Graphs and Complexity

5) Testing graph properties

Irena.Rusu@univ-nantes.fr

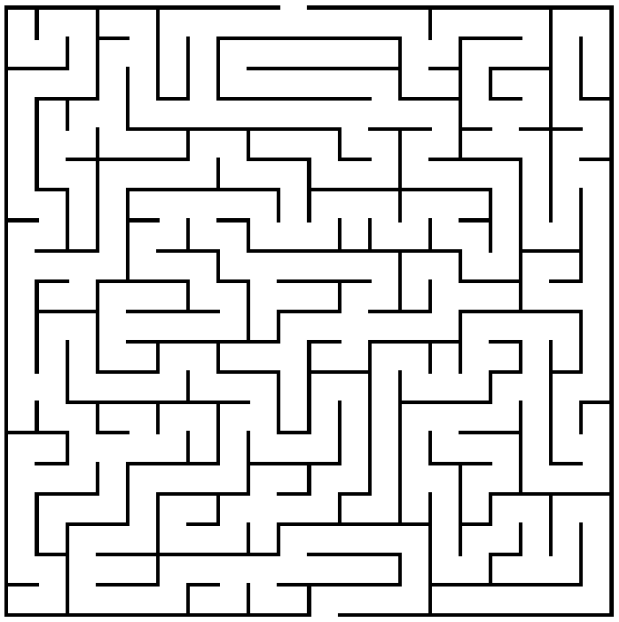
LS2N, bât. 34, bureau 303

tél. 02.51.12.58.16

Plan

- Does G have a cycle/circuit?
- Graphs with no cycle
  - Undirected case: trees
  - Directed case: DAGs
- Graphs with no odd cycles

Does G have a cycle/circuit?



© Obtained using https://www.mazegenerator.net/

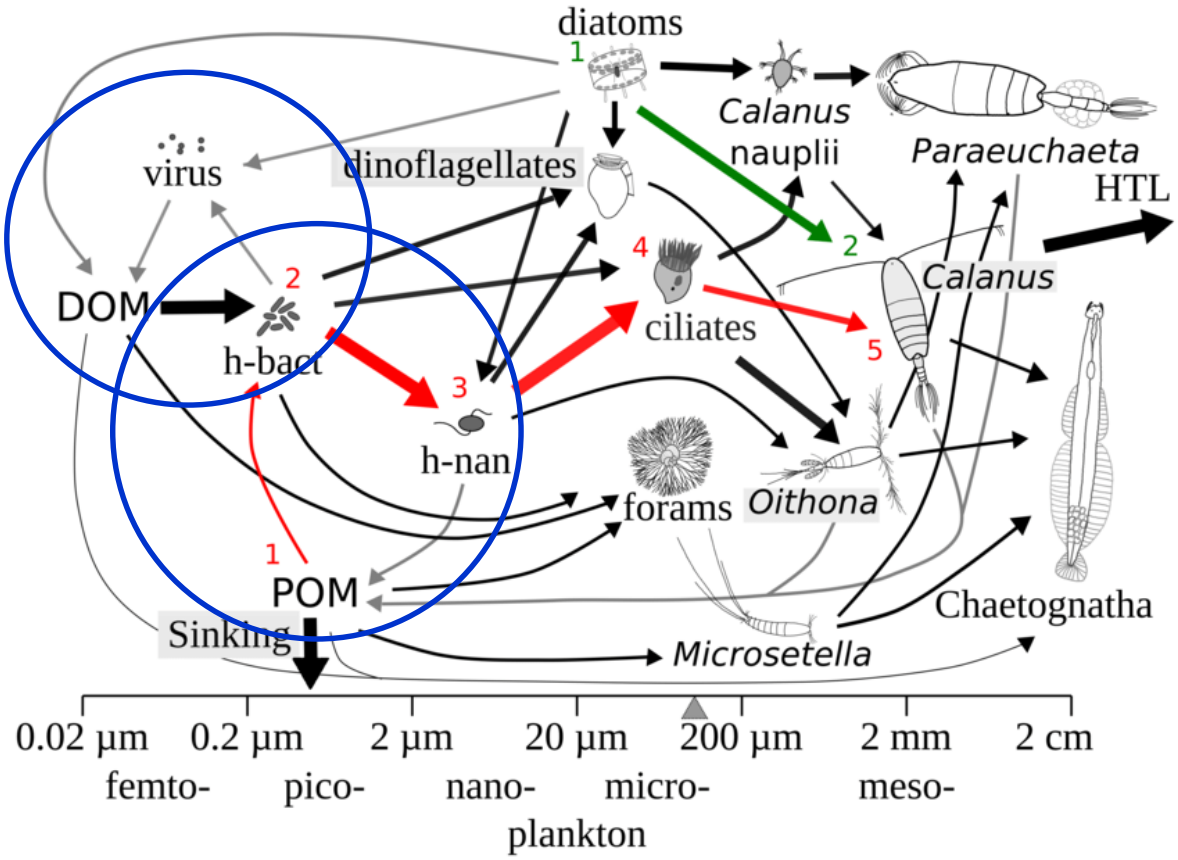
- Is this a safe maze ?
- … i.e. not allowing to walk along the same « loop » again and again?

Graph:

- Unit cells in the (imaginary) grid are the vertices
- Cells with a common (thus invisible) side define edges
- « loop »=cycle

Does G have a cycle/circuit?

- Food web, from North Atlantic.
- Several circuits are present.



(c) Sünie L. Basedow et al., Journal of Plankton Research, 2016

In digraphs, we have a solution

- Find the strong connected (digraph) components, and conclude.

- Advantage: already written
- Drawbacks:
  - Once a cycle is found, the algorithm doesn't stop
  - If we need a cycle (and usually we do), the algorithms do not allow to identify it

In undirected graphs, theory helps

$G=(S,A)$ undirected graph $|S|=n$, $|A|=m$

Property

If $m \geq n$ then G contains a cycle.

Proof.

$m \geq n \rightarrow$ there is a CC named C with $m(C) \geq n(C)$

Remove all vertices with degree = 1 from C.

Then C is not empty and is still connected.

Since all vertices still in C have degree ≥ 2, a cycle exists.

Remarks

- The proof indicates a way to find a cycle (DFS search)
- Property not immediately adaptable to digraphs.
- What if $m<n$?

Algorithm for testing the existence and (if yes) finding a cycle in all graphs (directed or not)?

Reminders

G=(S,A) directed or not (works similarly)

Function DFSNum (*s* vertex of *G*)

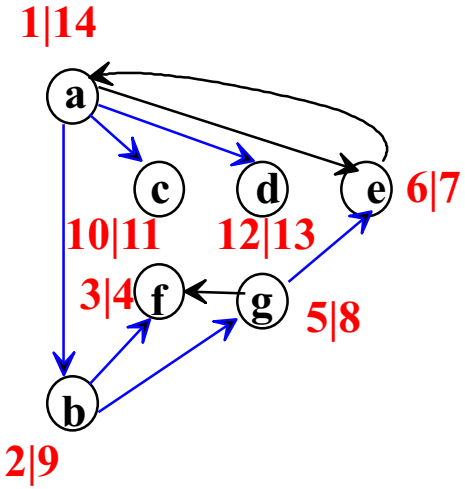
```
begin
    nb ← nb + 1 ; d [s] ← nb ;
    for each successor t of s do
        if d [ t ] = 0 then
            DFSNum ( t ) ;
        endif
    endfor
    nb ← nb+1 ; f [s] ← nb
end
```

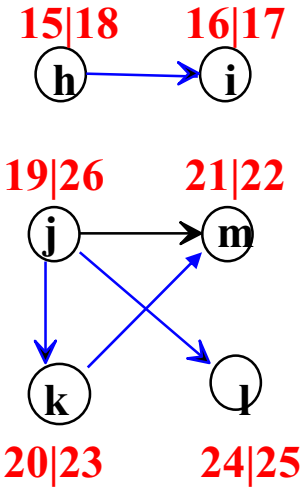
Visiting the whole graph *G*

```
for each vertex s of G do
    d [s] ← 0 ; f [s] ← 0 ;
endfor
nb ← 0 ;
for each vertex s of G do
    if d [s] = 0 then
        DFSNum (s)
    endif
endfor
```

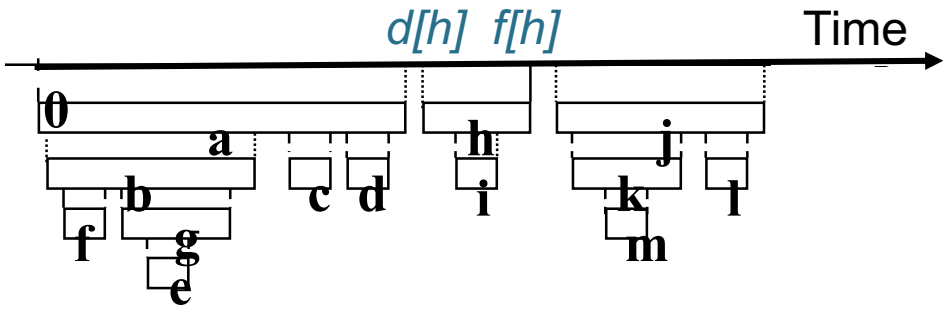
[d[s]..f[s]] exploration of *s*

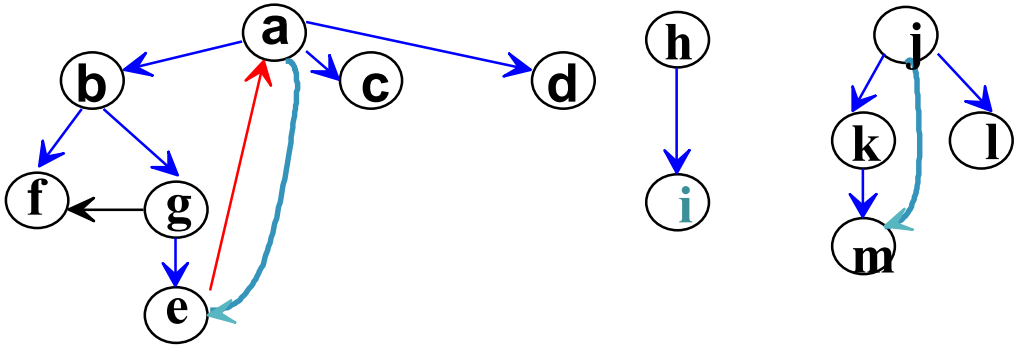
Running time: $\Theta(m+n)$ *or* $\Theta(n^2)$





DFSNum defines interesting arcs





« tree » arc

« forward » arc

« back » arc

« cross » arc

Idea for an algorithm

$G=(S,A)$ graph (directed or not)

Property.

G has a cycle if and only if there is a back arc in a DFS search of G.

Note. In this case, each DFS search has a back arc.

Property.

(s,t) arc of G is a(n)

- tree or a forward arc iff $d[s] < d[t] < f[t] < f[s]$
- back arc iff $d[t] < d[s] < f[s] < f[t]$
- cross arc iff $f[t] < d[s]$

First algorithm

- Perform DFSNum
- Test each arc, and stop when a backward arc is found.

Running time: $\Theta(m+n)$ *or* $\Theta(n^2)$

Does not find a cycle (and it is not easy to retrieve one)

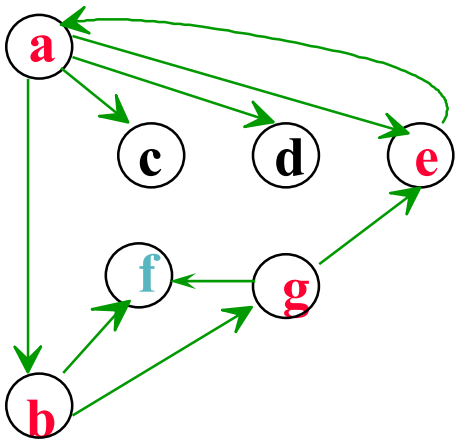
Better algorithm

During a DFS search:

state [s] = black — s not reached yet by the search

state [s] = red — the exploration of s (and its successors) started but it is not finished yet

state [s] = blue — the exploration of s (and its successors) is over.



During the exploration of vertex e, one finds a cycle using the arc (e, a) since a is also explored.

Running time: $\Theta(m+n)$ *or* $\Theta(n^2)$

Algorithm → TD !

Plan

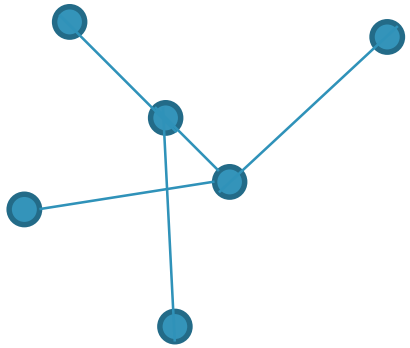
- Does G have a cycle/circuit?
- Graphs with no cycle
  - Undirected case: trees
  - Directed case: DAGs
- Graphs with no odd cycles

Undirected case: (unrooted) trees

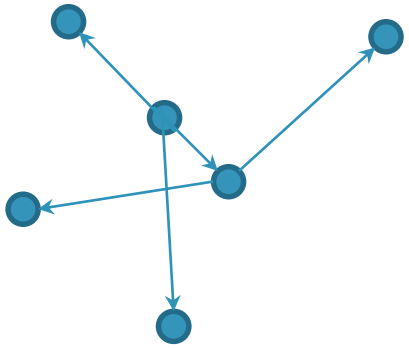
G=(S,A) undirected graph

G is a tree if it is connected and contains no cycle.

G is a forest if it is not connected and contains no cycle.



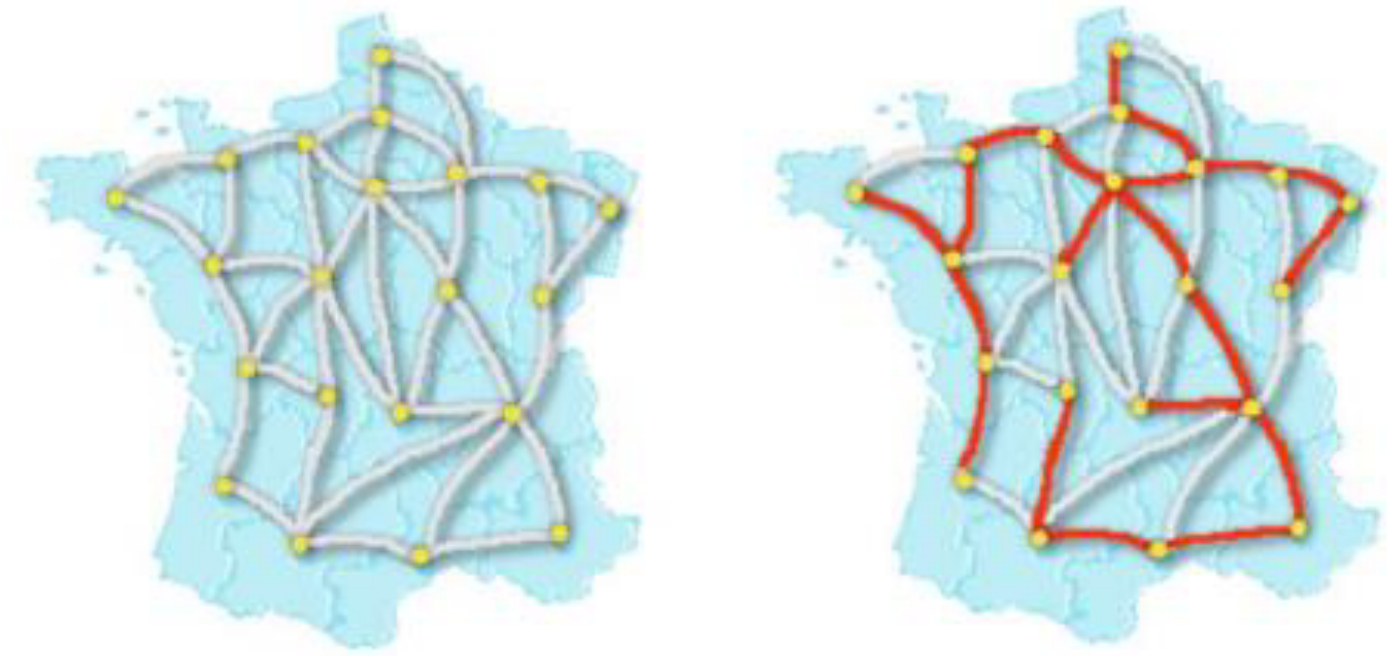
Tree



Rooted tree

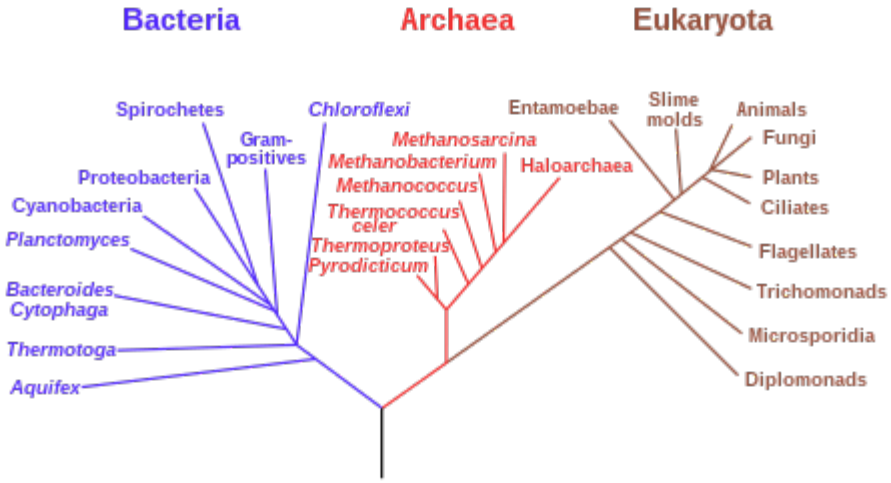
Applications

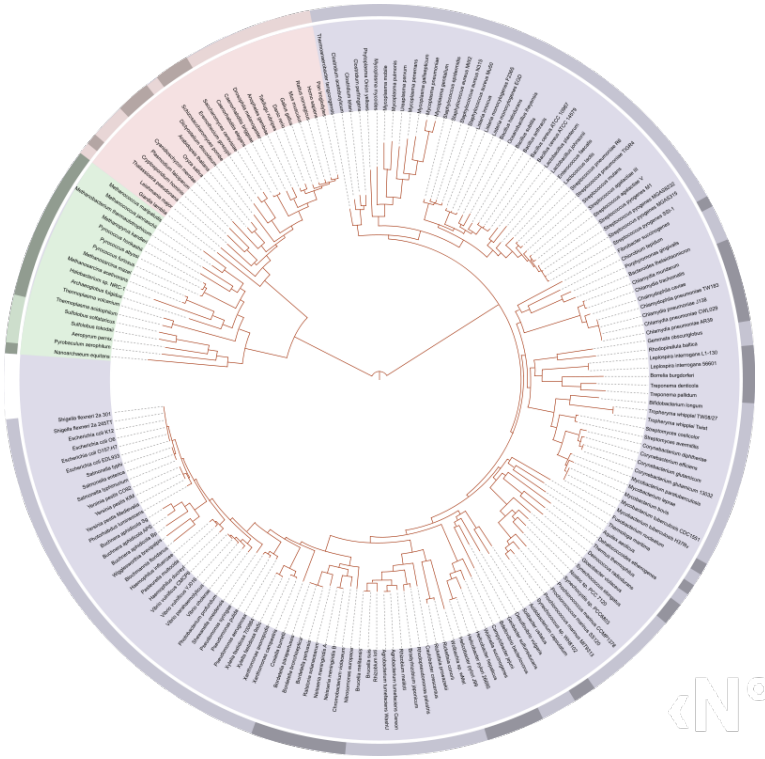
Minimum spanning tree



Applications

Phylogenetic trees





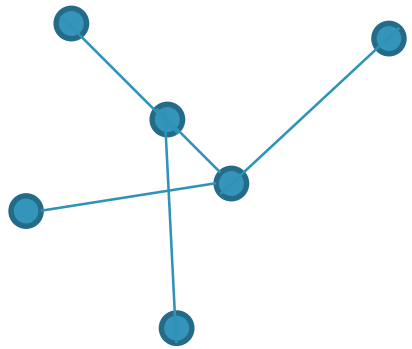
Characterization of trees

Theorem Let G be an undirected graph with n vertices ($n \geq 1$).
The following properties are equivalent:

G is connected and has no cycle

G has no cycle and has exactly $n-1$ edges

G is connected and has exactly $n-1$ edges



G has no cycle and the addition of any edge creates a unique cycle.

For all s,s' from S ($s \neq s'$), there is a unique path joining s and s'.

G is connected and the suppression of any edge makes G non-connected.

Is G a tree ?

$G=(S,A)$ undirected graph $n=|S|$, $m=|A|$

Goal: algorithm for testing whether G is a tree or not.

First algorithm (already a good one):

- Test the connexity
- Test for the existence of a cycle

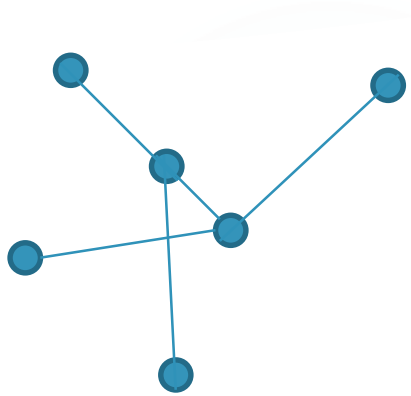
Easier algorithm:

- Test the connexity
- Test whether $m=n-1$.

One may use any characterization !

Is G a tree?

- Why graphs that are connected and without cycles (trees) always look like that ?



Leaf: vertex of degree 1.

Property.

Each tree G with $n \geq 2$ vertices has at least two leaves.

New idea for an algorithm

Algorithm (rough description)

As long as possible do

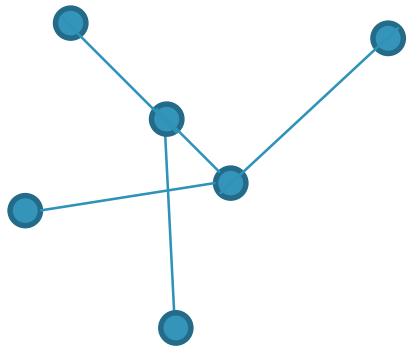
    Remove one leaf and its incident edge

If the result is one vertex then

    G is a tree

Else

    G is not a tree.



Best running time:
$\Theta(m+n)$

Theorem showing the correction (1)

Theorem

G is a tree if and only if the algorithm ends with one vertex.

Sketch of proof.

⇒: G is a tree → $G - x$ (where x is a leaf) is a tree

Recurrence: Each G' obtained during the algorithm is a tree.

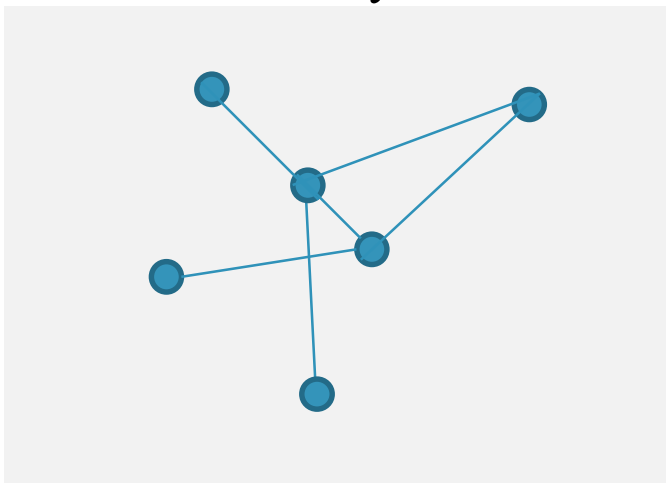
The trees are smaller and smaller, thus after $n-1$ steps one gets the tree with one vertex and no edge.

Theorem showing the correction (2)

⇐: If the algorithm ends with one vertex, then

- G does not contain cycles
- G is connected 

With a cycle



Not connected

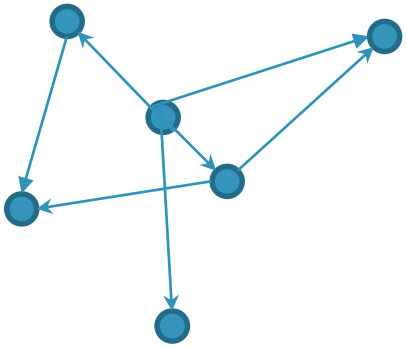
Plan

- Does G have a cycle/circuit?
- Graphs with no cycle
  - Undirected case: trees
  - Directed case: DAGs
- Graphs with no odd cycles

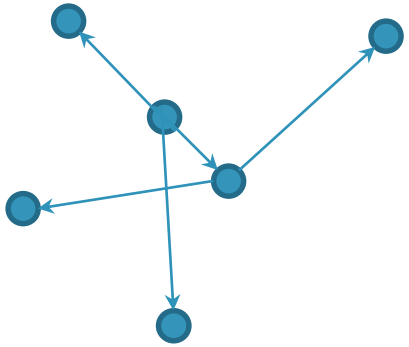
Directed acyclic graphs

$G=(S,A)$ directed graph

G is a directed acyclic graph (DAG) if it contains no cycle.



A (general) DAG



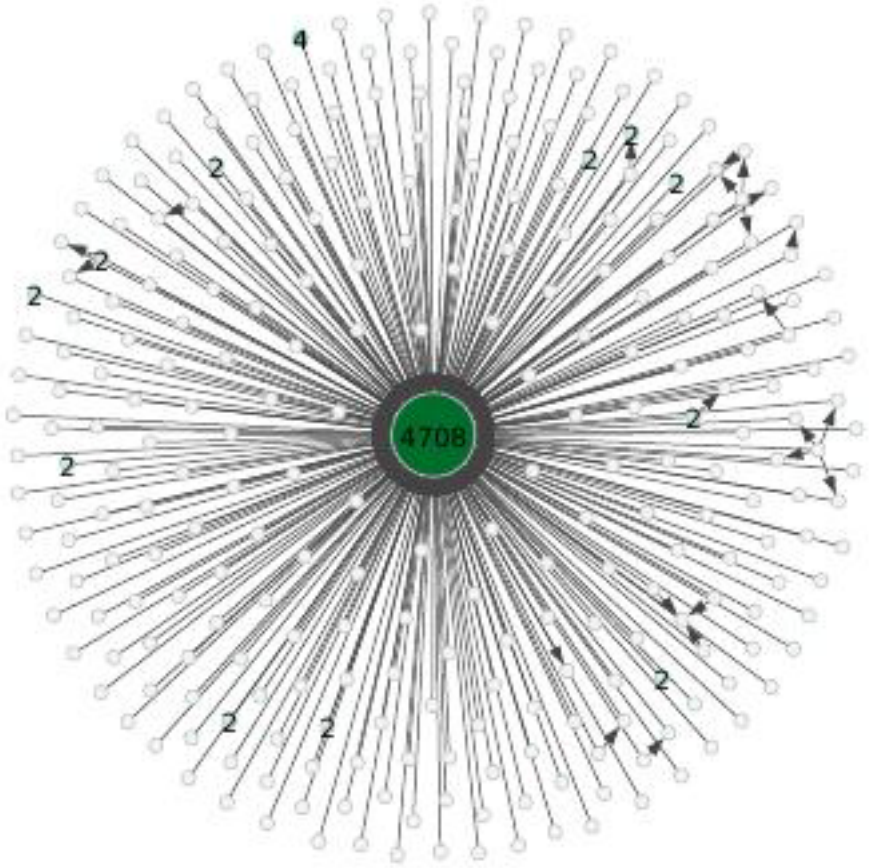
A rooted tree

(particular type of DAG)

Applications

A DAG represents a partial order, for instance this one:

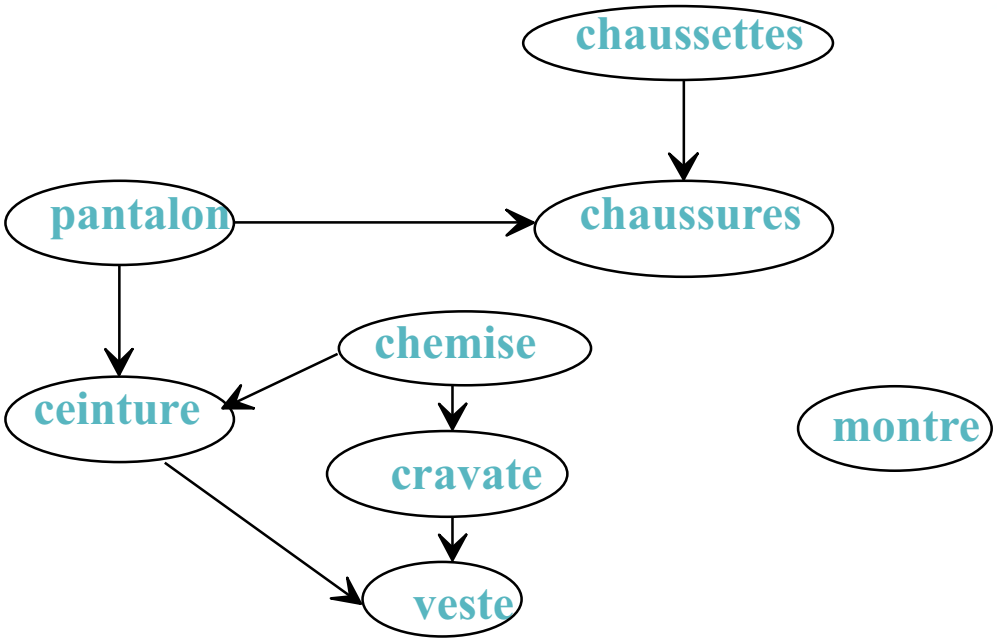
(Forget the numbers and recall the graph is obtained by shrinking the SCC into one vertex each.)



(b) 5K-ELITE

Applications

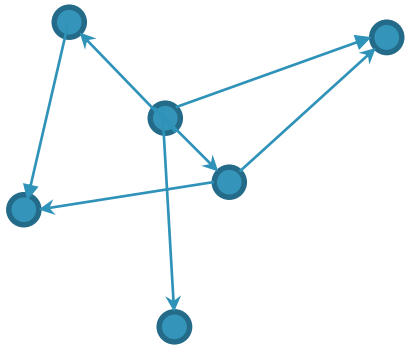
Or this one:

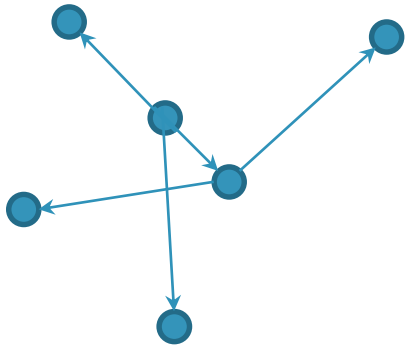


Is G a DAG?

Property

Each DAG *G* has at least one vertex with indegree equal to 0.





Easy algorithm follows

Algorithm (rough description)

As long as possible do

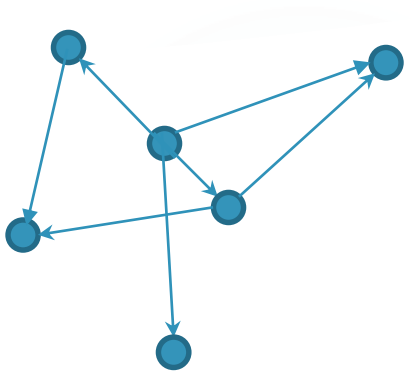
Remove one vertex of indegree 0 and its outgoing arcs

If the result is the empty graph (no vertex) then

G is a DAG

Else

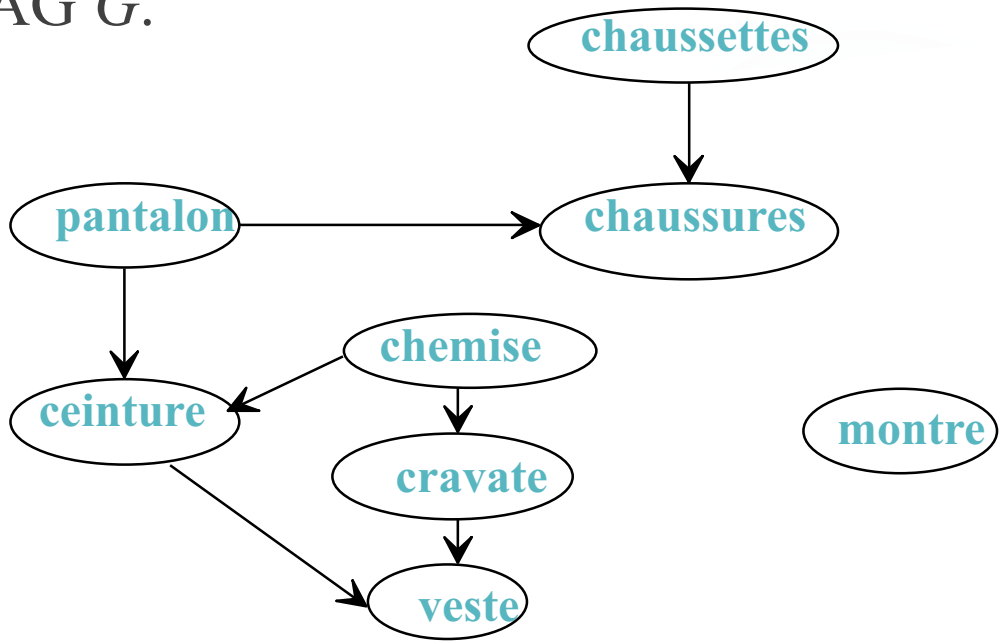
G is not a DAG.



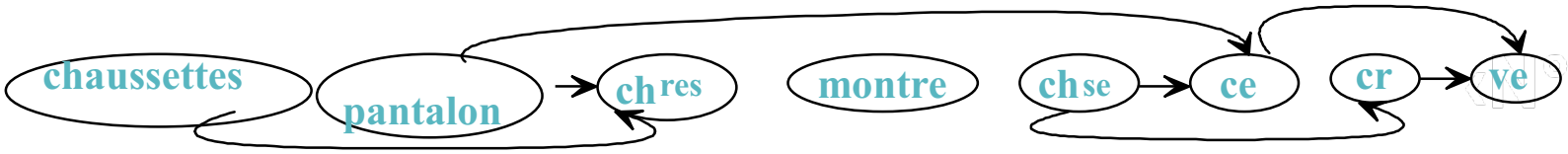
Best running time:
$\Theta(m+n)$

Topological sorting

Topological sorting: find a total order compatible with the partial order represented by the DAG *G*.



Find a correct dressing order:



Algorithm

First idea:

- Similar to the algorithm for testing whether G is a DAG.
- The sequence of degree 0 vertices chosen by the algorithm is the total order.
- Works fine but it's not so nice (too basic).

Best running time:
$\Theta(m+n)$

Another idea (much more subtle):

- Perform a DFSNum search
- Order the vertices in decreasing order of the values $f[s]$

Simplification *d[s]* useless

only the order of the values *f[s]* is needed, and these values are computed in increasing order

```
Function Topo(s vertex of G) : void
Begin
    visited[s] ← true;
    for each successor t of s do
        if not visited[s] then
            Topo(t)
        endif
    endfor
    L ← Push(L,s)
end
```

```
Function TopologicalSort(G : DAG) : stack
Begin
    for each vertex s of G do
        visited[s] ← false;
    endfor
    L ← Empty_stack
    for each vertex s of G do
        if not visited[s] then
            Topo(s)
        endif
    return L
end
```

Best running time: $\Theta(m+n)$

Plan

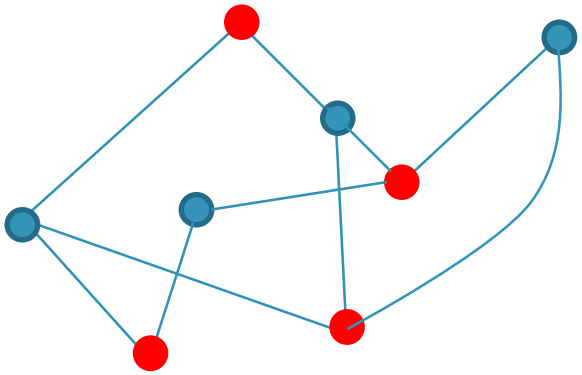
- Does G have a cycle/circuit?
- Graphs with no cycle
  - Undirected case: trees
  - Directed case: DAGs
- Graphs with no odd cycles

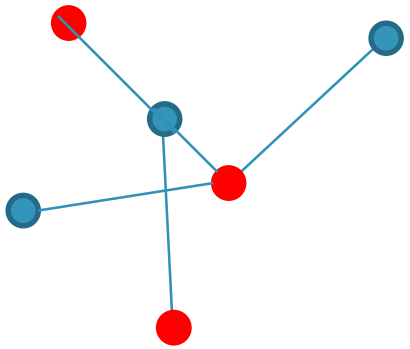
Bipartite graphs

$G=(S,A)$ undirected graph (may be defined for digraphs too)

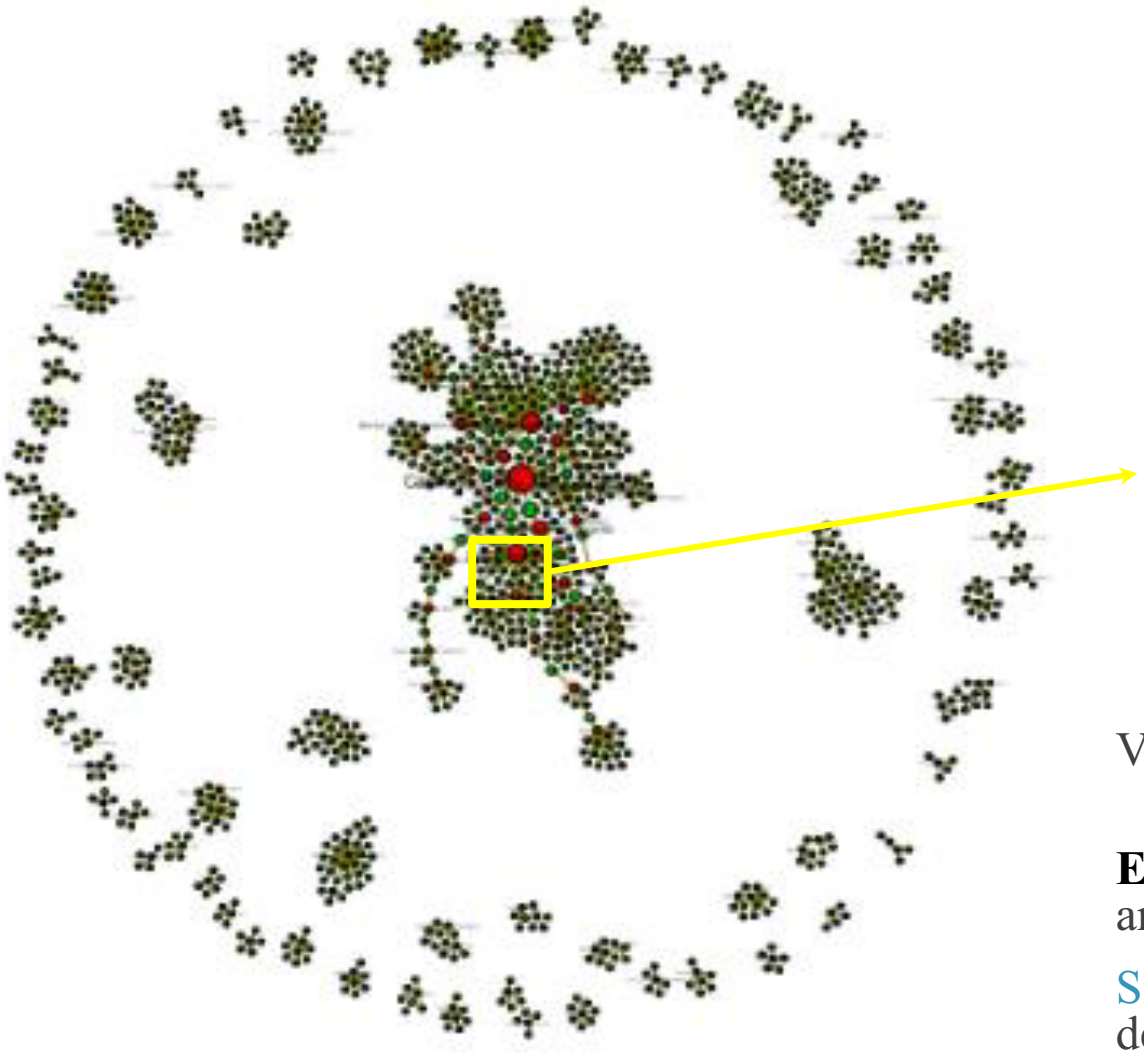
Graph G is a bipartite graph if S can be written as $S=X \cup Y$ such that:

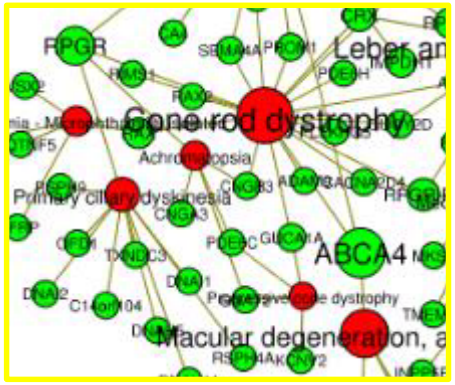
- $X \cap Y = \emptyset$, and
- each edge has one endpoint in X and the other in Y.





Applications





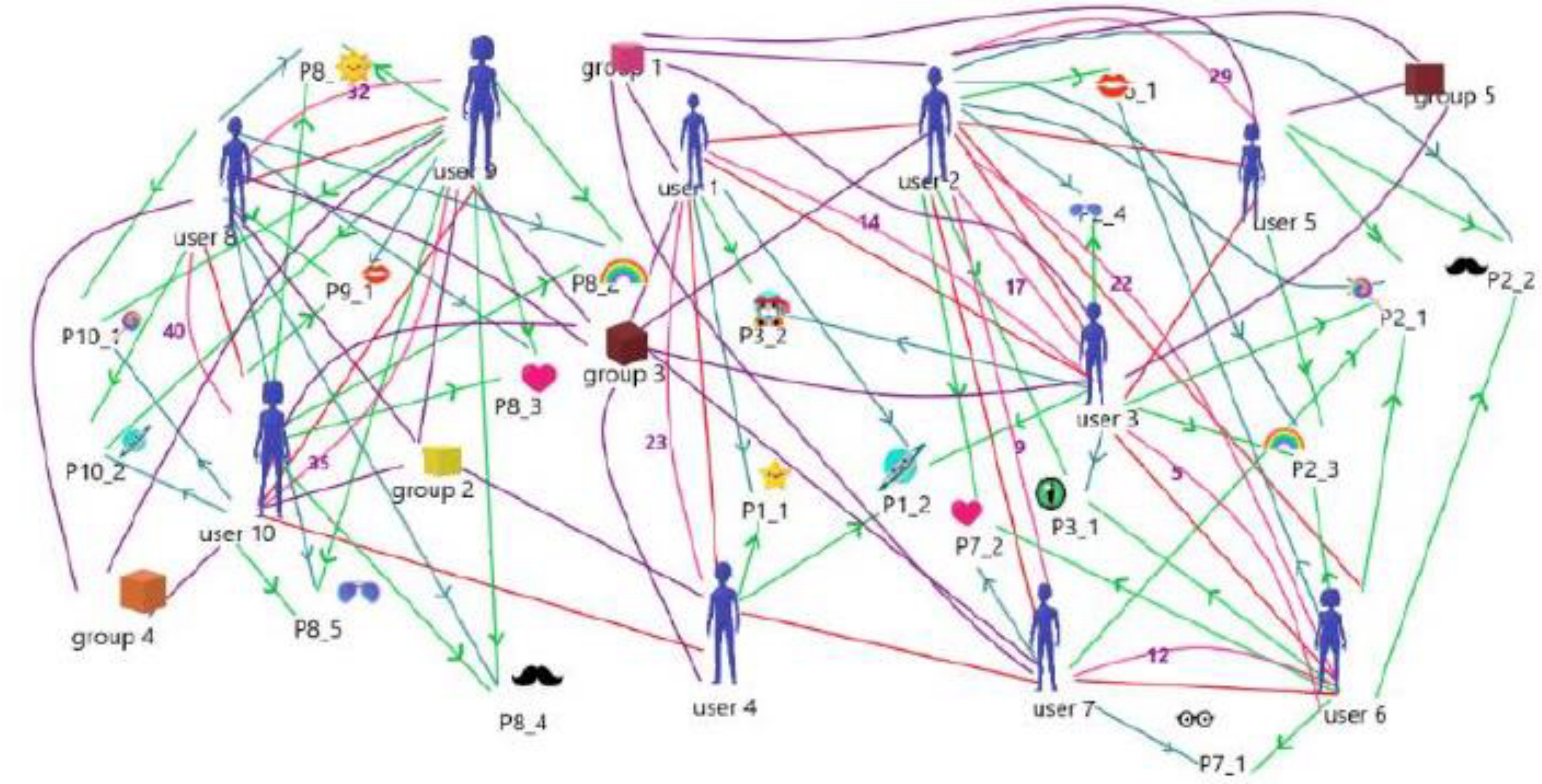
© https://research.cchmc.org/od/04/

Vertices : orphan diseases and involved genes

Edge between any disease and any gene involved in the disease

Size of a vertex : illustrates its degree

Representation of the Facebook network



Vertices: users, posts and groups.

Edges/arcs: friendship, reactions on posts, creation of posts, messages and group membership.

The group membership subgraph is a bipartite graph.

Is G a bipartite graph?

$G=(S,A)$ undirected graph

Property

G is a bipartite graph if and only if it does not contain cycles with an odd number of vertices (called odd cycles)

Idea of an algorithm:

- Perform a BFS search
- Compute the level of each vertex before adding it to the queue
- If one finds a vertex that has a neighbor on its own level then G is not bipartite
- Otherwise G is bipartite

Algorithm

```
Function BFSbip (s vertex of G) : void
begin
    Q ← Enqueue (Empty-Queue, s) ;
    while not IsEmpty (Q) do
        s' ← First (Q) ; Q ← Dequeue (Q, s') ;
        for each successor t of s' do
                if level[t]=level[s'] then
                        bool ← false
                else
                        if level[t]=-1 then
                                level[t] ← level[s]+1
                                Q ← Enqueue (Q, t)
                        endif
                endif
        endfor
    endwhile
end
```

```
Function Bipartite (G) : boolean
    for each vertex s of G do
        level[s] ← -1
    endfor
    bool ←true
    for each vertex s of G do
        if level[s] = -1 then
            level[s] ← 0
            BFSbip (s)
        endif
    endfor
    return bool
```

Remarks

- Unlike BFS, in BFSbip each vertex is put exactly once in Q

→ no need to test whether the vertex *s'* has been visited before

- Running time as for BFS: $\Theta(m+n)$ with adjacency list(s).

Note the large diversity of problems solved by DFS or BFS searches.