DFS, Puntos de Articulación, Puentes, SCC, Chinese Postman

Destefanis, Eric

12 de julio de 2019

Contenido

Contenido

Información del DFS

De CLRS

"Depth-first search yields valuable information about the structure of a graph."

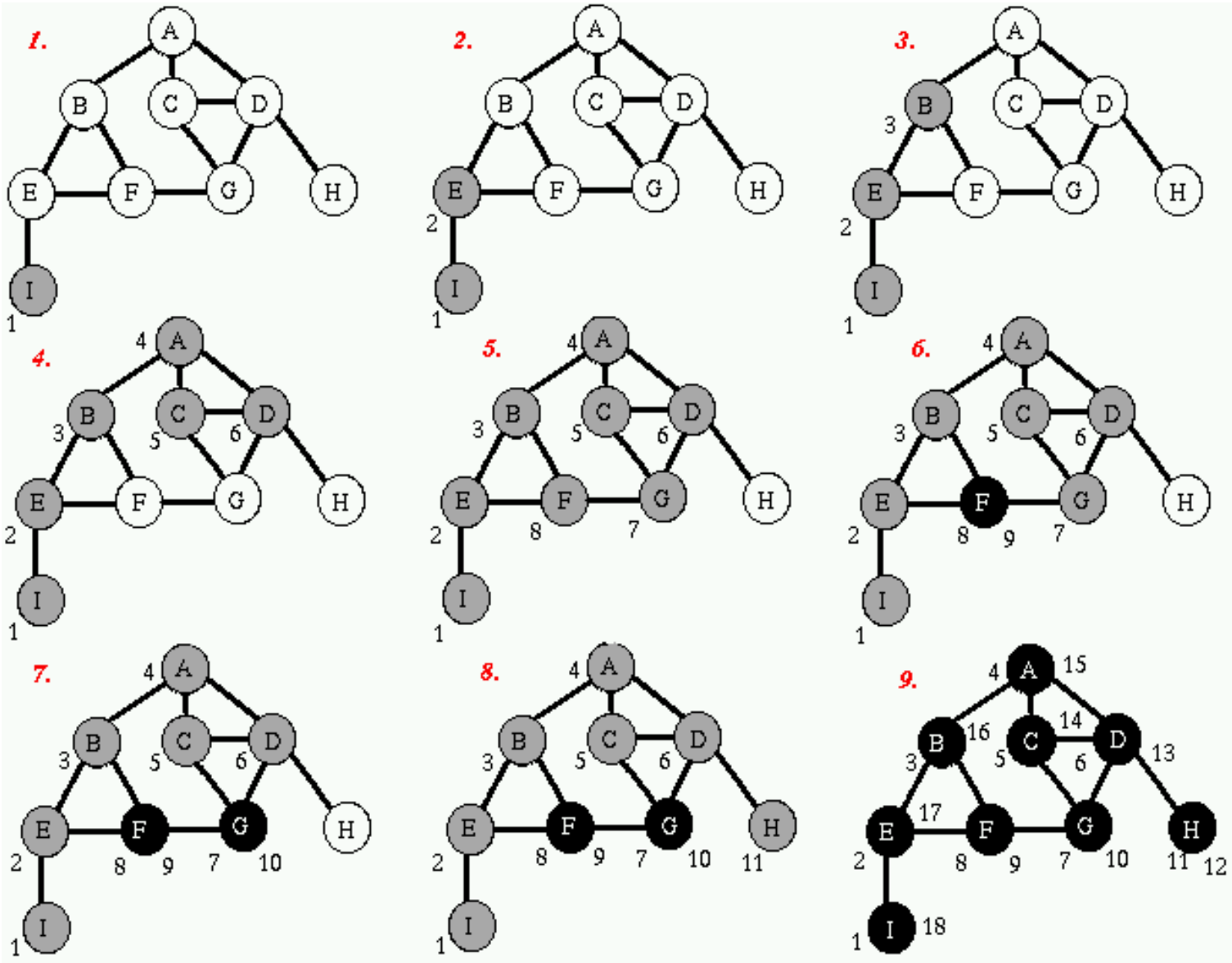
DFS

Durante un DFS, es buena idea siempre tener en cuenta 3 posibles estados para cada nodo. Podriamos representarlos como:

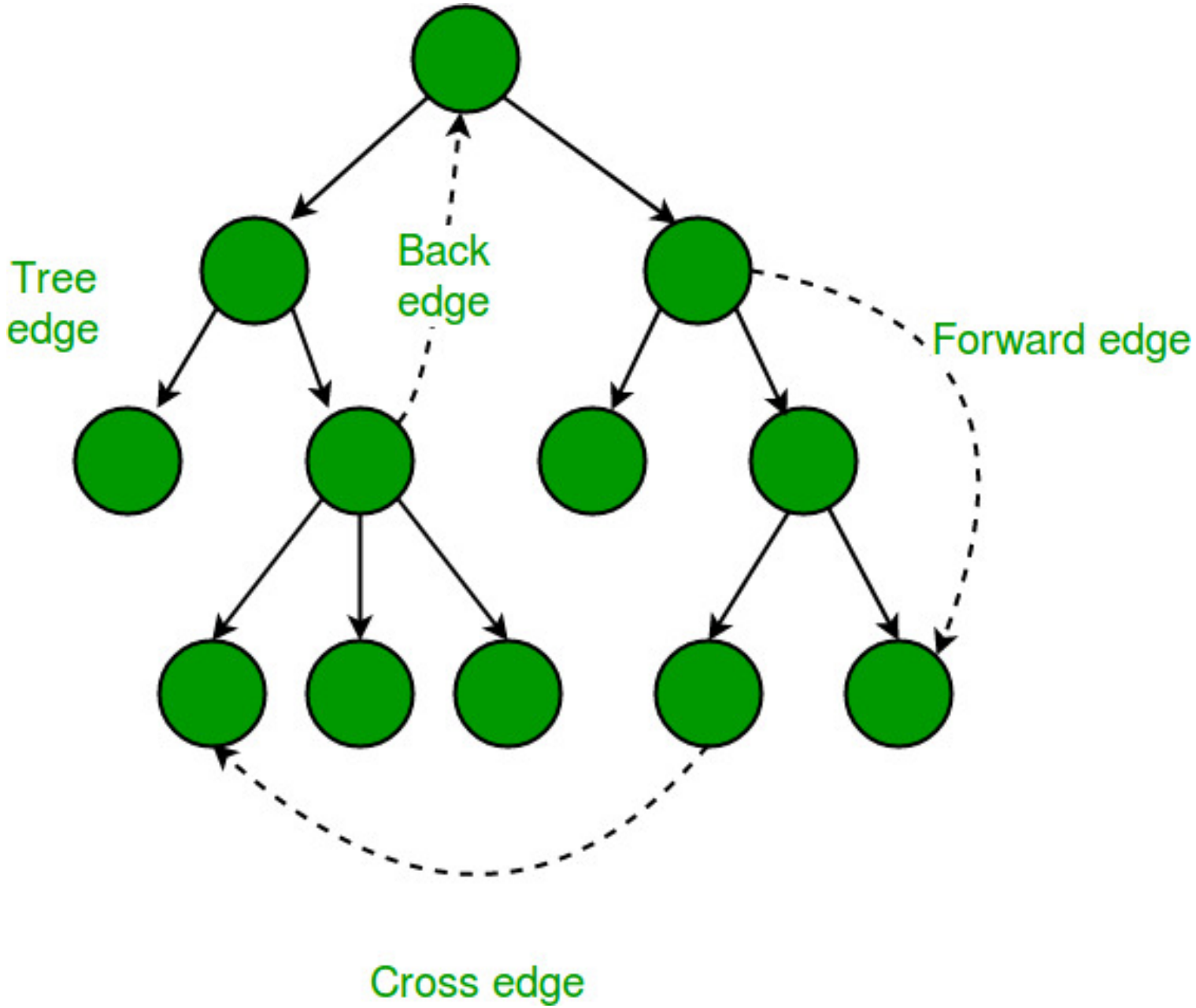
- Blanco: el nodo aun no ha sido visitado.
- Gris: El nodo ha sido visitado, pero aun estamos visitando alguno de sus descendientes.
- Negro: El nodo y sus descendientes ya fueron visitados.

Los nodos empiezan todos pintados de blanco.

Dibujo


1.
A
B
C
D
E
F
G
H
I
1
2.
A
B
C
D
E
F
G
H
I
2
1
3.
A
B
C
D
E
F
G
H
I
3
2
1
4.
A
B
C
D
E
F
G
H
I
4
3
5
6
2
1
5.
A
B
C
D
E
F
G
H
I
4
3
5
6
8
7
2
1
6.
A
B
C
D
E
F
G
H
I
4
3
5
6
8
9
7
2
1
7.
A
B
C
D
E
F
G
H
I
4
3
5
6
8
9
7
10
2
1
8.
A
B
C
D
E
F
G
H
I
4
3
5
6
8
9
7
10
11
2
1
9.
A
B
C
D
E
F
G
H
I
15
4
16
14
13
3
5
6
17
8
9
7
10
11
12
2
18
1

Dibujo



Tipos de aristas

El algoritmo de DFS nos induce una clasificacion de aristas en 4 tipos:

- Tree edge
  - Viaja a un nodo blanco
  - Es con la que se descubre un nodo por primera vez
  - Viaja al hijo del nodo actual en un árbol de DFS
- Back edge
  - Viaja a un nodo gris
  - Viaja a un ancestro del nodo actual
- Forward edge
  - Viaja a un nodo negro con descubrimiento posterior al nodo actual
  - Viaja a un descendiente (no necesariamente hijo) del nodo actual
  - Solo aparece en grafos dirigidos
- Cross edge
  - Viaja a un nodo negro que finaliza antes que el descubrimiento del nodo actual
  - No viaja ni a un ancestro ni a un descendiente
  - Solo aparece en dirigidos

Propiedades piolas

- Los descendientes de un nodo x serán exactamente los nodos blancos alcanzables por un camino de nodos blancos con origen en x, en el instante en que se descubre (white-path-theorem).
- Un grafo es acíclico (dirigido o no) si y solo si un recorrido de DFS no encuentra ninguna back-edge.
- Un orden topologico puede ser tomar los vertices en orden inverso de finalizacion.
- Un grafo dirigido es singly-connected si existe a lo sumo un camino entre cada par de nodos. Dar un algoritmo O(V x E) para decidir si un grafo es singly-connected. Y O(V x V)?

Contenido

Puntos de Articulación

Componentes Conexas

Una componente conexa de un grafo (no dirigido) $G = (V, E)$, es un conjunto de vertices V' tal que para todo par de vértices $x, y \in V'$ existe un camino de x a y, y para todo par de vértices $w \in V - V', z \in V'$, se da que $(w, z) \notin E$.

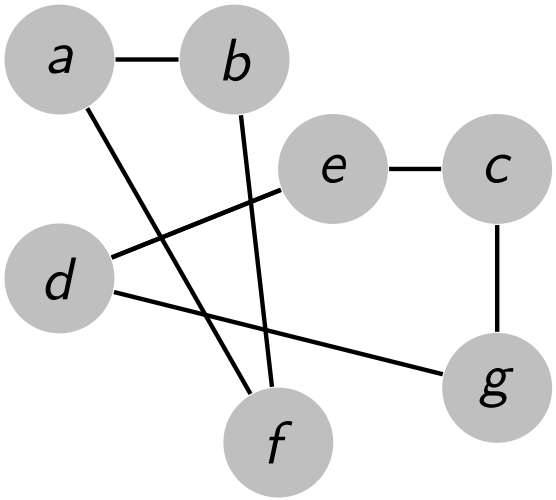
Punto de Articulación

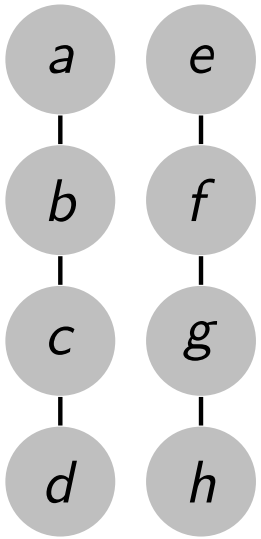
Un **punto de articulación** de un grafo (no dirigido) G es un vértice v tal que $G - v$ tiene más componentes conexas que G.

Grafo biconexo

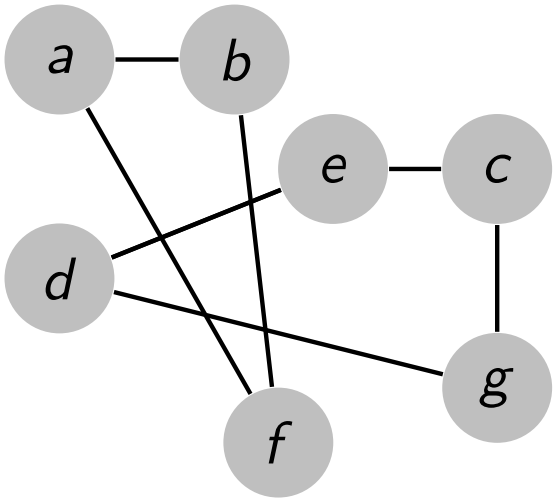
Un grafo es **biconexo** si no tiene puntos de articulación.

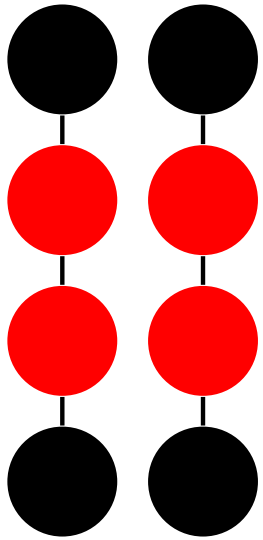
Ejemplos



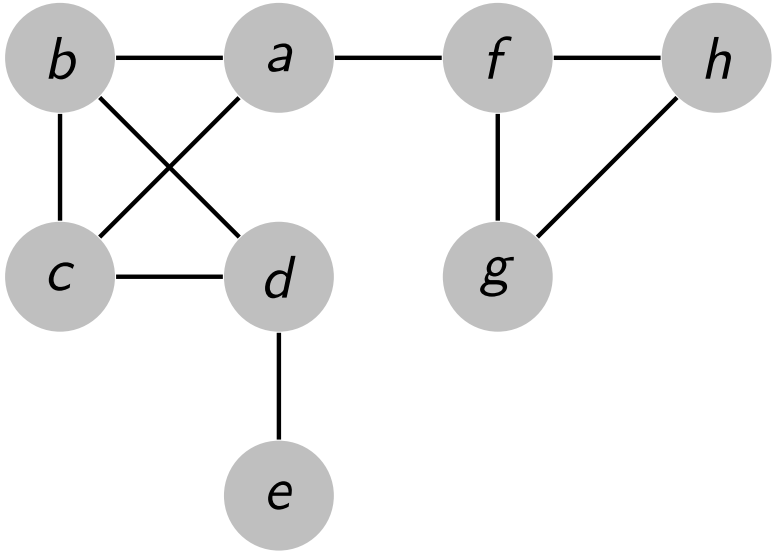


Ejemplos marcando Puntos de Articulación



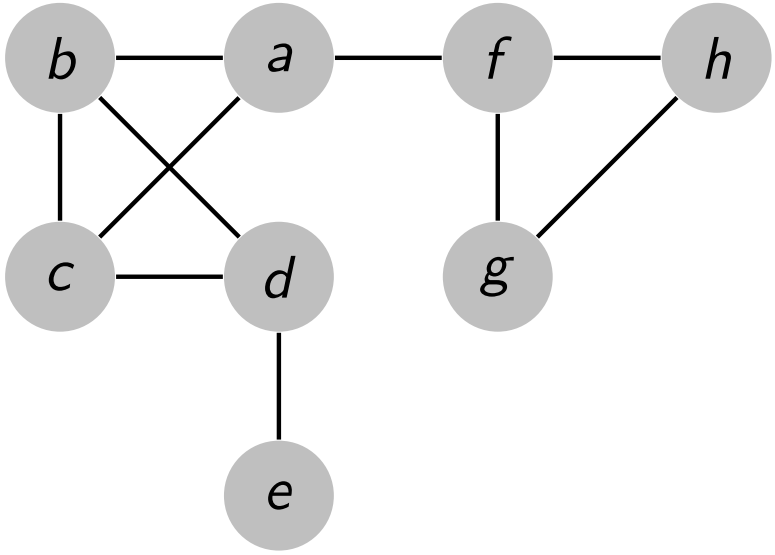


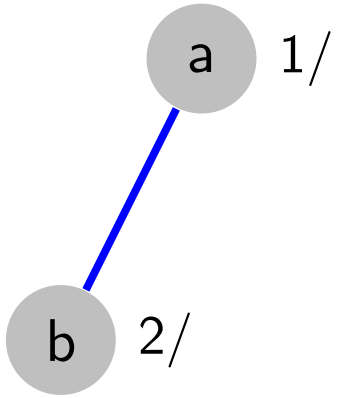
Corremos DFS en algun nodo



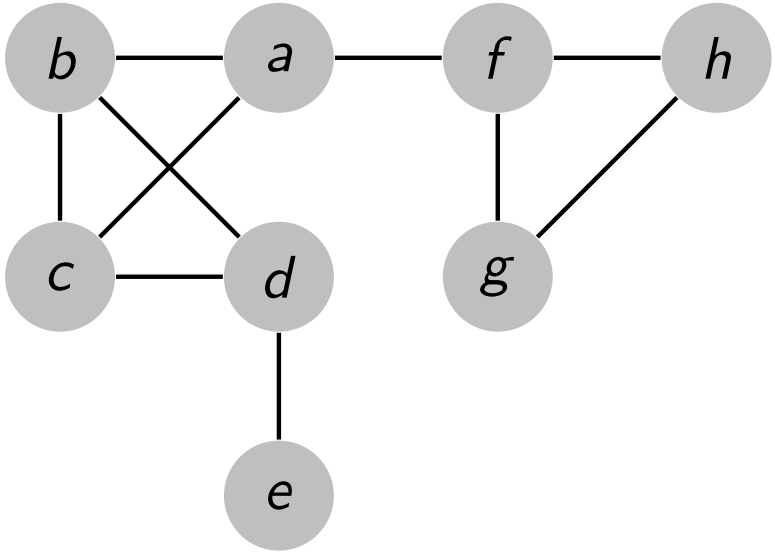
a 1/

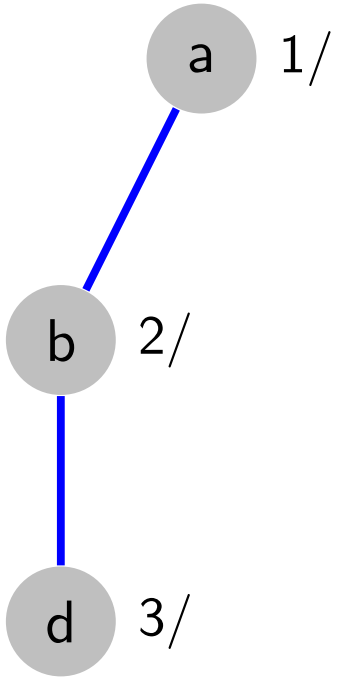
Corremos DFS en algun nodo



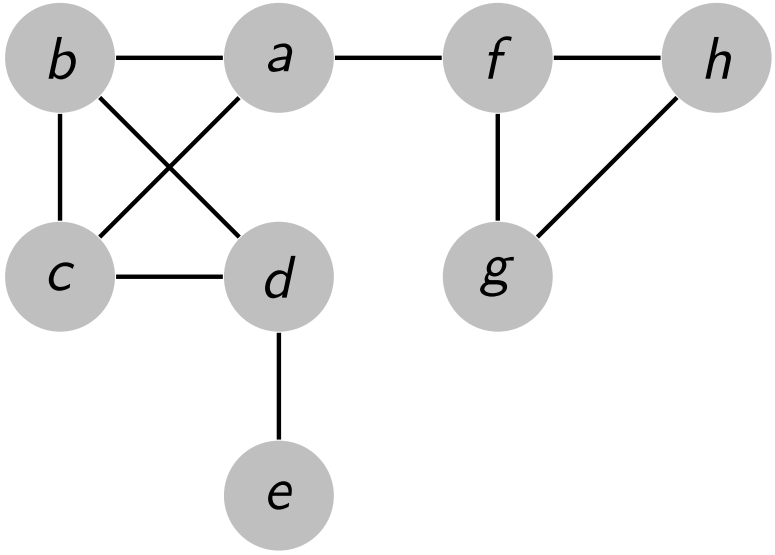


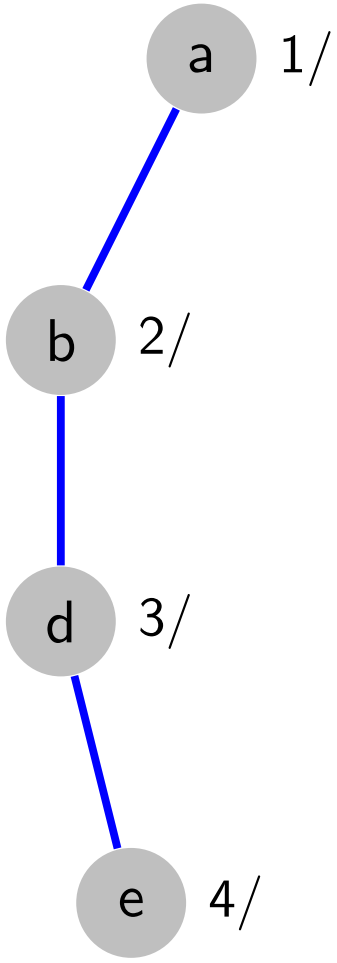
Corremos DFS en algun nodo



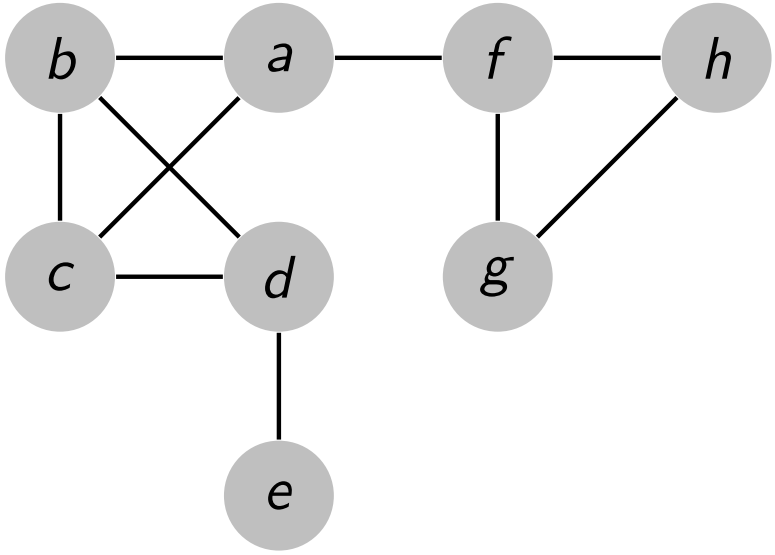


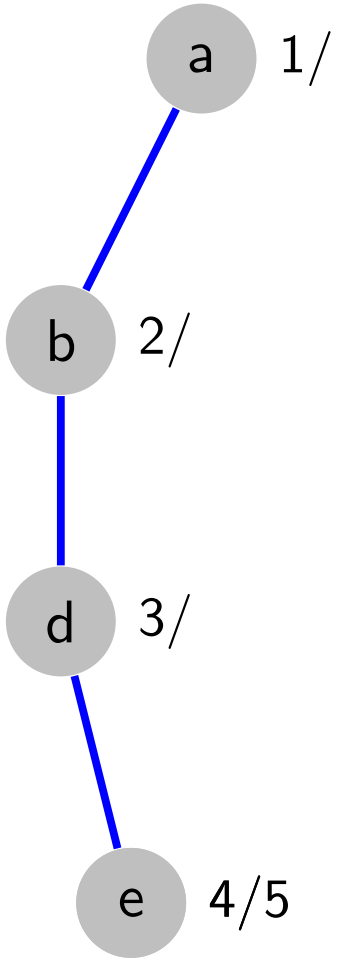
Corremos DFS en algun nodo



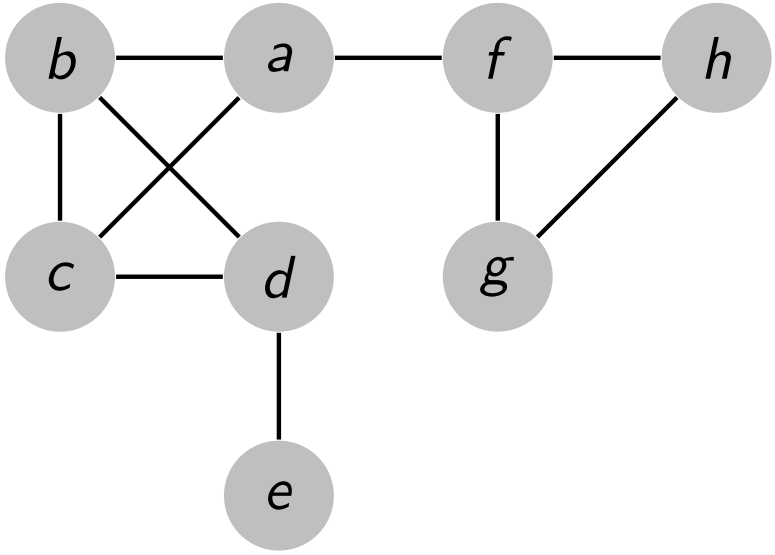


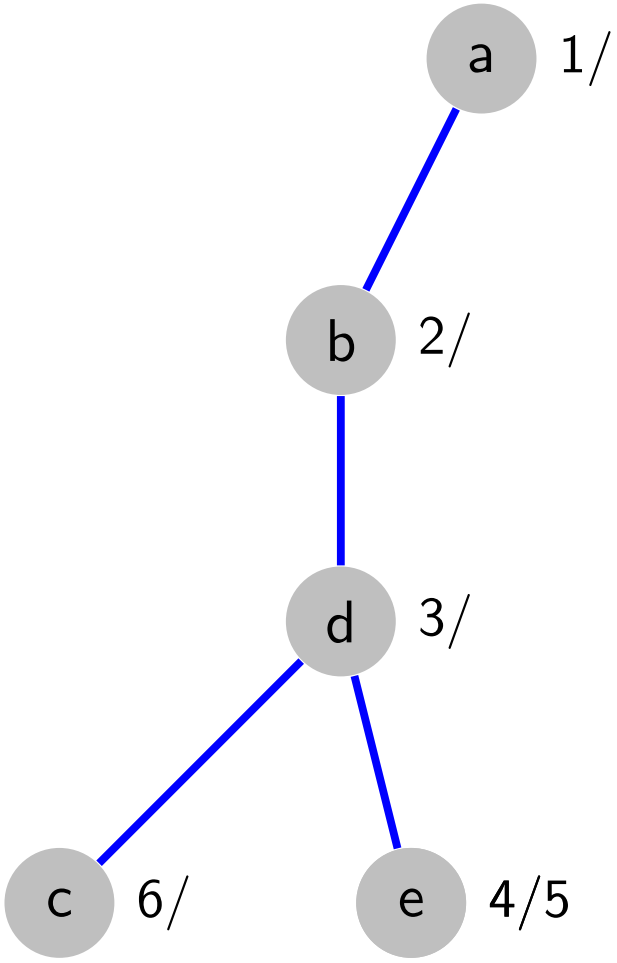
Corremos DFS en algun nodo



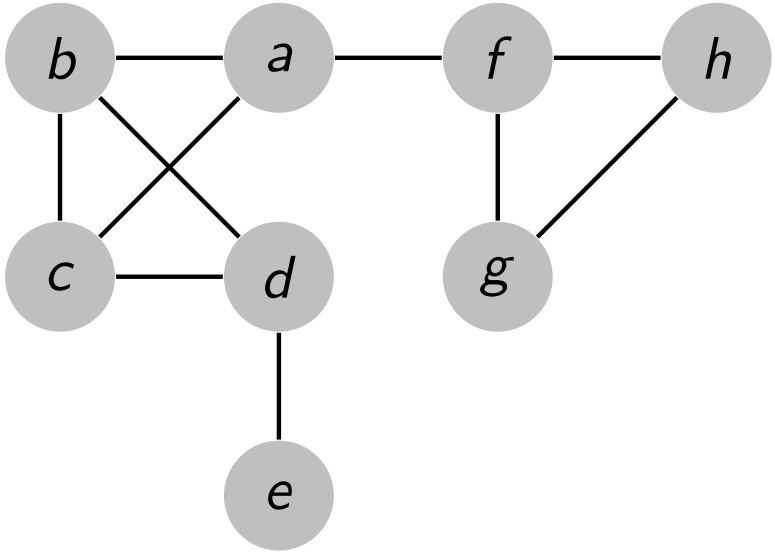


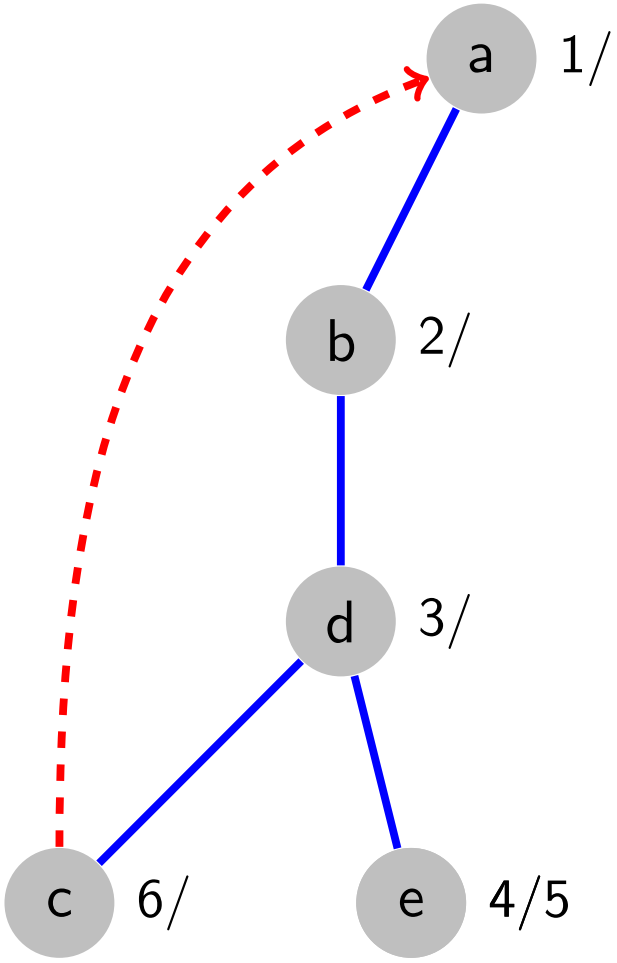
Corremos DFS en algun nodo



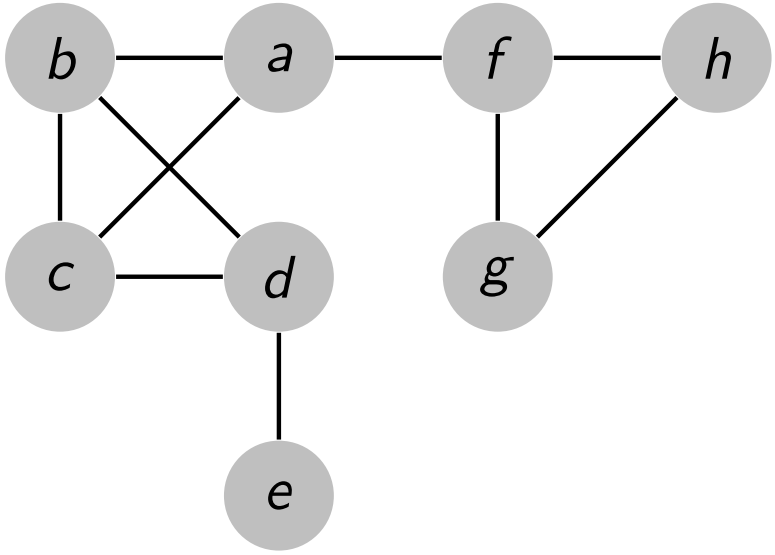


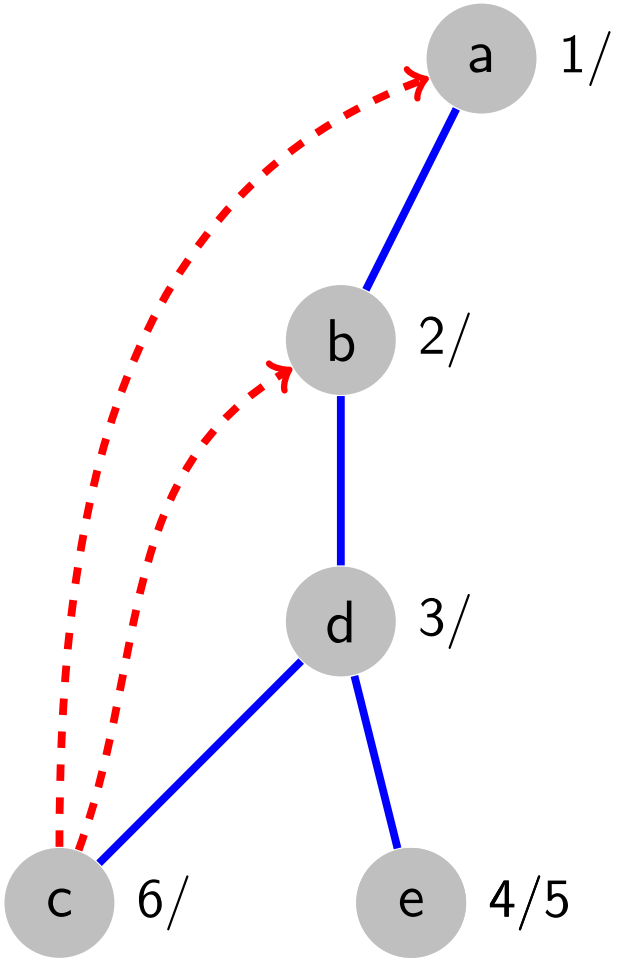
Corremos DFS en algun nodo



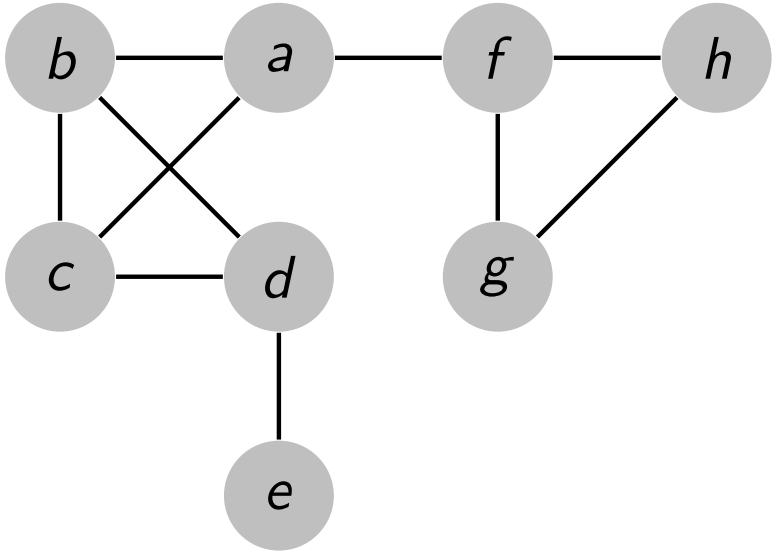


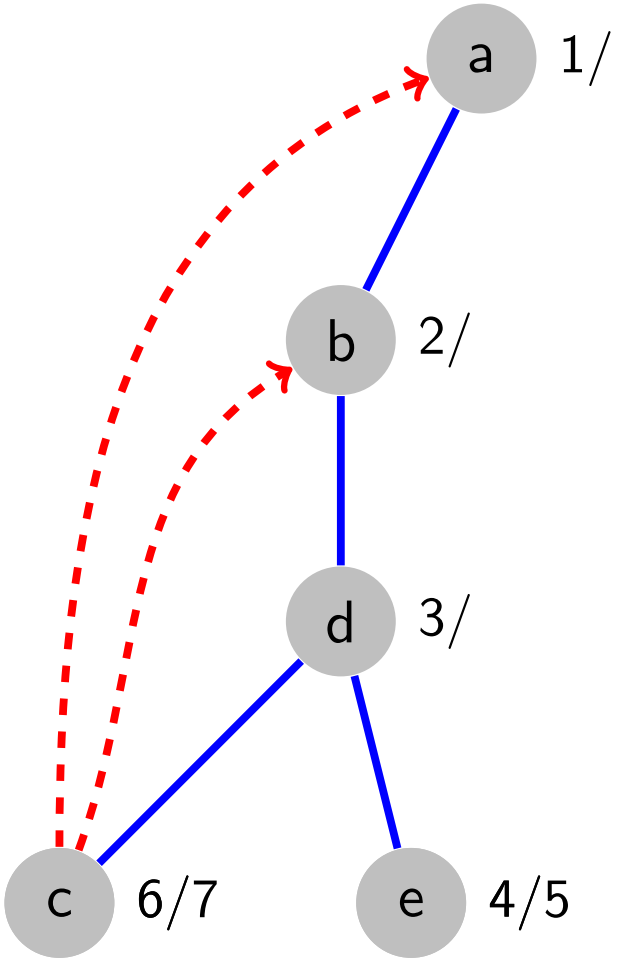
Corremos DFS en algun nodo



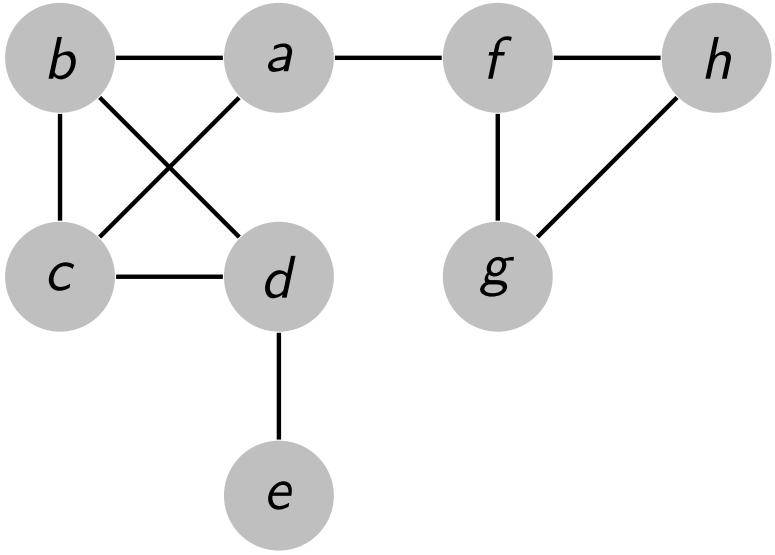


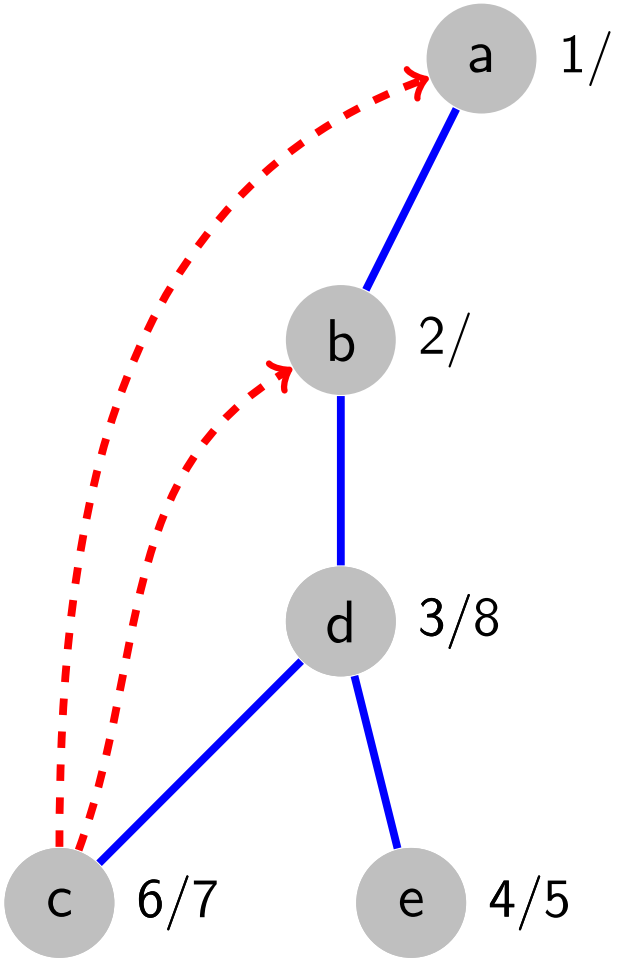
Corremos DFS en algun nodo



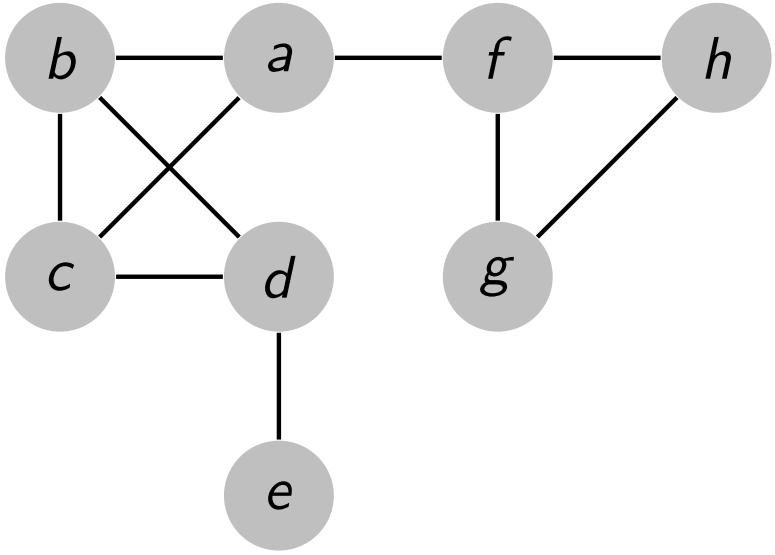


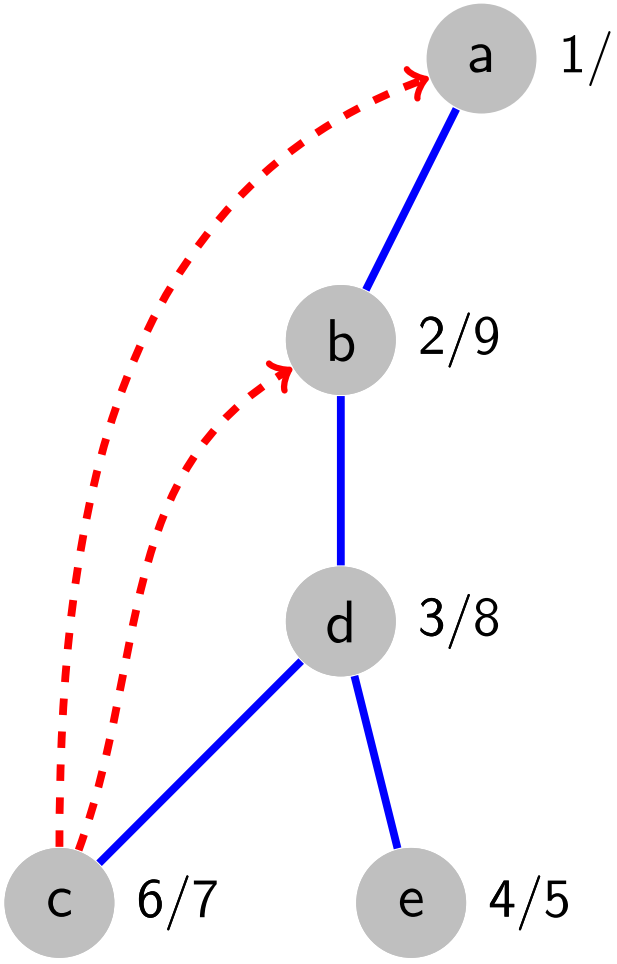
Corremos DFS en algun nodo



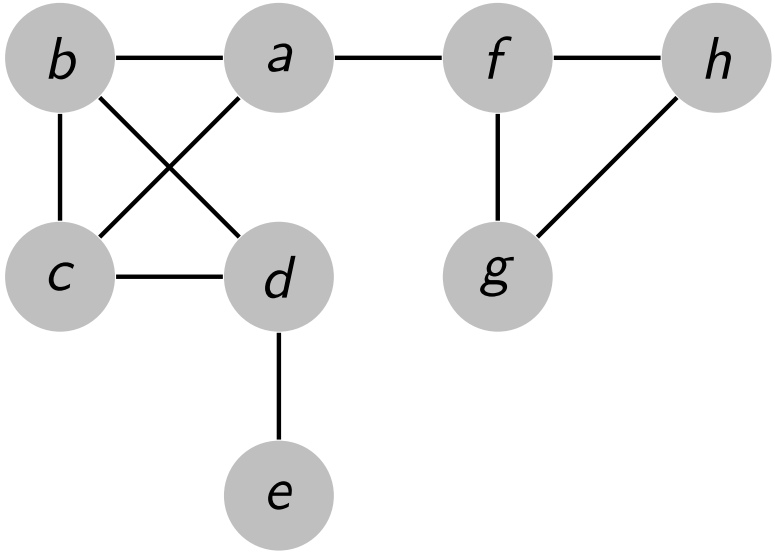


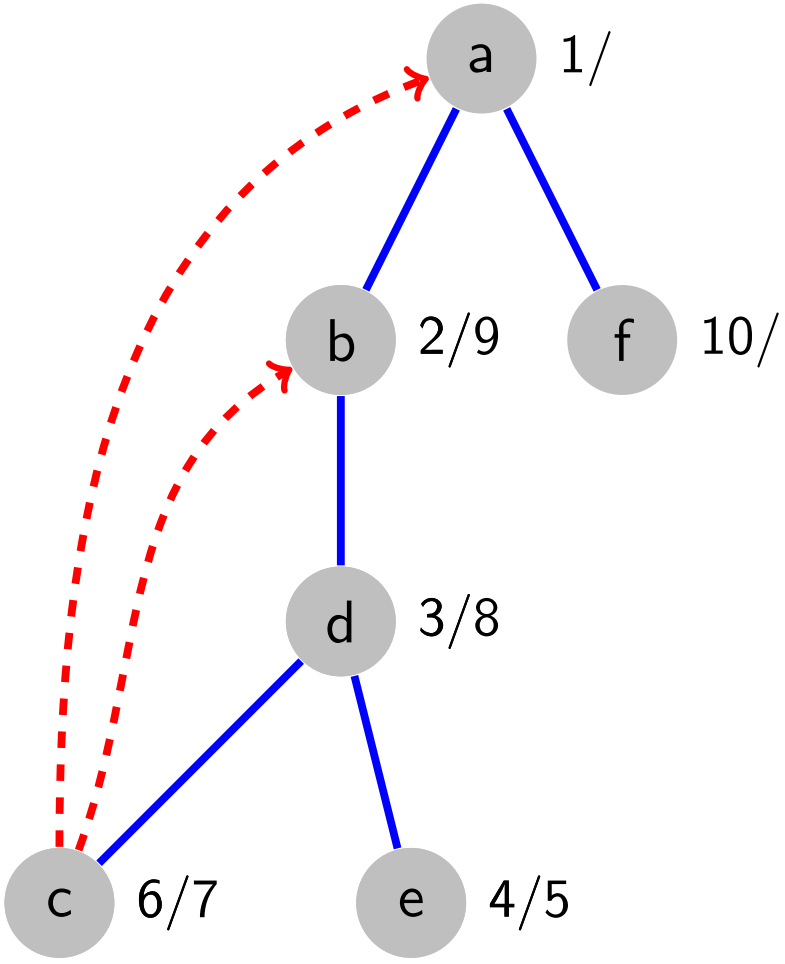
Corremos DFS en algun nodo



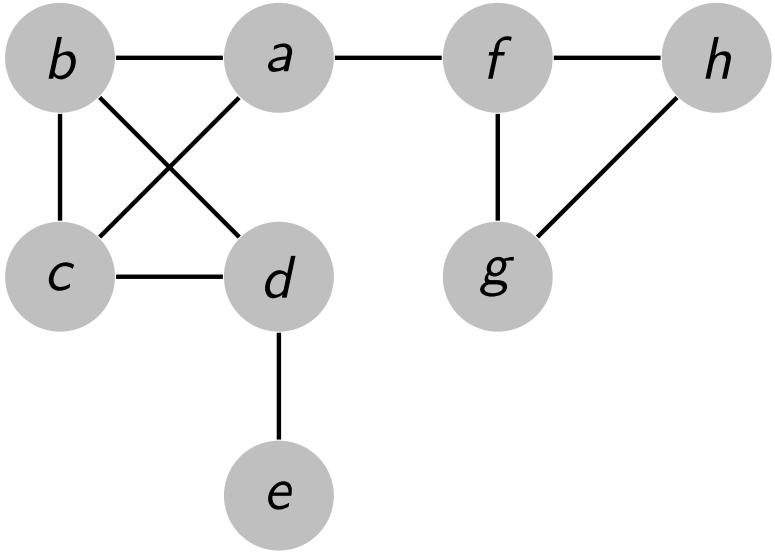


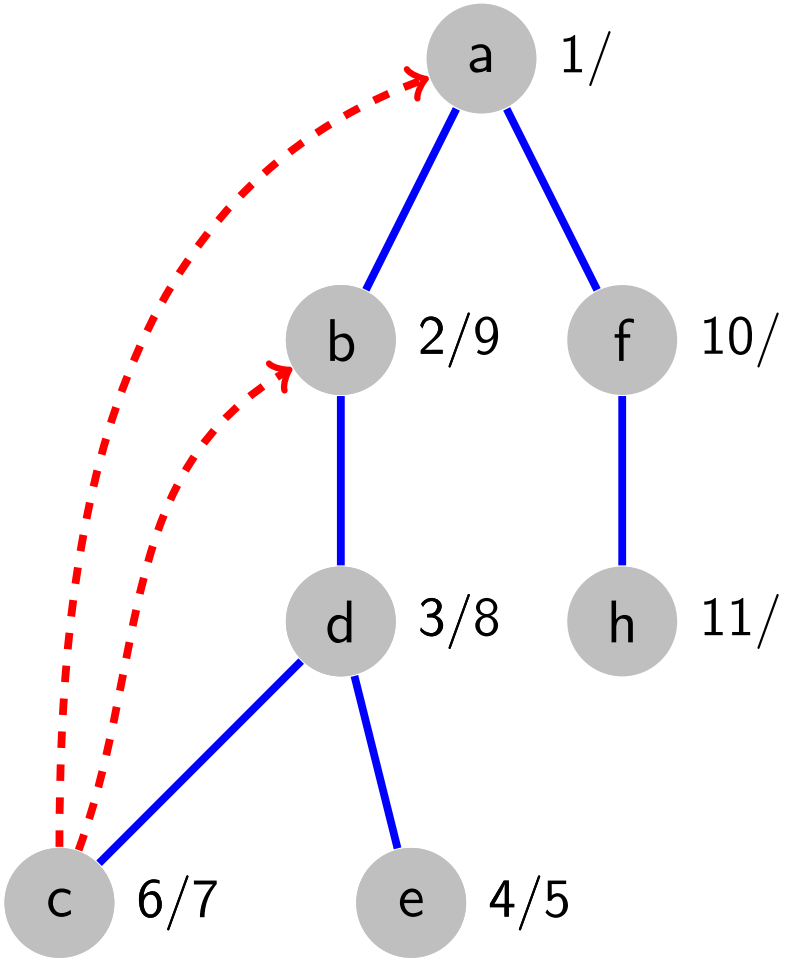
Corremos DFS en algun nodo



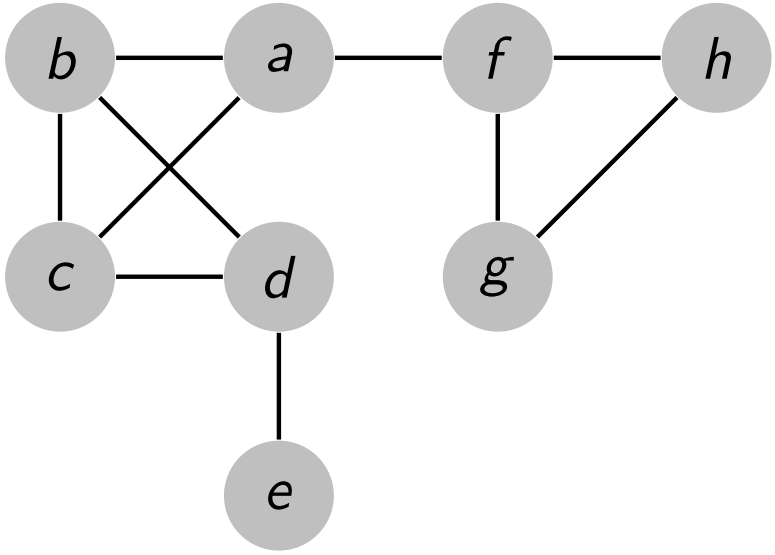


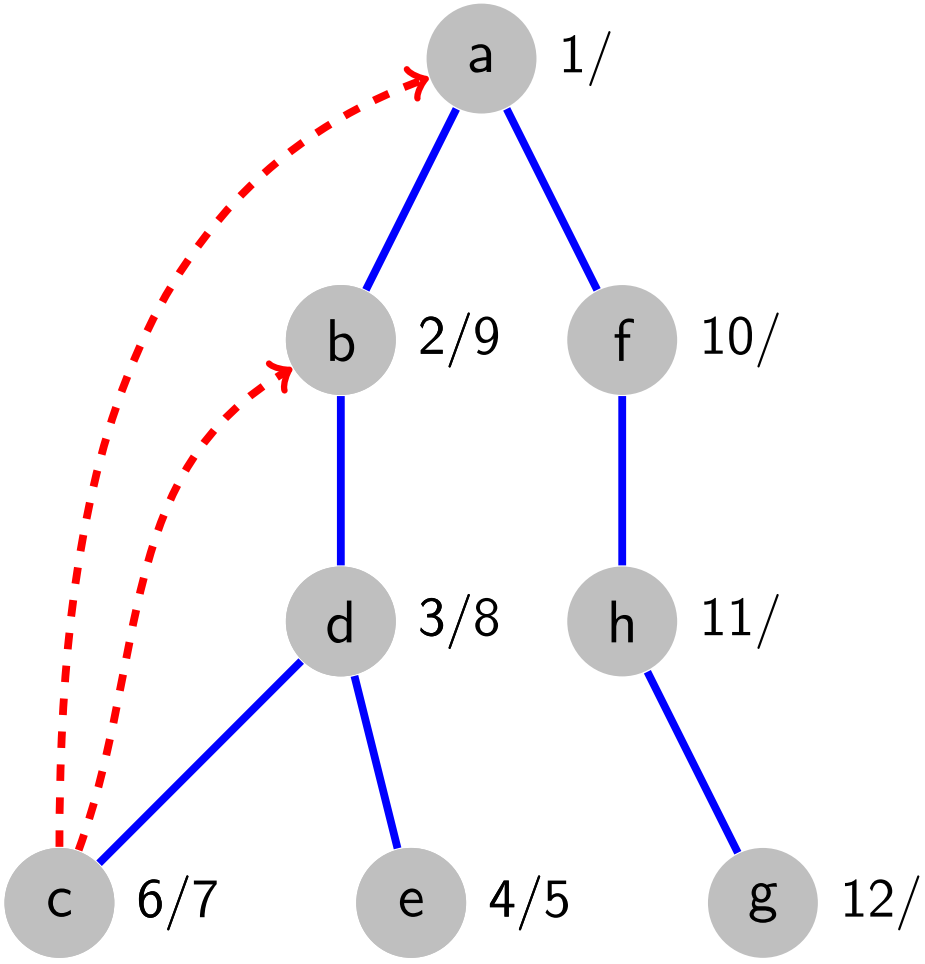
Corremos DFS en algun nodo



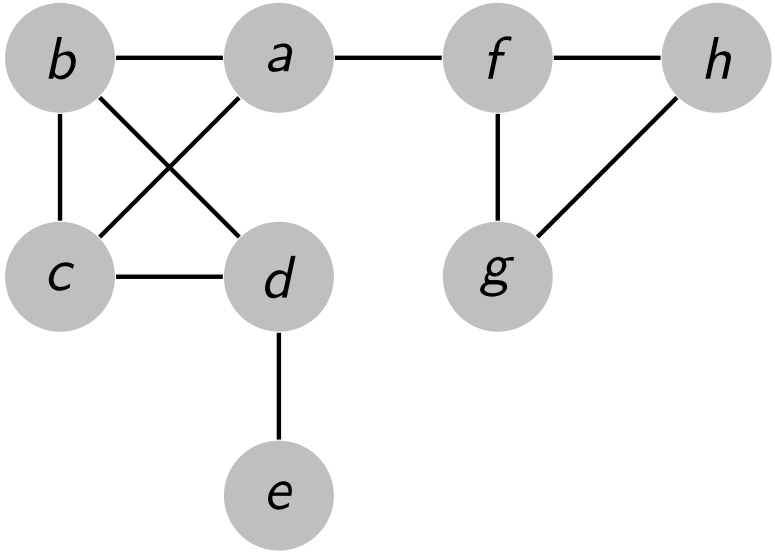


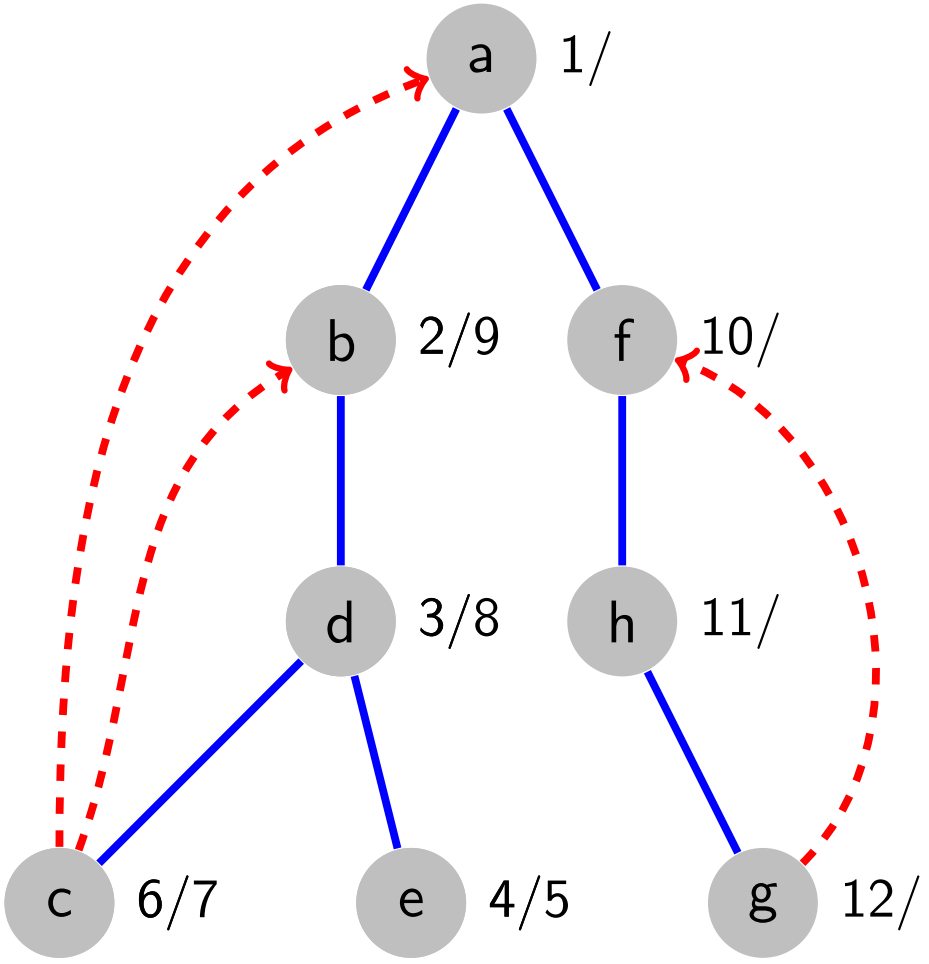
Corremos DFS en algun nodo



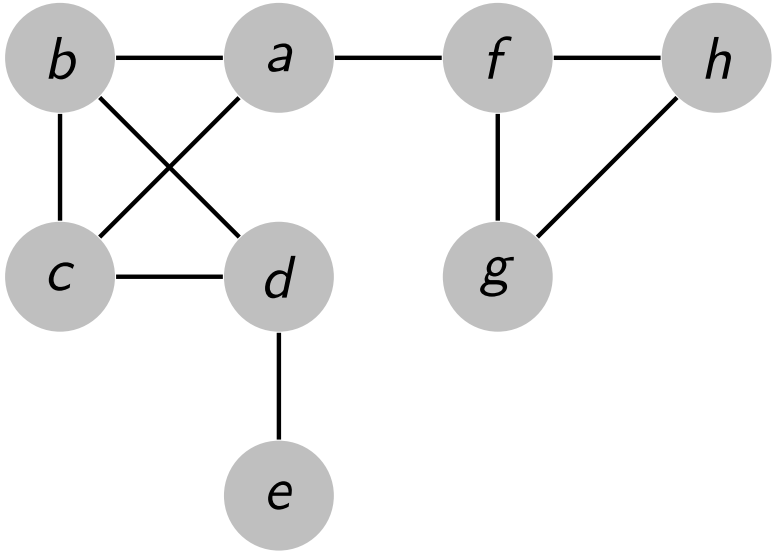


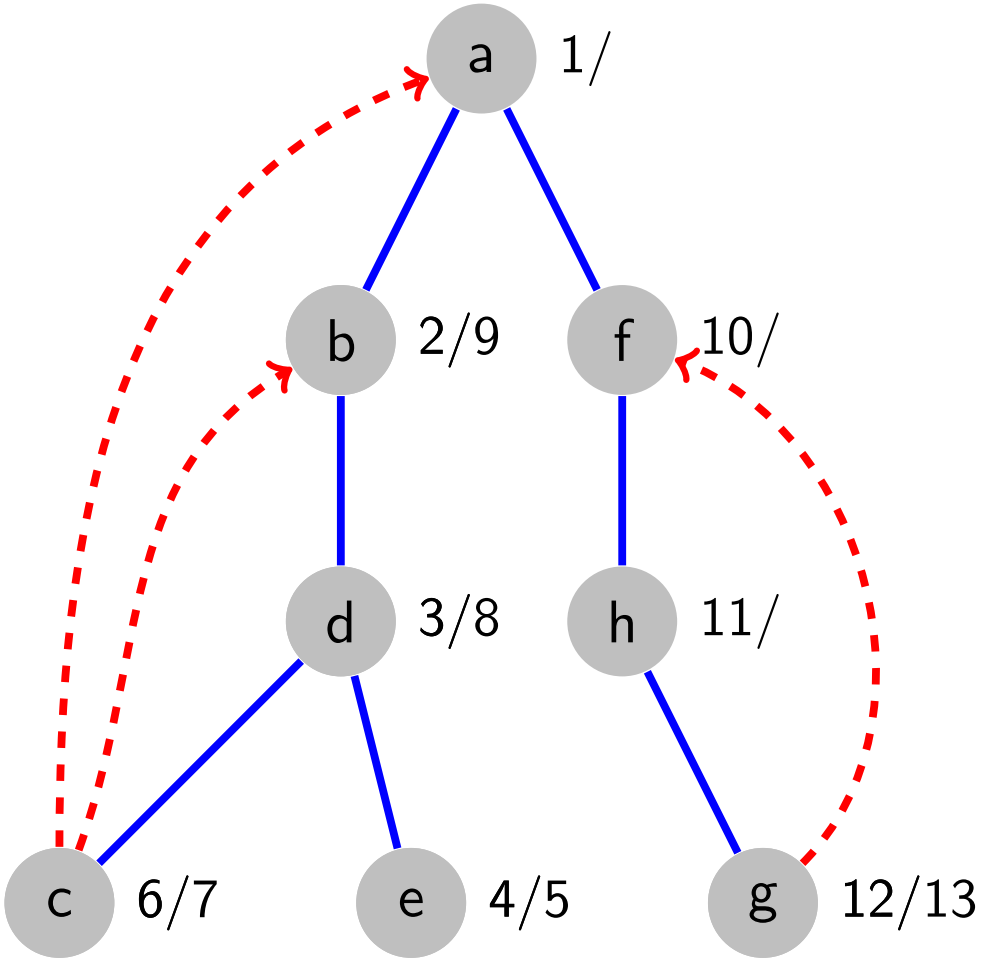
Corremos DFS en algun nodo



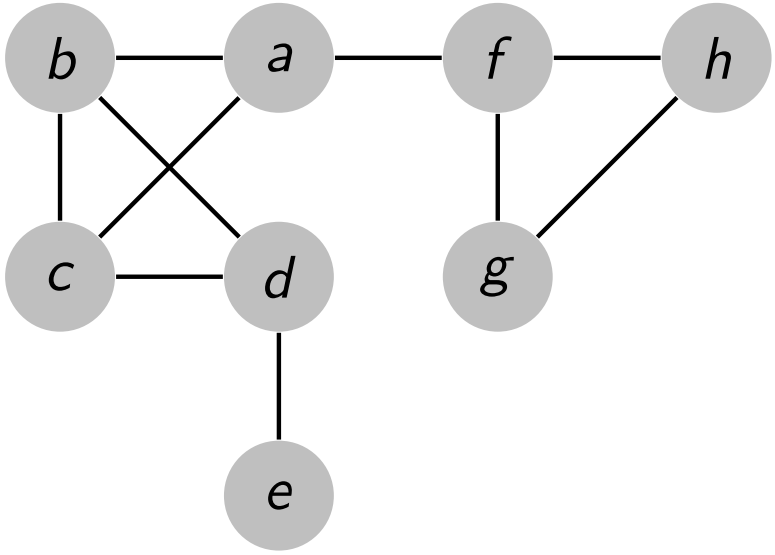


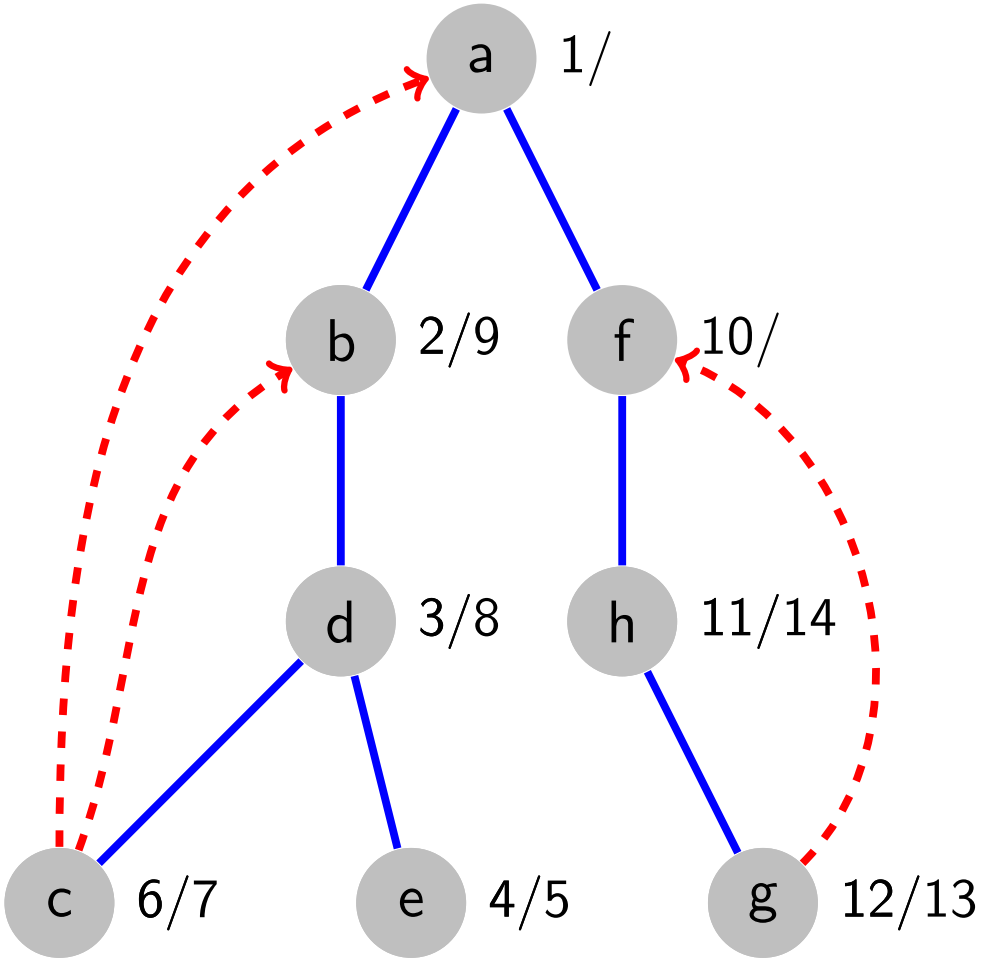
Corremos DFS en algun nodo



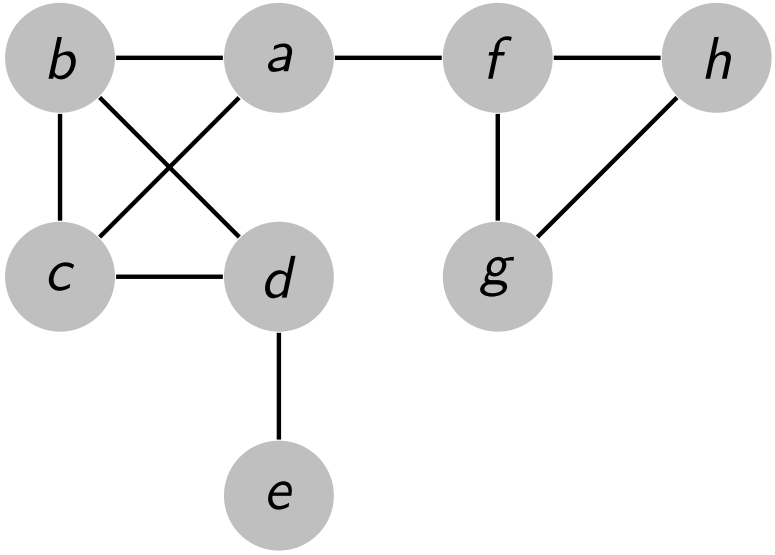


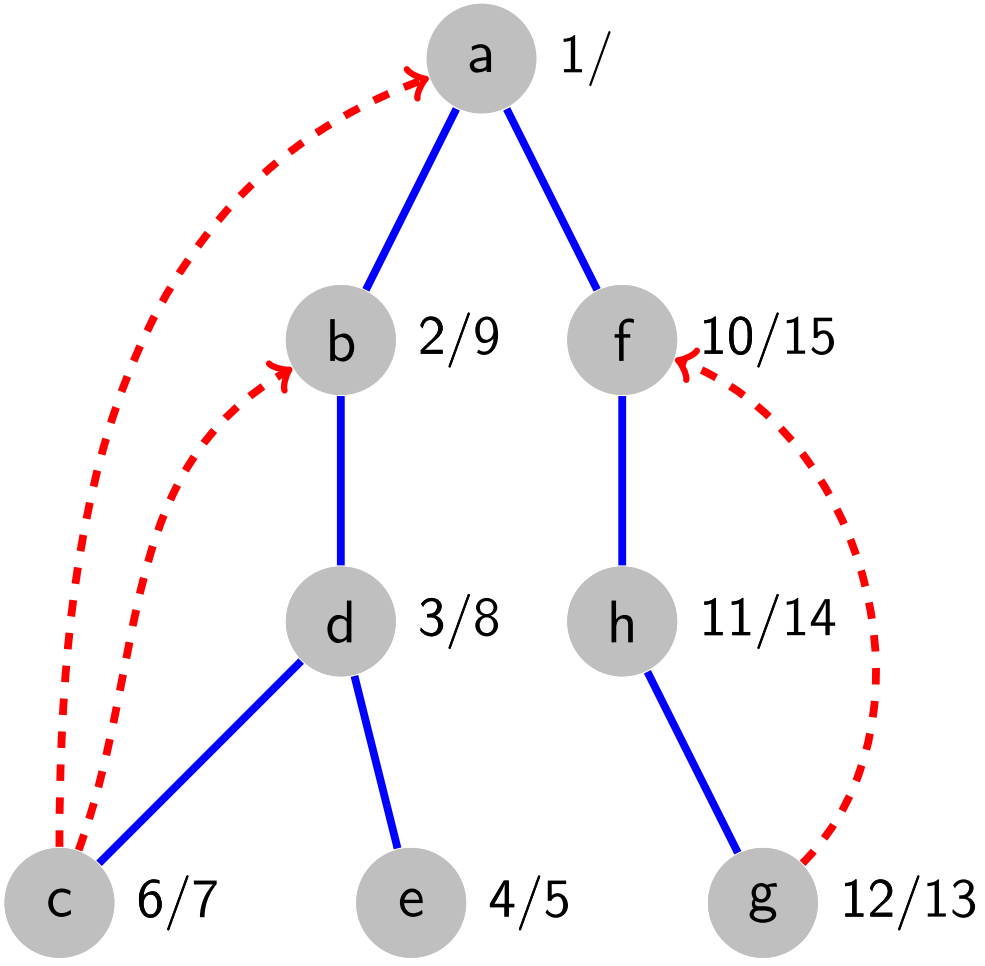
Corremos DFS en algun nodo



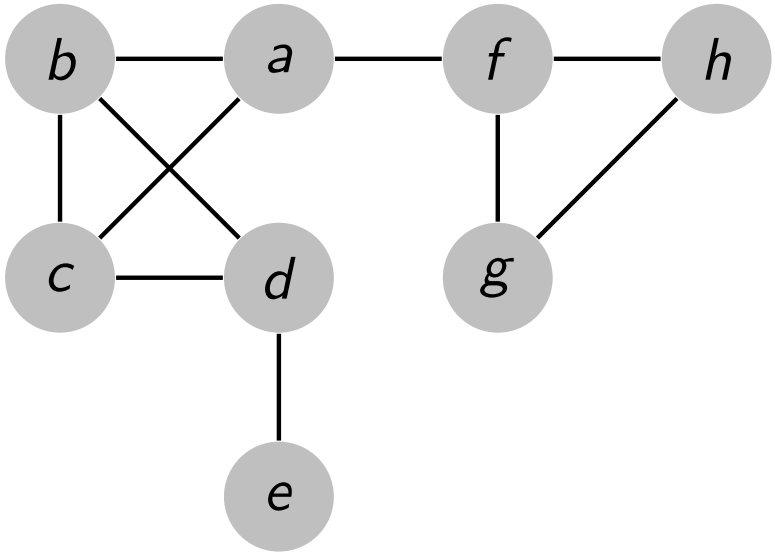


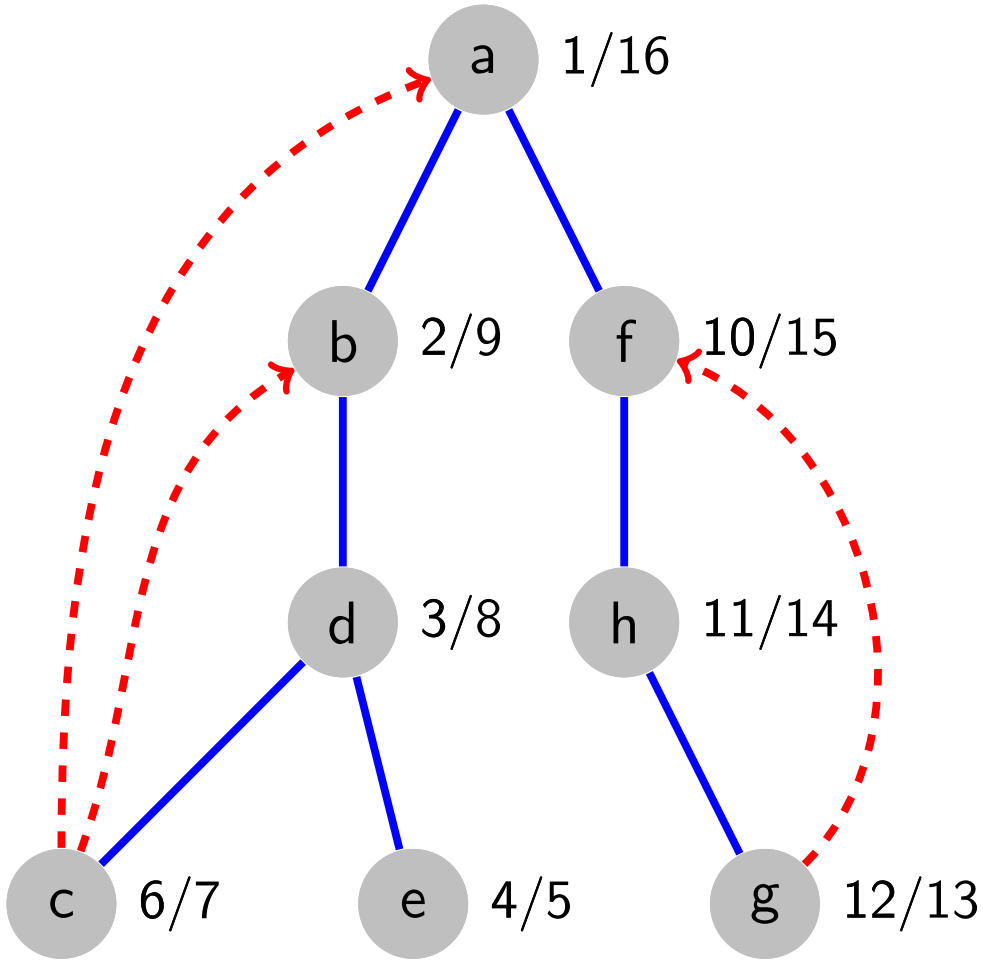
Corremos DFS en algun nodo





Corremos DFS en algun nodo



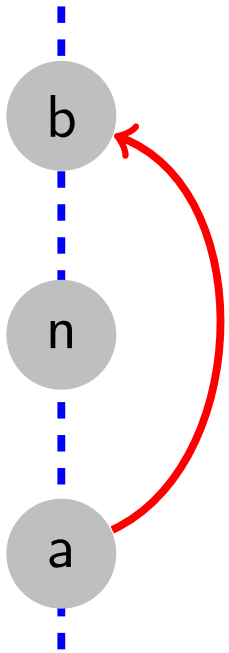


Para los puntos de articulación observamos 3 casos

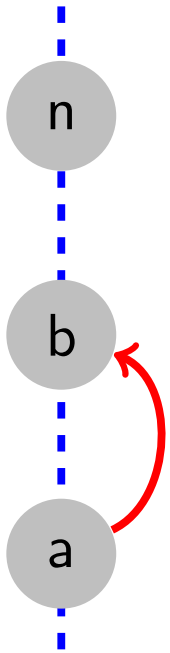
Supongamos que estoy parado en la recursion del DFS en un nodo n, y quiero saber si es un punto de articulación. Observamos 3 casos:

- Caso 1: Sus descendientes tienen un back edge que vuelve a un ancestro.
- Caso 2: Ningun back edge parte de un descendiente a un ancestro
- Caso 3: Como el caso 2, pero hay algun back edge que vuelve a n.

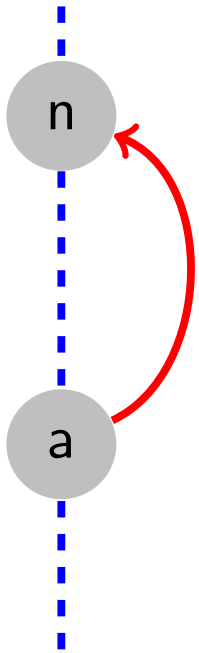
Caso 1



Caso 2



Caso 3



Código DFS para obtener Puntos de Articulación

```
1  Num desc[]  // Inicializado arreglo en 0s
2
3  Num cont = 1
4
5  Num DFS_PA (Node n)
6      desc[n] = ++cont
7      Num min = cont
8      for m in E(n):
9          if desc[m] = 0:
10             Num min_m = DFS_PA(m)
11             if min_m < min:
12                 min = min_m
13             if min_m >= desc[n]:
14                 // n es un punto de articulacion
15                 Imprimir n
16         else if desc[m] < min:
17             min = desc[m]
18     return min
```

Preguntitas

- Podria usar como 'cont' la profundidad/altura de cada nodo con respecto al Arbol generado por el DFS, y aun asi el algoritmo funcionaria?
- Funcionaria BFS?

Puentes

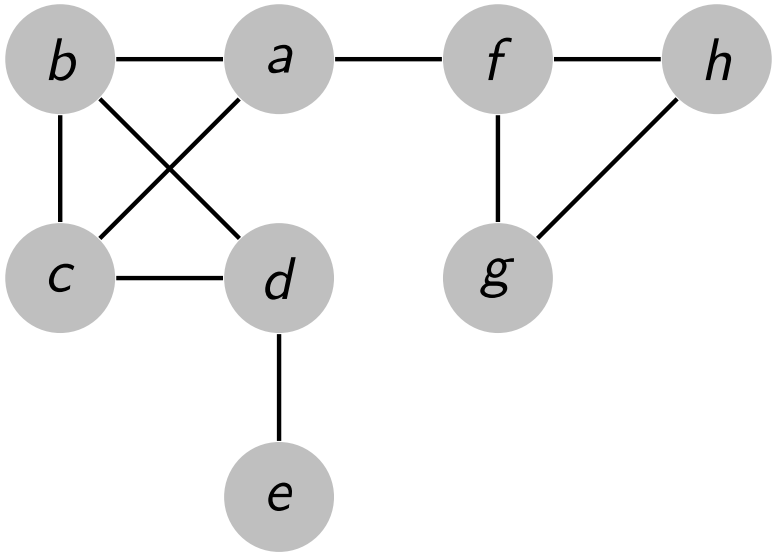
Definición de puente

Un puente de un grafo G es una arista e tal que $G - e$ tiene más componentes conexas que G.

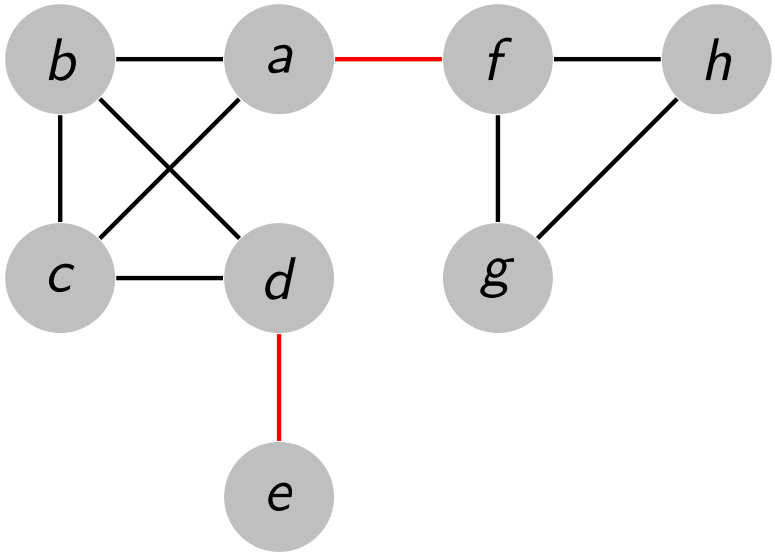
Preguntas...

- Sea G conexo, v un punto de articulación y e un puente ¿Puede ser que $G - v$ tenga más de dos componentes conexas? ¿Y $G - e$?
- ¿Existe algún grafo biconexo que tenga un puente?

Ejemplo



Ejemplo marcando puentes

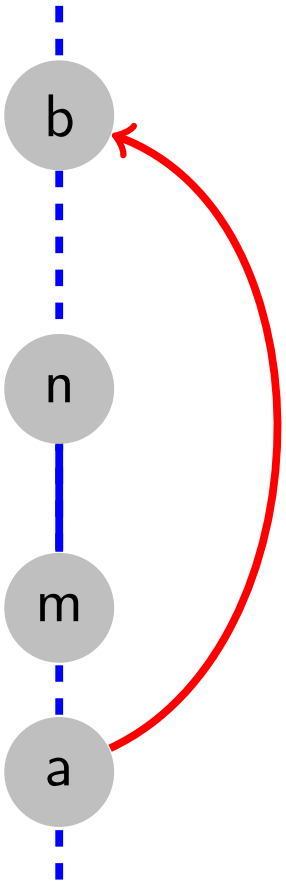


Para los puentes volvemos a observar 4 casos

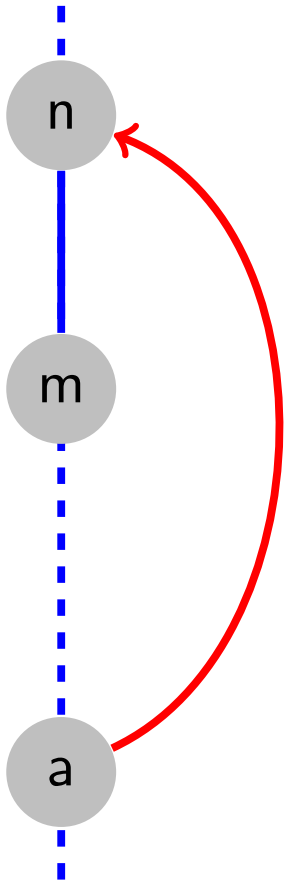
Supongamos que estoy parado en la recursion del DFS en un nodo n, con arista a m, y quiero saber si (n,m) es un puente. Observamos 4 casos:

- Caso 1: Los descendientes de m tienen un back edge que vuelve a un ancestro de n.
- Caso 2: Los descendientes de m tienen un back edge que vuelve a n.
- Caso 3: Ningun back edge parte de un descendiente de m a un ancestro de n.
- Caso 4: Como el caso 2, pero hay algun back edge que vuelve desde un descendiente de m hasta exactamente m.

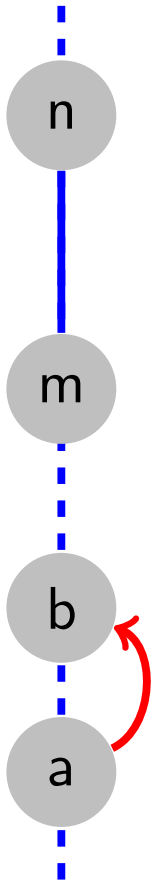
Caso 1



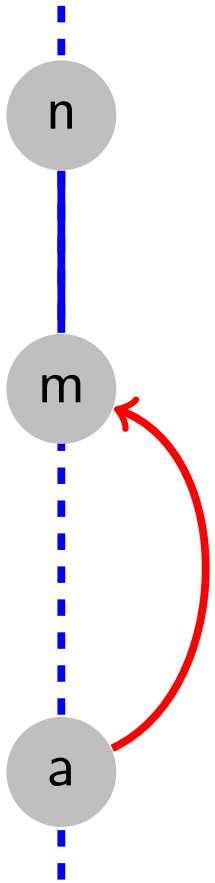
Caso 2



Caso 3



Caso 4



Código para obtener puentes

```
1  Num desc[]  // Inicializado en 0's
2  Num cont = 1
3  Num DFS_PAYP (Node n)
4      desc[n] = ++cont
5      Num min = cont
6      for m in E(n):
7          if desc[m] = 0:
8              Num min_m = DFS_PA(m)
9              if min_m < min:
10                 min = min_m
11             if min_m >= desc[n]:
12                 // n es un punto de articulacion
13                 Print(n)
14             if min_m > desc[n]:
15                 // n-m es un puente
16                 Print(n, m)
17         else if desc[m] < min:
18             min = desc[m]
19     return min
```

Diametro de un Grafo no dirigido

Sea G un arbol, sin pesos en sus aristas.
Sea dist(a,b) la distancia minima entre nodos a y b.
Cual es el valor maximo de d(a, b) para cualquier par de nodos?

Solucion

Tomar un nodo u.
Correr BFS(u) y obtener el nodo mas lejano, denotemoslo v.
Correr BFS(v) y obtener la distancia D al nodo mas lejano de v.
D es el diametro del grafo.
Ejercicio: probar la correctitud del algoritmo.

Contenido

Conceptos

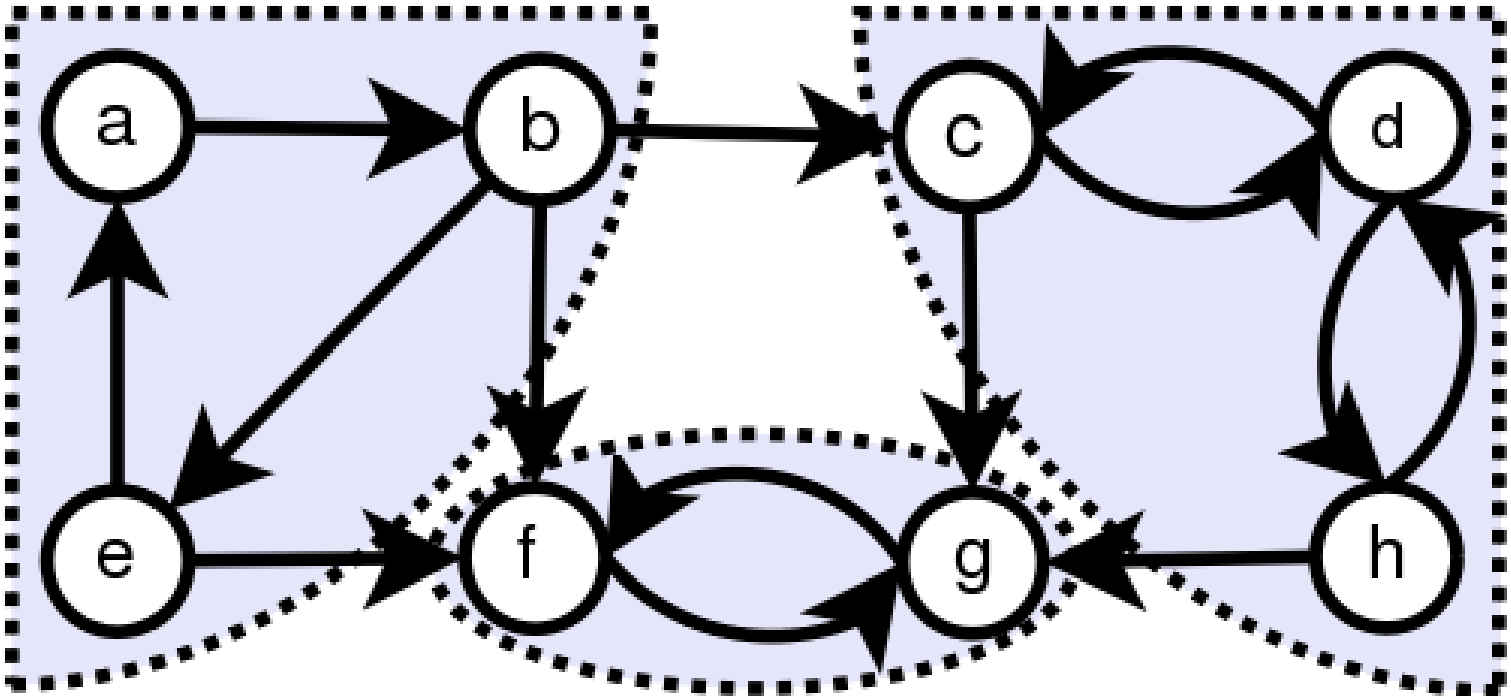
Grafo fuertemente conexo

Un grafo dirigido G es llamado fuertemente conexo si para cada par de vértices u y v existe un camino de u hacia v y un camino de v hacia u.

Componentes fuertemente conexas (Strongly Connected Components, SCC)

Los componentes fuertemente conexos de un grafo dirigido son sus subgrafos maximales fuertemente conexos.

Dibujo



Algoritmo de Kosaraju

```
1) visited = [false para todo nodo], Componente = [-1 para
    todo nodo], L = [].
2) Para todo nodo correr dfs(u)
3) for v in L: // Iterando de izquierda a derecha
    Correr Asignar(u, u)

dfs(u):
    if !visited[u]
        visited[u] = true
        for v in edges[u]:
            dfs(v)
        L = [v] + L

Asignar(u, root):
    if Componente[u] = -1:
        Componente[u] = root
        for v in redges[u]:
            Asignar(v, root)
```

Algoritmo de Tarjan

```
1  cont = 1; desc= Num[]; onStack = bool[]
2  S = new Stack()
3  Num strongconnect(Node v)
4      desc[v] = cont, Num min = cont
5      cont = cont + 1
6      S.push(v), onStack[v] = true
7      for w in E[v]:
8        if desc[w] = 0:
9          Num min_m = strongconnect(w)
10         if min > min_m: min = min_m
11       else if onStack[w]:
12         if min > desc[w]: min = desc[w]
13     if min = desc[v]:
14       do
15         w = S.pop(), onStack[w] = false, nuevaSCC.add(w)
16       while w != v
17       Print(nuevaSCC)
18 for v in V:
19     if desc[v] = 0:
20         strongconnect(v)
```

Contenido

Euler Path

Definición

Dado un grafo, un camino euleriano es un camino que pasa por cada arista exactamente una vez. Un ciclo euleriano es un camino euleriano que termina en su nodo de origen.

Solution

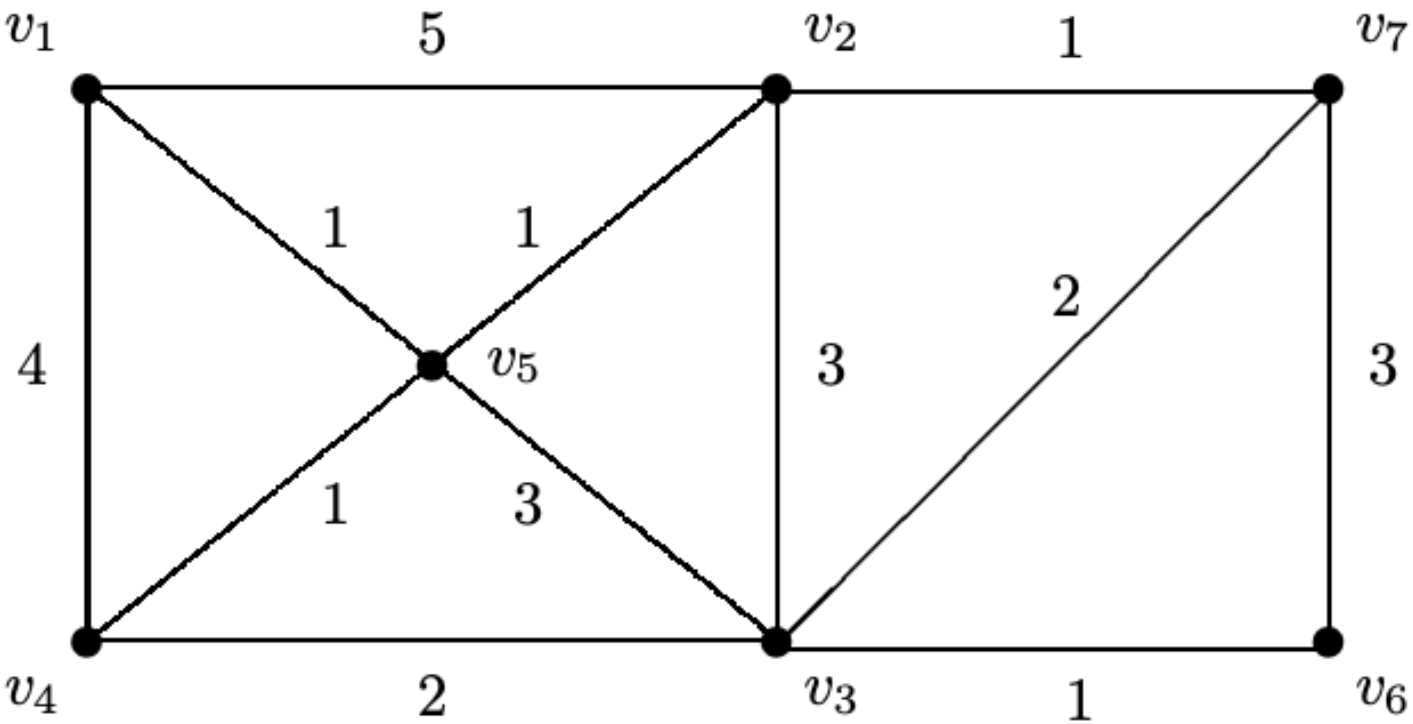
```
1  Elegir nodo de inicio v:
2  * si todos los nodos tienen grado par elegir cualquiera
3  * si hay 2 nodos con grado impar, elegir cualquiera de ellos
4  * si hay mas de dos nodos con grado impar, no hay solucion
5
6  res = []
7  S = pila vacia
8  while !S.empty && !E[v].empty
9      if E[v].empty:
10         res.append(v)
11         v = S.top();
12         S.pop()
13     else:
14         S.push(v)
15         u = E[v][0]
16         E[v].remove(0)
17         v = u
18 res.append(v)
```

Problema del Cartero Chino

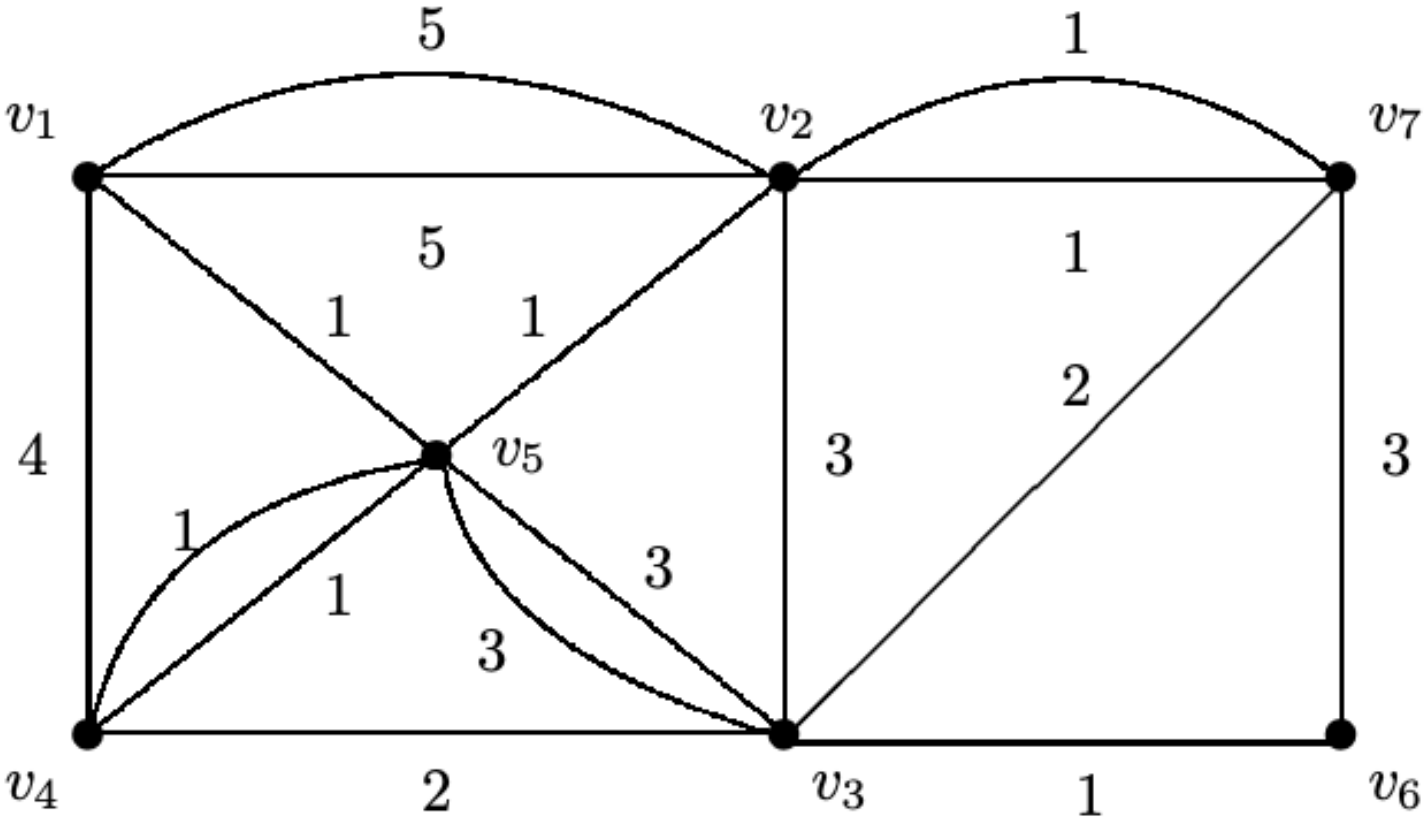
Definición

Dado un grafo no dirigido con pesos, encontrar el camino que recorre cada arista al menos una vez y tiene costo mínimo.

Dibujo



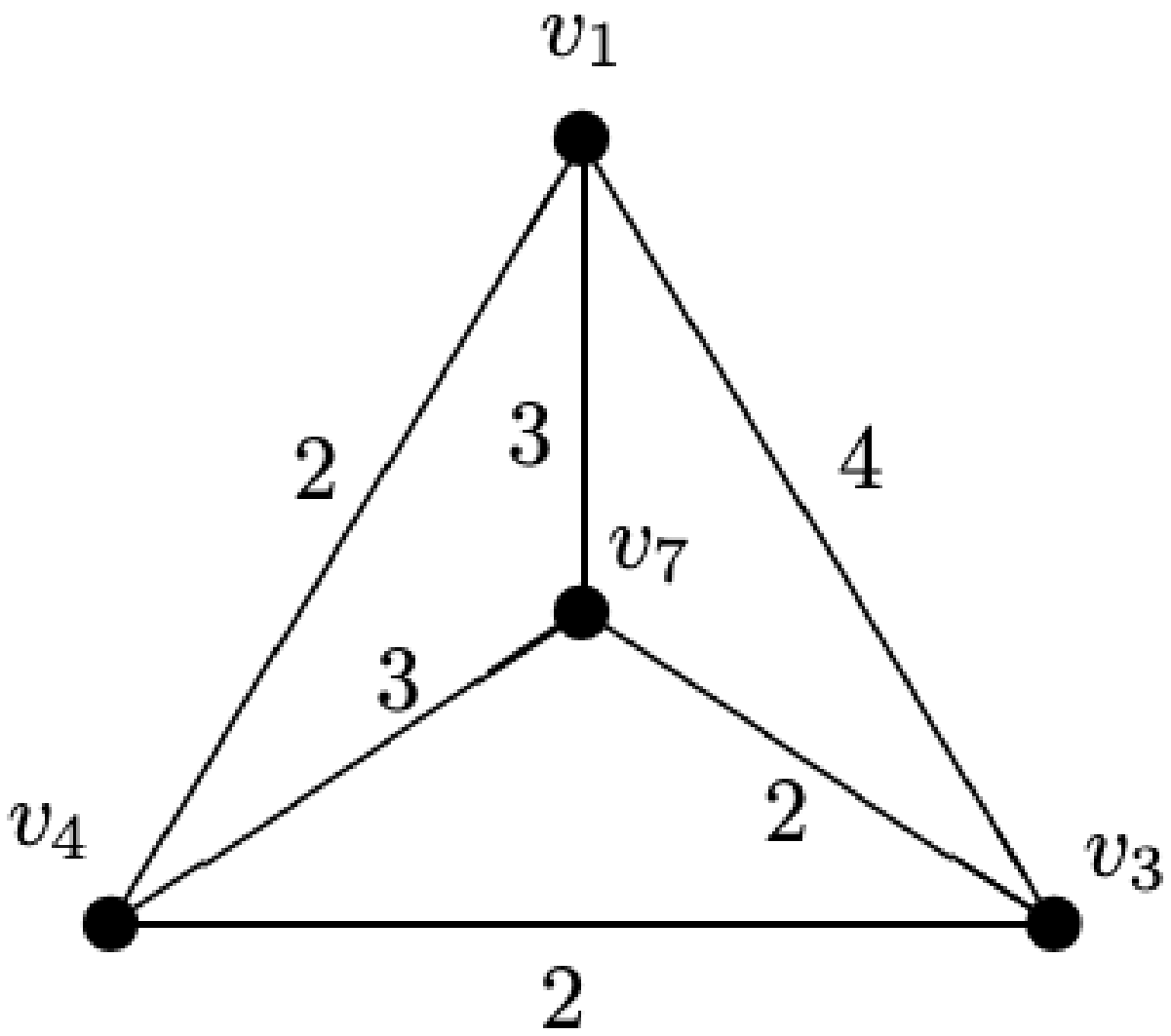
Dibujo



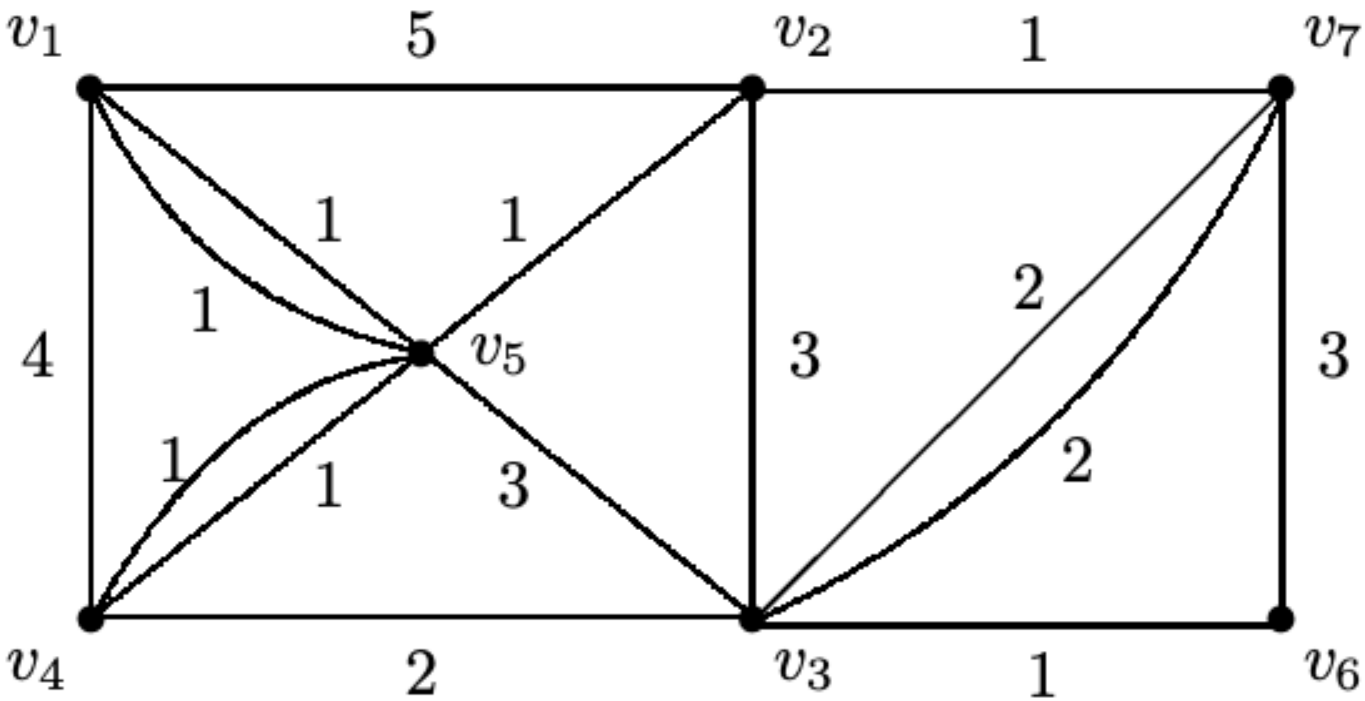
Solución Propuesta por Edmonds 1965

- Sea S = v_1, v_2, ... v_{2m} el conjunto de vertices con aridad impar. Para cada par de vértices $v_i, v_j \in S$ construir el camino mas corto $P_{i,j}$.
- Construir un grafo completo con pesos K_{2m} donde $V(K_{2m}) = S$ y $weight(v_i v_j) = length(P_{i,j})$ para todos $v_i, v_j \in S$. Encontrar el matching M de costo mínimo.
- Para cada $(v_i v_j) \in M$ agregar una duplica de cada arista de $E(P_{i,j})$, en el grafo original G. Llamemos G' a este nuevo grafo.
- Obtener el ciclo Euleriano de G'.

Dibujo



Dibujo



Fin


FIN DE LA
PRESENTACION
MUCHAS GRACIAS
POR SU ATENCION
memegenerator.es