado a partir del vértice 1: 1, 2, 4, 3, 8, 5, 7, 6, habrá ……

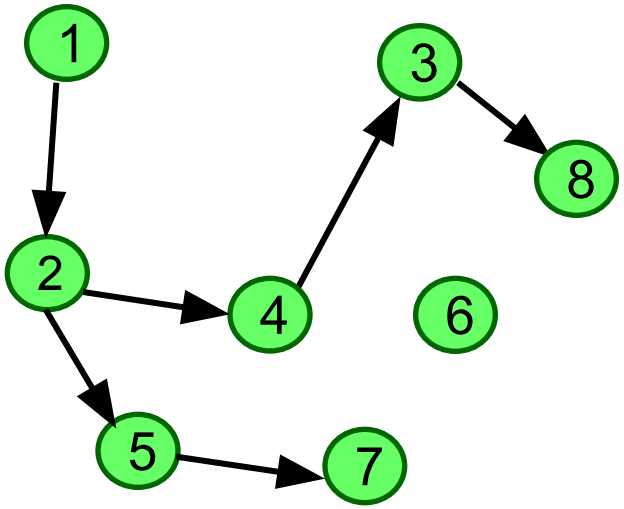


a) 1 arco de avance
b) 2 arcos de avance
c) más de 2 arcos de avance
d) Ninguna de las opciones

Bosque de expansión del DFS

Ejercicio 1:

Dado el siguiente grafo dirigido, en el bosque abarcador del DFS realizado a partir del vértice 1: 1, 2, 4, 3, 8, 5, 7, 6, habrá ……

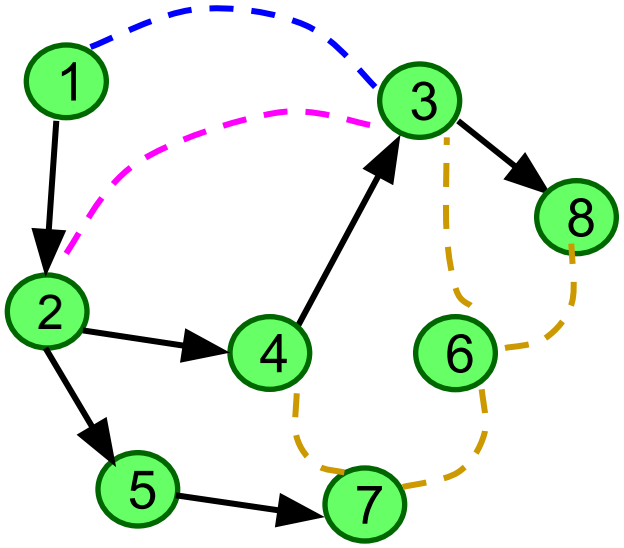


a) 1 arco de avance
b) 2 arcos de avance
c) más de 2 arcos de avance
d) Ninguna de las opciones

Bosque de expansión del DFS

Ejercicio 1:

Dado el siguiente grafo dirigido, en el bosque abarcador del DFS realizado a partir del vértice 1: 1, 2, 4, 3, 8, 5, 7, 6, habrá ……

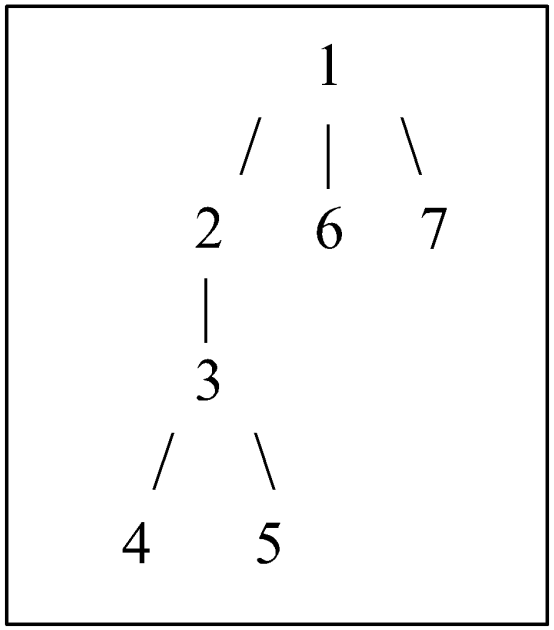


a) 1 arco de avance
b) 2 arcos de avance
c) más de 2 arcos de avance
d) Ninguna de las opciones

Bosque de expansión del DFS

Ejercicio 2:

El recorrido en profundidad de un grafo G no dirigido ha producido el árbol que se muestra en la figura, en el que cada nodo está numerado siguiendo el orden de visita del recorrido en profundidad.



Cuál de las siguientes afirmaciones es correcta?

(a) El nodo 6 es adyacente al nodo 4.

(b) El nodo 2 puede ser adyacente al nodo 5, y el nodo 4 puede ser adyacente al nodo 1.

(c) El nodo 6 y 7 no son adyacentes, y el nodo 5 y el nodo 7 si lo son.

(d) El nodo 1 sólo puede ser adyacente a los nodos 2, 6 y 7.

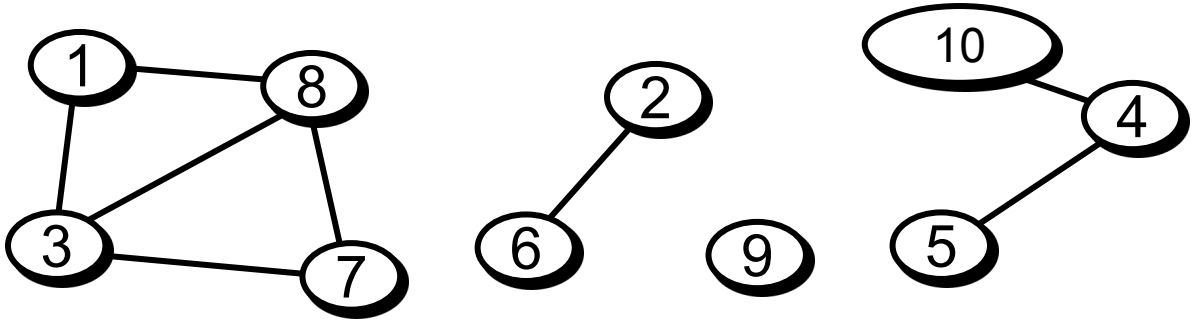
(e) Se trata de un grafo fuertemente conexo.

Agenda - Grafos

1. Recorridos
2. Aplicaciones de los recorridos

Aplicaciones del DFS

- **Problema 1:** encontrar las componentes conexas de un grafo no dirigido.



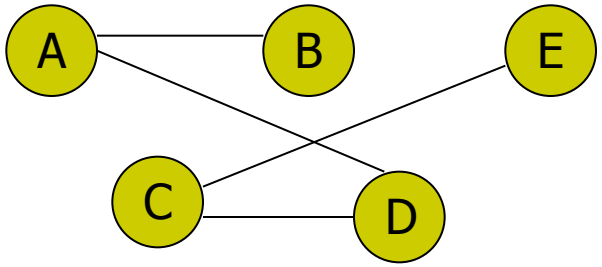
- **Problema 2: prueba de aciclicidad.** Dado un grafo (dirigido o no dirigido) comprobar si tiene algún ciclo o no.
- **Problema 3:** encontrar las componentes fuertemente conexas de un grafo dirigido.

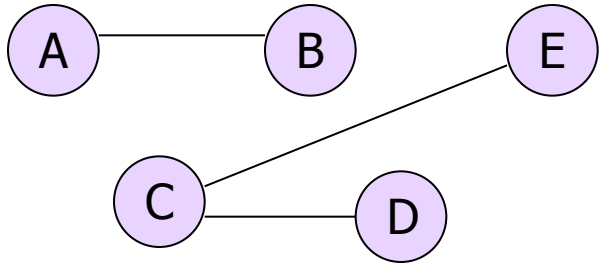
Aplicaciones del DFS

- **Problema 1:** Encontrar las componentes conexas de un grafo no dirigido
  - Si el grafo es conexo
    - Un recorrido desde cualquier vértice
    - Visitará a TODOS los vértices del grafo
  - Si no lo es
    - Partiendo de un vértice, tendremos una componente conexa → conjunto de vértices recorrido
    - Para descubrir otras
      - Repetir recorrido desde un vértice no visitado
      - Hasta que todos los vértices hayan sido visitados

Aplicaciones del DFS

- **Problema 1: Encontrar las componentes conexas de un grafo no dirigido**





Recorrido desde E

E C D A B

Conjunto recorrido = Conjunto de Vértices **Es CONEXO**

Recorrido desde E

E C D

Componente Conexa 1

Recorrido desde B

B A

Componente Conexa 2

Aplicaciones del DFS

- **Problema 2:** Prueba de aciclicidad
  - **Grafo no dirigido.** Hacer un recorrido dfs. Existe algún ciclo si y sólo si aparece algún arco que no es del árbol de expansión.
  - **Grafo dirigido.** Hacer un dfs. Existe un ciclo si y sólo si aparece algún arco de retroceso.

Orden de complejidad de la prueba de aciclicidad: igual que los recorridos.

- Con matrices de adyacencia: $\mathbf{O(|V|^2)}$.
- Con listas de adyacencia: $\mathbf{O(|V| + |E|)}$.

Aplicaciones del DFS

- **Problema 3:** Encontrar las componentes fuertemente conexas

Una aplicación clásica del depth-first search es descomponer un grafo dirigido en componentes fuertemente conexas (o conectadas).

Una ***componente fuertemente conexa*** de un grafo dirigido G=(V,E) es el conjunto máximo de vértices V' $\subseteq$ V tal que para cada par de vértices u y v en V', existe un camino tanto $u \rightarrow v$ como $v \rightarrow u$.

Aplicaciones del DFS

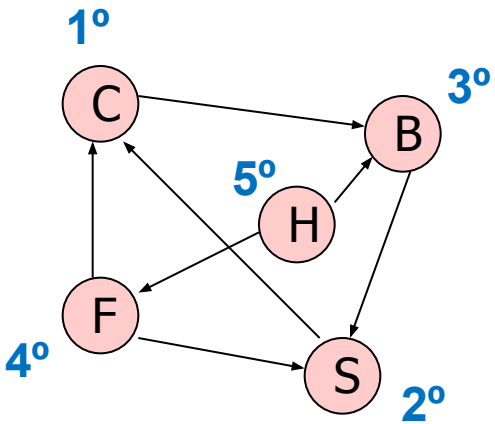
Encontrar las componentes fuertemente conexas de un grafo dirigido: Algoritmo de Kosaraju

Pasos:

1. Aplicar DFS(G) rotulando los vértices de G en post-orden (apilar).
2. Construir el grafo reverso de G, es decir G^R (invertir los arcos).
3. Aplicar DFS (G^R) comenzando por los vértices de mayor rótulo (tope de la pila).
4. Cada árbol de expansión resultante del paso 3 es una componente fuertemente conexa.

Si resulta un único árbol entonces el digrafo es fuertemente conexo.

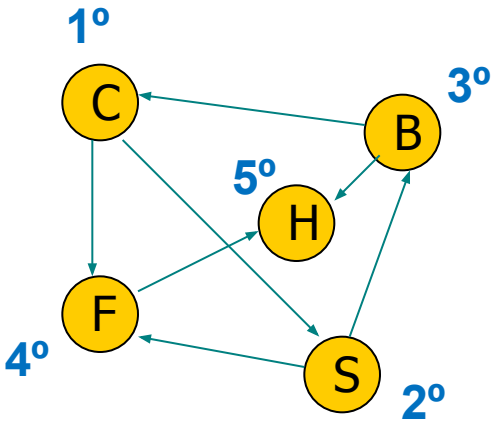
Algoritmo de Kosaraju

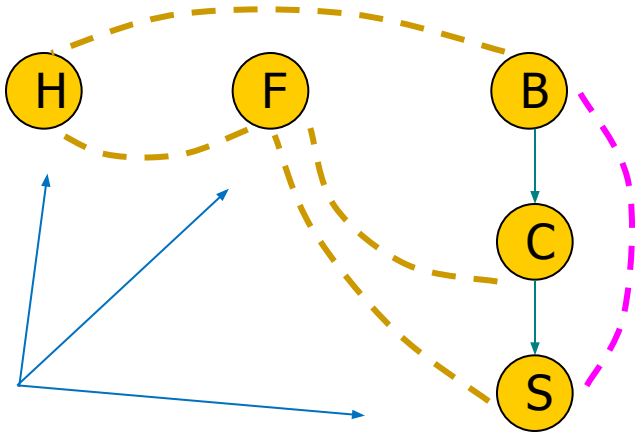


1. Aplicar el recorrido en profundidad, por ejemplo, desde **B** y rotular los vértices en post-orden

2. Construir el grafo reverso G^R

3. Aplicar DFS comenzando de los vértices de mayor rótulo en G^R





Componentes fuertemente conexas

Algoritmo de Kosaraju

Complejidad del algoritmo

- Se realizan dos DFS
- Se recorren todas las aristas una vez para crear el grafo reverso

$O(|V| + |E|)$

Problema: Ir y venir (Come and go)

https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=671&page=show_problem&problem=2938

En una ciudad hay N intersecciones conectadas por calles de una y dos direcciones. Se trata de una ciudad moderna que cuenta con túneles y puentes. Debe ser posible trasladarse entre dos intersecciones cualesquiera V y W.
Su tarea es escribir un algoritmo que lea la organización de las calles de una ciudad y determine si se cumple este requisito de conectividad.

Input

La entrada contiene varios casos de prueba. La primera línea de cada caso de prueba contiene dos enteros N y M, separados por un espacio, indicando el número de intersecciones ($2 \leq N \leq 2000$) y el número de calles ($2 \leq M \leq N\,(N\text{-}1)\,/2$). Las siguientes M líneas describen el sistema de calles, y cada línea describe una calle. La descripción de una calle consiste de tres enteros V, W y P, separados por un espacio, donde V y W son distintos identificadores de una intersección ($1 \leq V, W \leq N, V \neq W$) y P puede ser 1 ó 2; si P=1, la calle es de una sola dirección y el tráfico va de V a W; si P=2, la calle tiene dos direcciones y conecta V y W. Cada par de intersecciones está conectada por a lo sumo una calle.
El último caso de prueba es seguido por una línea que contiene solamente dos números 0, separados por un blanco.

Problema: Ir y venir (Come and go)

Output

Para cada caso de prueba el algoritmo debe imprimir una única línea conteniendo un entero G, donde G es igual a 1 si la condición de conectividad se cumple y 0 en caso contrario.

Sample Input

```
4 5
1 2 1
1 3 2
2 4 1
3 4 1
4 1 2
3 2
1 2 2
1 3 2
3 2
1 2 2
1 3 1
4 2
1 2 2
3 4 2
0 0
```

Sample Output

```
1
1
0
0
```