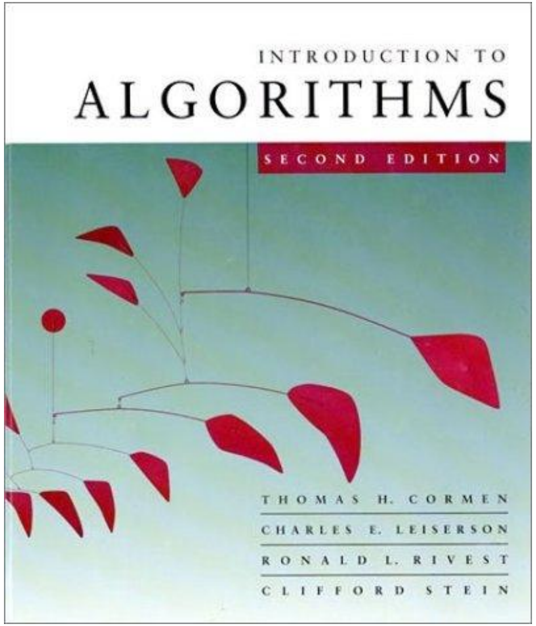
Grafos II

Training Camp Argentina 2015
Nicolás Alvarez

DFS



"El DFS nos dá información valiosa sobre la estructura de un grafo".
El Cormen

Veremos cómo utilizar la información que provee un recorrido DFS para problemas sobre orden topológico y componentes biconexas

DFS: Coloreo de nodos

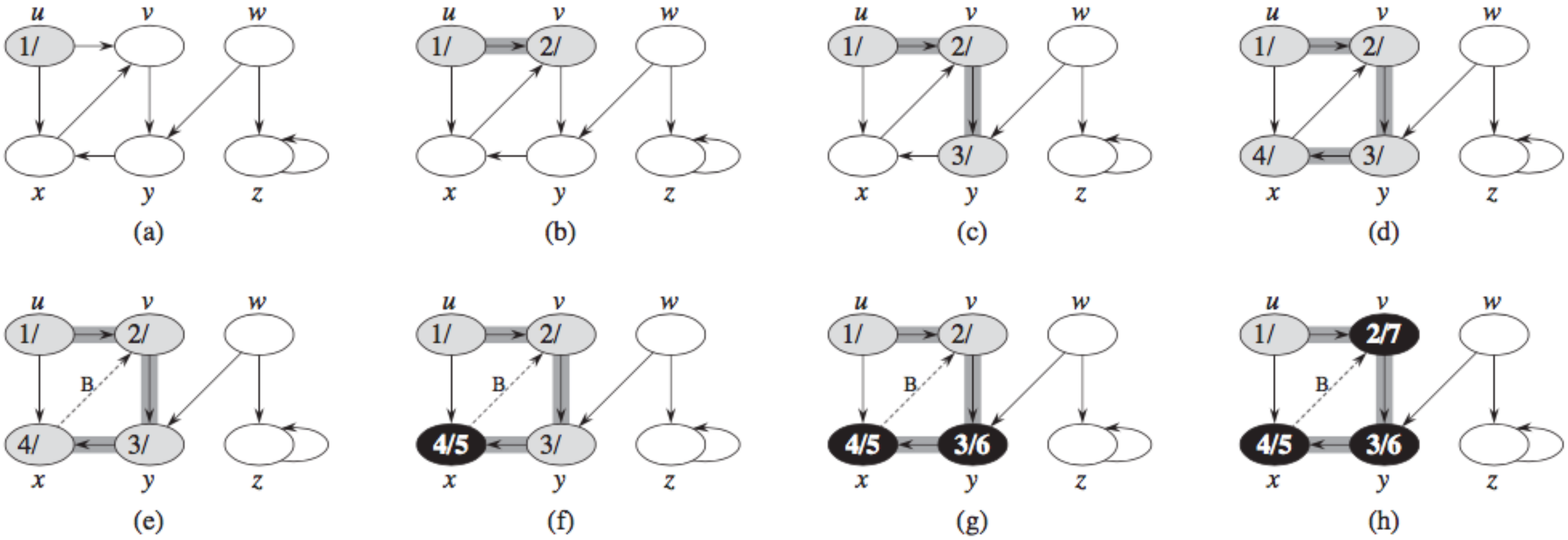
- Inicialmente todos los nodos están **blancos**
- Al momento de descubrir un nodo se lo pinta de **gris**
- Cuando se finaliza el procesamiento de un nodo se lo pinta de **negro**

Durante el recorrido, a cada nodo **v** se le asignan 2 *timestamps*.

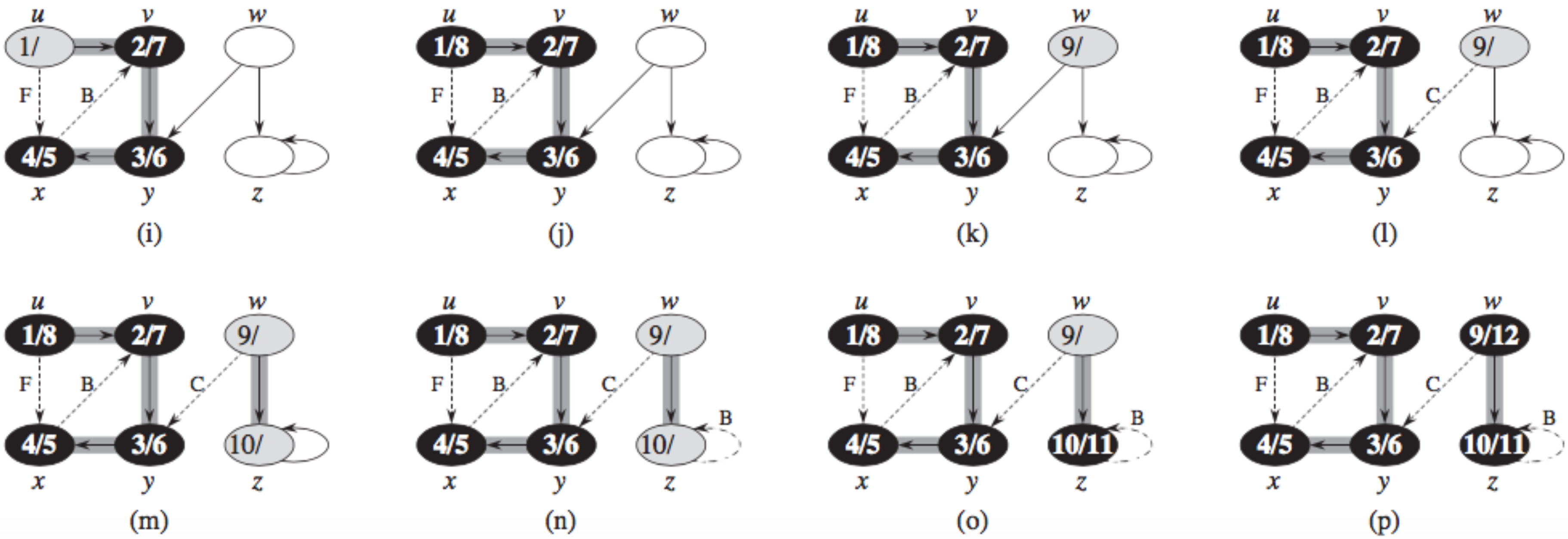
d[v] = momento de descubrimiento (pintado de gris)

f[v] = momento de finalización (pintado de negro)

DFS: Traza



DFS: Traza



DFS: Pseudocódigo

```
t = 0
d[] = {-1,-1,...,-1}
f[] = {-1,-1,...,-1}

dfs(u):
    d[u] = t++
    for v in adj(u)
        if d[v] == -1 then dfs(v)
    f[u] = t++

for u in V(G)
    if d[u] == -1 then dfs(u)
```

Estructura de paréntesis

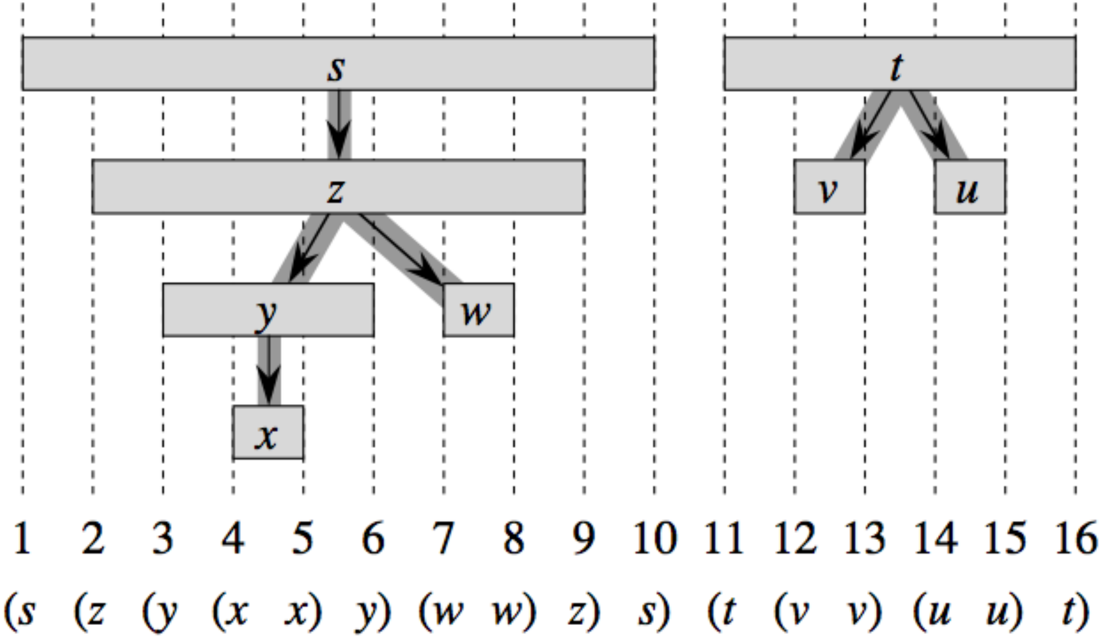
Una propiedad importante es que los tiempos d[u] y f[u] forman una expresión bien parentizada.

Esto implica que para cualquier par de nodos **u** y **v** pueden ocurrir 3 casos:

- (d[u], f[u]) está contenido en (d[v], f[v]) $\Rightarrow$ u es descendiente de v
- (d[v], f[v]) está contenido en (d[u], f[u]) $\Rightarrow$ v es descendiente de u
- (d[u], f[u]) y (d[v], f[v]) son disjuntos $\Rightarrow$ ninguno es descendiente del otro

Estructura de paréntesis

(b)



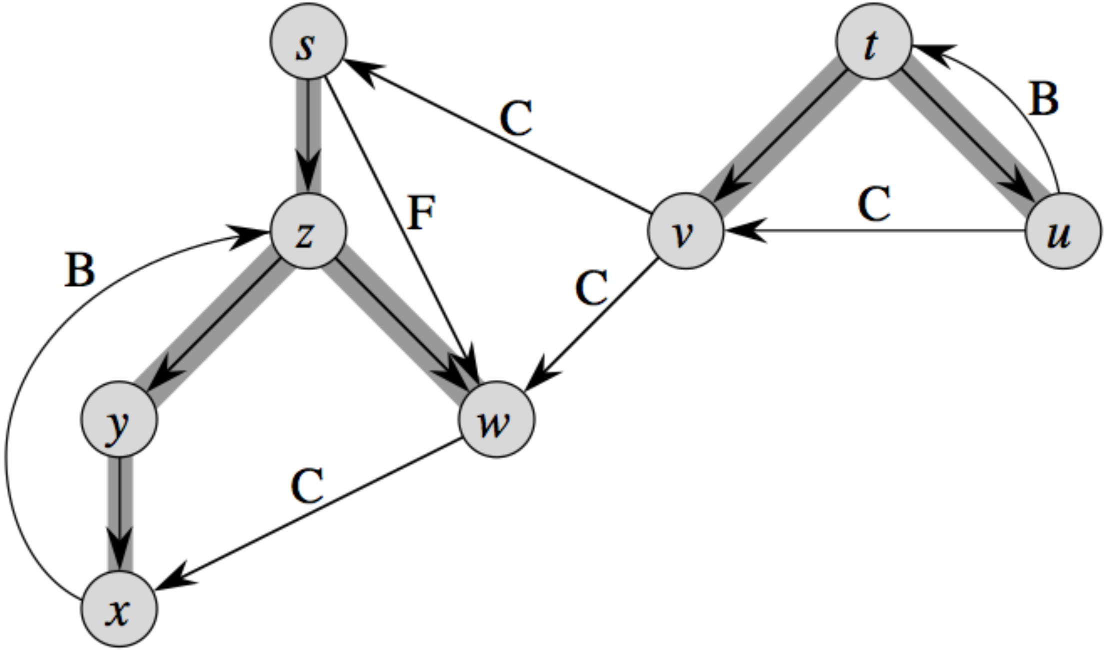
Clasificación de arcos

El DFS nos clasifica los arcos de un grafo en 4 categorías

- **Tree** edge:
  - Cuando viaja a un nodo blanco
  - Viaja a un hijo
- **Back** edge:
  - Cuando viaja a un nodo gris
  - Viaja a un ancestro
- **Forward** edge
  - Cuando viaja a un nodo negro
  - El nodo es descendiente
- **Cross** edge
  - Cuando viaja a un nodo negro
  - El nodo no es descendiente

Clasificación de arcos

(c)



Propiedades

- Los **forward** y **cross** edges sólo aparecen en grafos dirigidos
- Un grafo es acíclico si y sólo si no existen **back** edges
- 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).
- Los **tree** edges conforman un spanning forest del grafo
- En grafos no dirigidos, cada árbol del spanning forest corresponde a una componente conexa

Orden topológico

Un orden topológico sobre un grafo acíclico dirigido o DAG es un ordenamiento de los nodos tales que si existe el arco (u,v) u aparece antes que v en el orden

Si el grafo tiene ciclos, no hay orden topológico posible

Los DAGs suelen usarse para indicar precedencia entre eventos.

Orden topológico

Se puede utilizar las timestamps dadas por el DFS.

Si hay arco (u,v) entonces f[v] > f[u]

Demostración

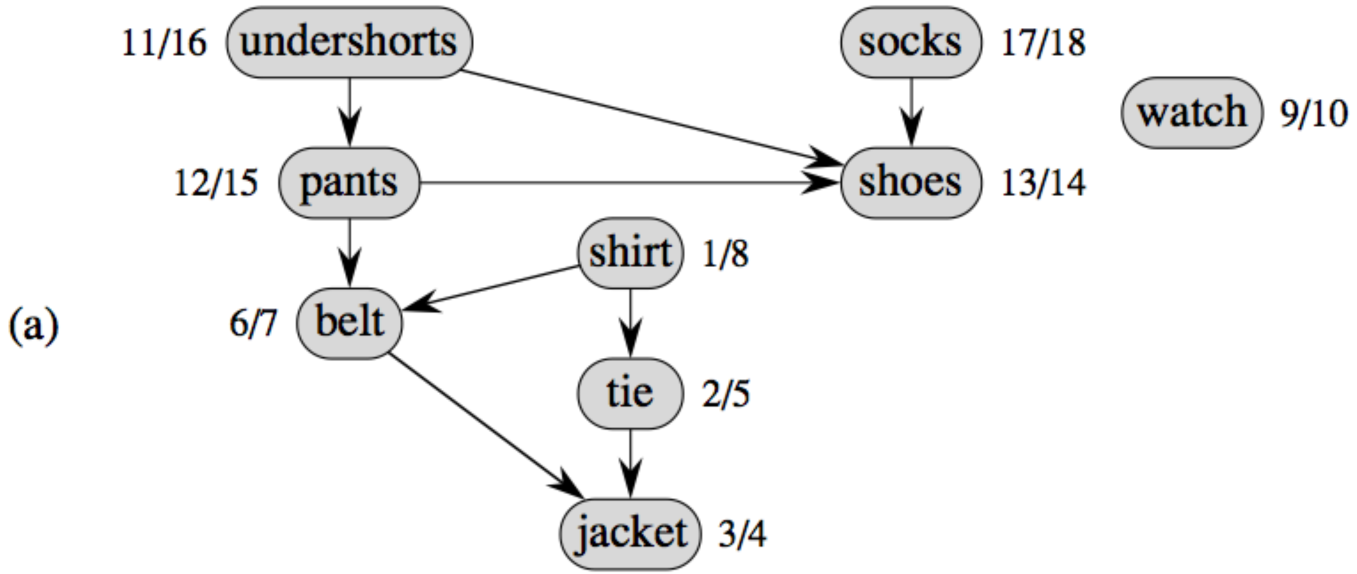
(u,v) es **tree** o **forward** edge ⇒ d[u] < d[v] < f[v] < f[u]

(u,v) es **cross** edge => d[v] < f[v] < d[u] < f[u]

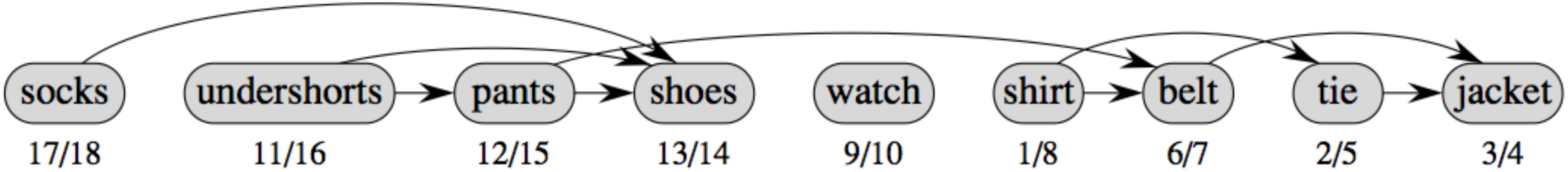
(u,v) NUNCA es **back** edge porque el grafo es acíclico

El orden topológico se obtiene haciendo un DFS y ordenando por f[u] decreciente.

Ejemplo: Vestirse



Ejemplo: Vestirse

(b) 

Caminos más cortos en un DAG

Ya conocemos algoritmos para caminos más cortos:

- Dijkstra: $O(V^2)$ o $O(E \log(V))$
- Bellman-Ford: $O(VE)$
- Floyd-Warshall: $O(V^3)$

Para DAGs se puede hallar caminos más cortos en $O(V+E)$

Pseudocódigo

```
Ord = Orden topológico de G
dist[] = {∞,∞,....,∞}
dist[s] = 0

for v in Ord
    for (u,v) in G
        dist[v] = min(dist[v], dist[u] + costo(u,v))
```

Biconectividad: Definiciones

Dado un grafo no dirigido y conexo G = (V,E)

- Un punto de articulación es un nodo tal que al removerlo se obtiene un grafo disconexo.
- Un puente es un arco tal que al removerlo se obtiene un grafo disconexo.
- Un grafo es biconexo si no tiene puntos de articulación.
- Las componentes biconexas son los subgrafos biconexos maximales

Relaciones de equivalencia - Particiones

Las componentes fuertemente conexas particionan al conjunto de nodos de un grafo.

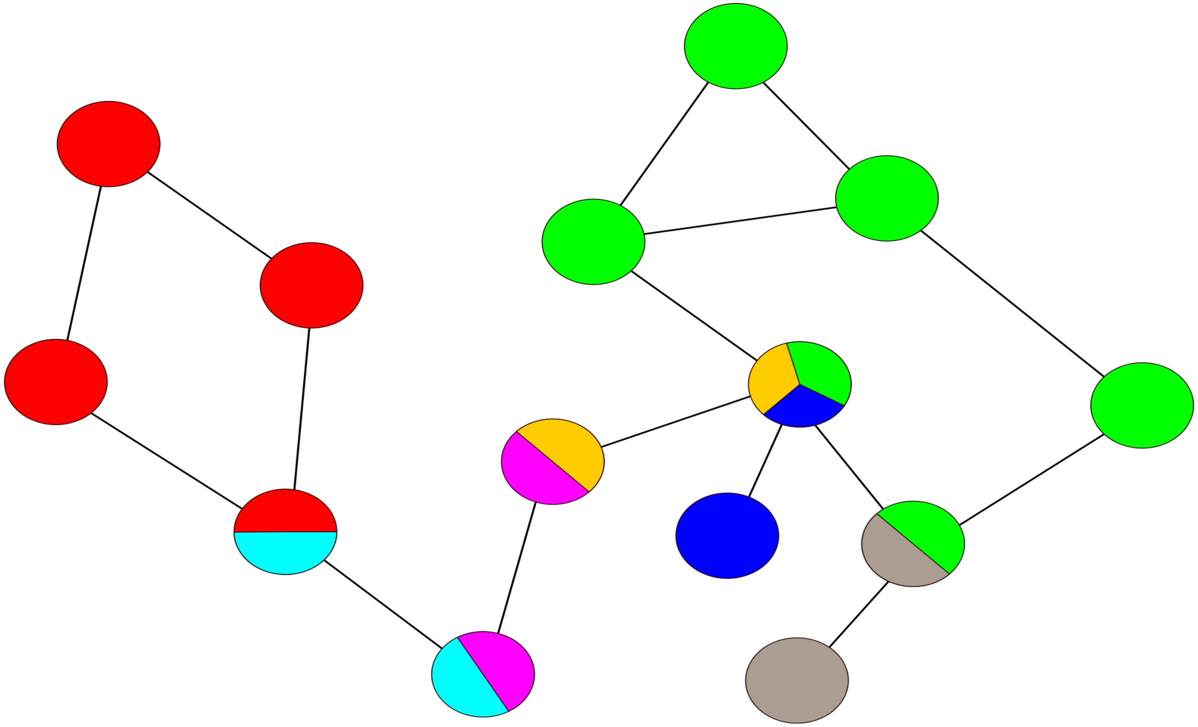
Se particionan por la relación de equivalencia alcance mutuo

$u \equiv v \Leftrightarrow u \rightsquigarrow v$ y $v \rightsquigarrow u$

Las componentes biconexas particionan los arcos.

La relación de equivalencia es cociclidad. Los arcos a y b son equivalentes si a = b o pertenecen a un mismo ciclo simple.

Gráfico



Los puentes serán las clases de equivalencia unitarias

Los puntos de articulación son los nodos con más de un color

Puntos de articulación: Intuición

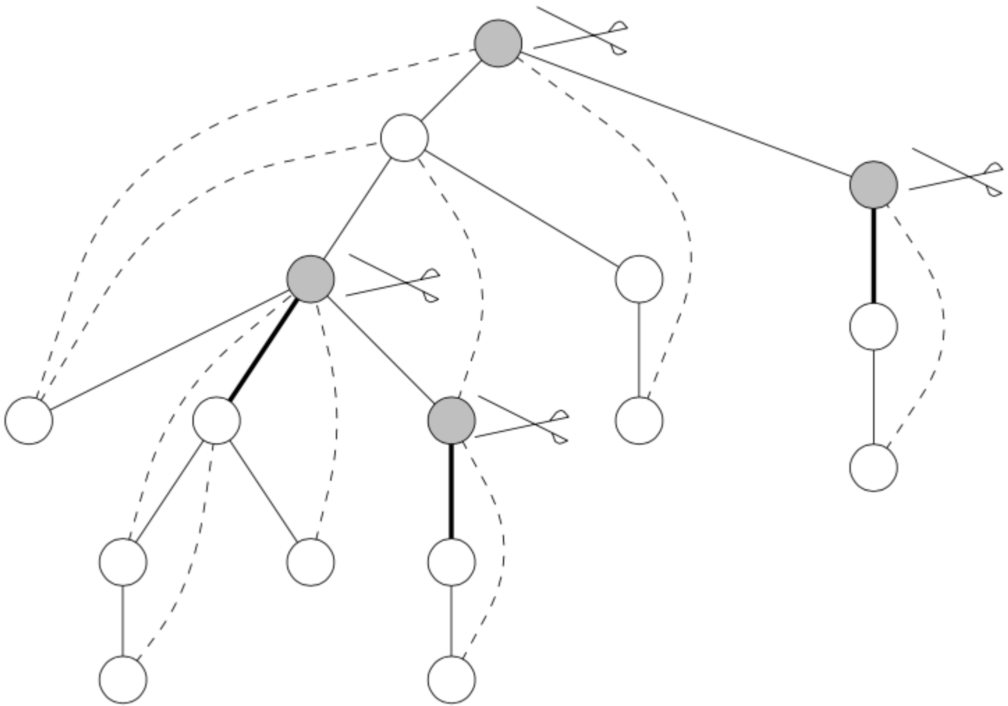
Dado que el grafo es no dirigido sólo tendremos **tree** edges y **back** edges.

Un nodo será punto de articulación cuando al eliminarlo, alguno de sus subárboles hijos queda "descolgado".

Es decir cuando no tengamos una back edge que lo salve.

La raíz es un caso especial.

Árboles descolgados



low[u]

Para calcular qué árboles se "descuelgan" definimos el valor low[u]

low[u] = min {d[w] | (v,w) es back edge y w es descendiente de u}

Es decir, para cada subárbol del DFS calculamos cuán alto podemos llegar mediante una back edge.

low[u]

low[u] = mínimo entre

- d[u]
- d[w] para cada (u,w) back edge
- low[v] para cada v hijo de u

Pseudocódigo

```
dfs(u)
    d[u] = t++
    low[u] = d[u]
    color[u] = gris
    for v in adj[u]
        if color[v] = blanco
            dfs(v)
            low[u] = min(low[u], low[v])
        if colov[v] = gris and v≠padre[u]
            low[u] = min(low[u], d[v])
```

Criterios

El nodo u es punto de articulación si:

- Para algún hijo v, low[v] >= d[u]
- Si u es la raíz y tiene más de un hijo

El arco (u,v) es puente si:

- low[v] > d[u]   asumiendo que u es el padre de v
- las back edges nunca son puentes

Problema: Street Directions

Street Directions - UVA Online Judge
https://uva.onlinejudge.org/external/6/610.html

Dada una ciudad conectada con calles bidireccionales tal que se puede llegar de cualquier punto a cualquier otro (es conexa), se quieren transformar en unidireccionales la mayor cantidad de calles posible de manera que siga siendo posible llegar de un punto a cualquier otro

Problema: Necklace

Necklace - ICPC Archive 4271

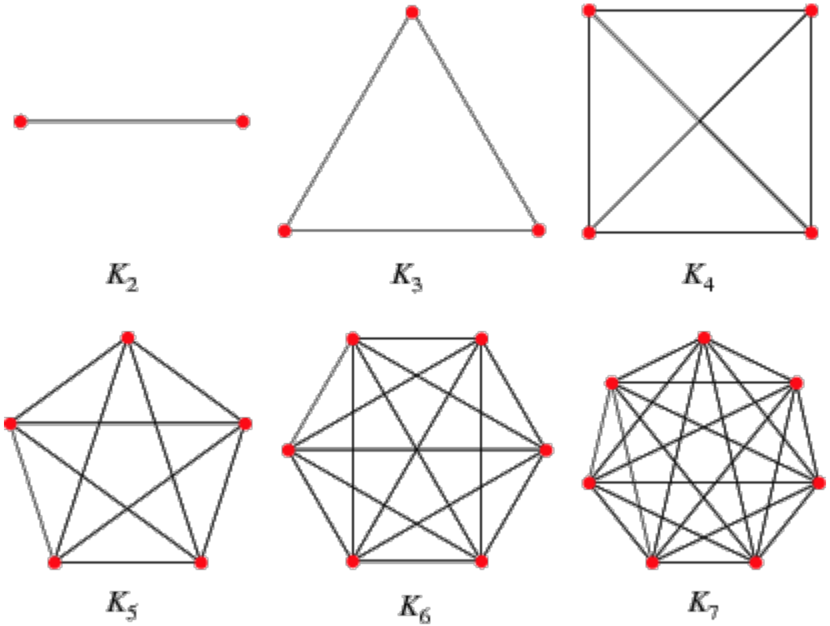
https://icpcarchive.ecs.baylor.edu/index.php?option=onlinejudge&page=show_problem&problem=2272

Dado un grafo y dos nodos S y T. Decidir si S y T se pueden conectar mediante un collar. Un collar es una secuencia de ciclos sin arcos en común y con un sólo vértice en común entre ciclos consecutivos.

Parte II
Problemas intratables

Clique

Un clique es un grafo donde hay un arco entre todo par de nodos



Maximum Clique

El problema de maximum clique pide encontrar el subgrafo más grande un grafo dado que es clique.

Es un problema NP-hard

Si lo resuelven en tiempo polinomial, resuelven un problema abierto hace 50 años y se ganan un millón de dólares!

Independent Set

Un independent set de un grafo es un conjunto de nodos tal que no hay arcos de entre ellos

El problema de maximum independent set consiste en encontrar un conjunto de máxima cardinalidad que cumpla con esta condición.

También es NP-hard

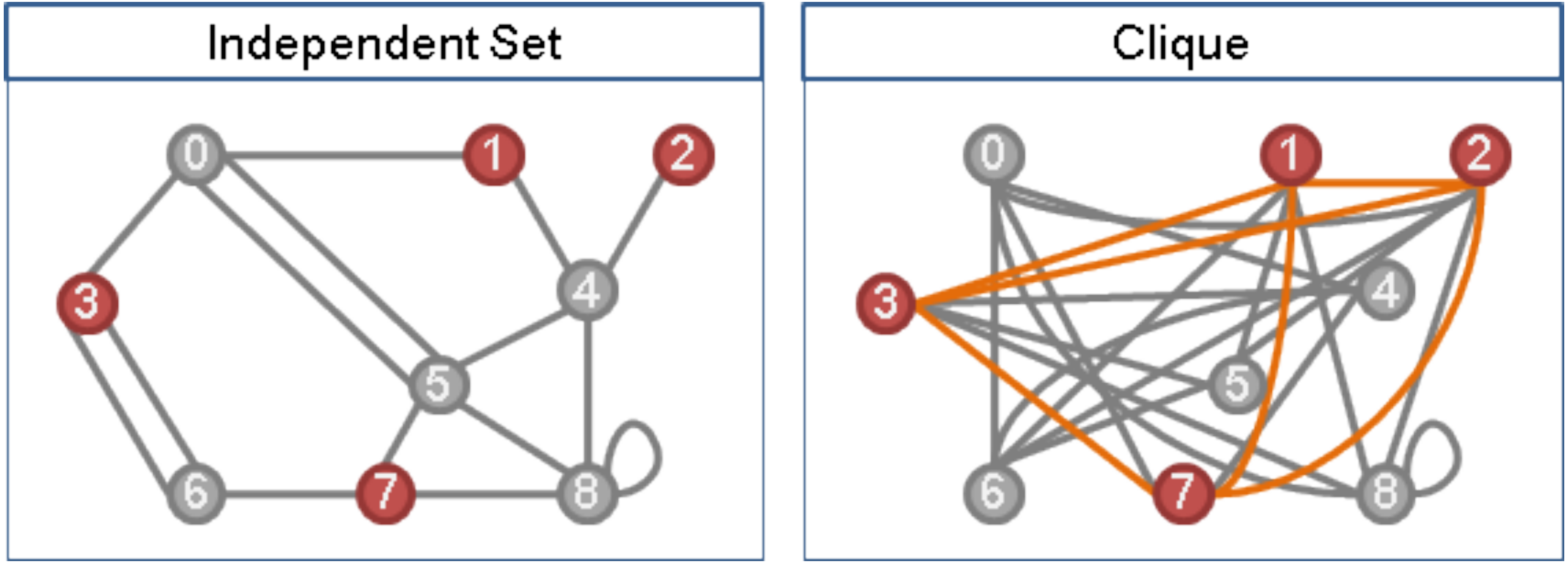
Vertex Cover

Un vertex cover es un conjunto de nodos tales que todo arco tiene al menos un extremo en el conjunto

El problema de minimum vertex cover exige hallar el menor conjunto que cumple esta propiedad.

Y es…   NP-hard

Reducción



Solución en bipartitos

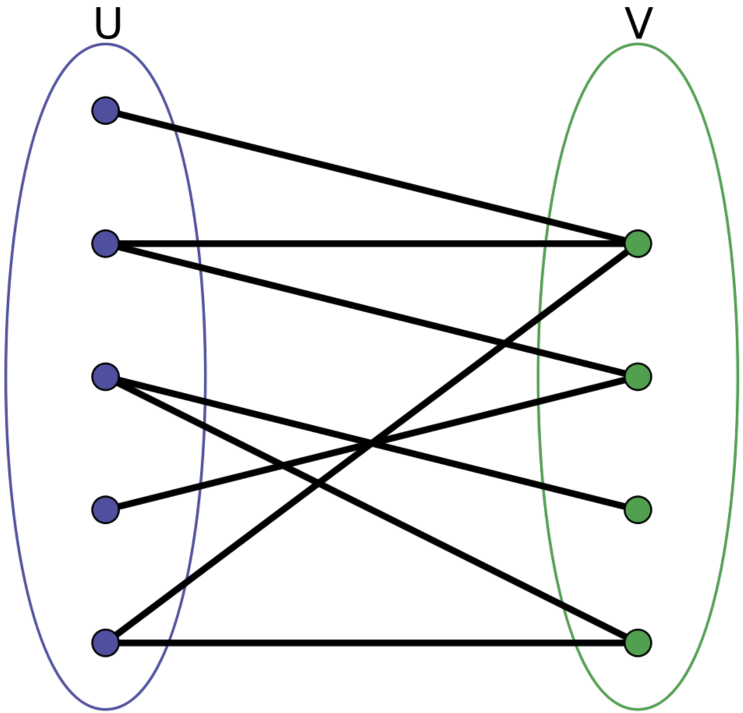
Los 3 problemas que mencionamos son NP-hard.

Pero existen soluciones eficientes para ciertas grafos restringidos

En particular, para grafos bipartitos existen soluciones polinomiales para los problemas de vertex cover e independent set.

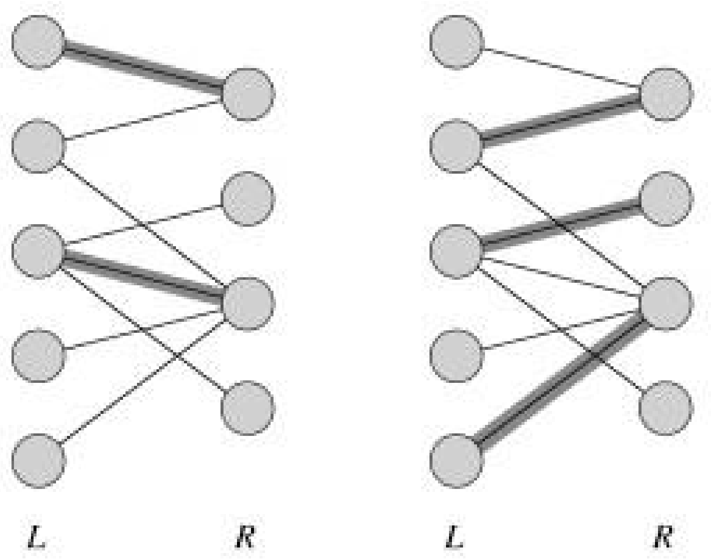
Grafo bipartito: Definición

Un grafo es bipartito si es posible separar los nodos en 2 conjuntos U y V tales que todos los arcos tienen un extremo en U y otro en V



Bipartite matching

Un matching es un conjunto de arcos tales que cada nodo es extremo de a lo sumo un arco



Teorema de König

El teorema de König dice que en un grafo bipartito, la cardinalidad de un vertex cover es igual a la de un matching máximo.

Es fácil ver que |VC| >= |M|

La demostración del teorema da una construcción de un vertex cover a partir de un matching donde se cumple la igualdad.

Problema: Kamehameha

Kamehameha - Codechef OCT13

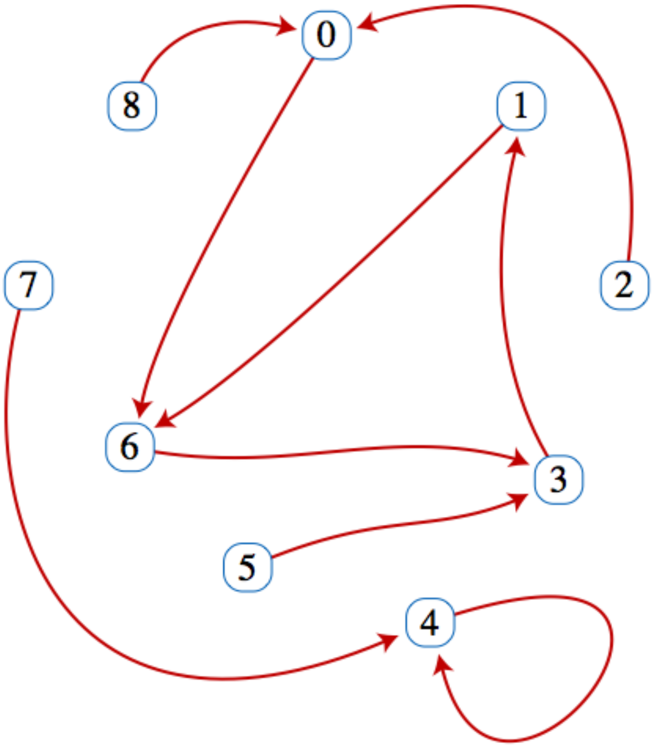
https://www.codechef.com/problems/KMHAMH

Dada una grilla de NxN (N <= 1000) donde algunas casillas tienen demonios. Goku con un kamehameha puede matar a todos los demonios de una fila o de una columna.

Para no cansarlo, hay que encontrar la mínima cantidad de kamehamehas para matar a todos los demonios

Grafos funcionales

x	$f(x)$
0	6
1	6
2	0
3	1
4	4
5	3
6	3
7	4
8	0



Grafos funcionales: Estructura

Si pensamos con los arcos no dirigidos, cada componente conexa tiene la misma cantidad de nodos y arcos.

Es como un árbol con una back-edge, o sea una componente conexa con exactamente un ciclo

Se puede demostrar que el ciclo está dirigido.

Los nodos que no conforman el ciclo, están dispuestos en forma de árboles incrustados en el ciclo.

Grafos funcionales: Vertex Cover

Hay algoritmo greedy empezando de las hojas (nodos de grado 1)

Tomamos una hoja, nunca es óptimo seleccionarla para el vertex cover. Por lo tanto seleccionamos a su padre y borramos todos los arcos cubiertos.

Al final, es posible que queden ciclos. Para estos se necesitan $\lceil \#nodos/2 \rceil$

Problema: The Explosion

The Explosion - SPOJ

http://www.spoj.com/problems/EXPLOSN/

Hay N sospechosos de un atentado (N <= 100.000). Cada uno de ellos menciona a exactamente a una persona como culpable. Se sabe que los inocentes siempre dicen la verdad y los culpables siempre mienten. Cuál es la mínima cantidad de culpables que puede haber que sea consistente con las respuestas dadas?