Recognizing Graph Search Trees

Nevena Pivač

joint work with
Jesse Beisegel[1], Carolin Denkert[1], Ekkehard Köhler[1]
Matjaž Krnc[2], Robert Scheffler[1], Martin Strehler[1]

[1] University of Cottbus-Seftenberg, Cottbus, Germany
[2]University of Primorska, Koper, Slovenia

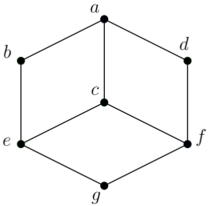
Given a graph G, we can put its vertices in some order.

Given a graph G, we can put its vertices in some order.

A rule which determines how we choose the next visited vertex defines a graph search method.

Given a graph G, we can put its vertices in some order.

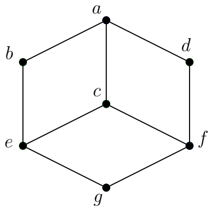
A rule which determines how we choose the next visited vertex defines a graph search method.



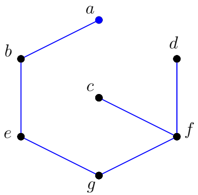
Graph G

Given a graph G, we can put its vertices in some order.

A rule which determines how we choose the next visited vertex defines a graph search method.



Graph G

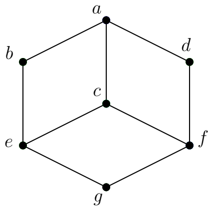


DFS

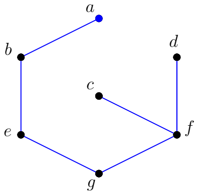
$\sigma = (a, b, e, g, f, d, c)$

Given a graph G, we can put its vertices in some order.

A rule which determines how we choose the next visited vertex defines a graph search method.

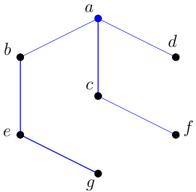


Graph G



DFS

$\sigma = (a, b, e, g, f, d, c)$

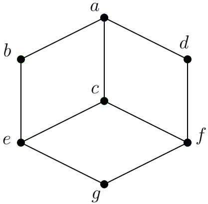


BFS

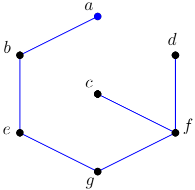
$\sigma = (a, b, c, d, e, f, g)$

Given a graph G, we can put its vertices in some order.

A rule which determines how we choose the next visited vertex defines a graph search method.

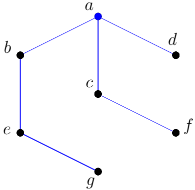


Graph G



DFS

$\sigma = (a, b, e, g, f, d, c)$



BFS

$\sigma = (a, b, c, d, e, f, g)$

Every graph search method produces some ordering of vertices σ.

Generic search

Graph search: a method of visiting all vertices in a graph that starts in a vertex and explores the graph by visiting a vertex in the neighborhood of already visited vertices.

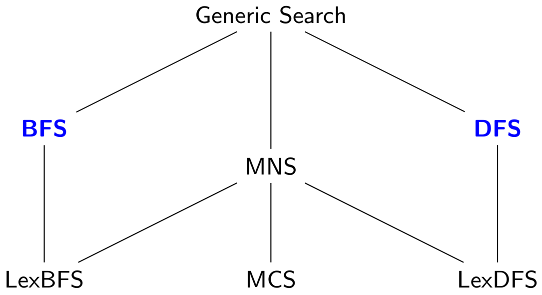
The rule of "how we choose the next visited vertex" defines the search method.

Generic search

Graph search: a method of visiting all vertices in a graph that starts in a vertex and explores the graph by visiting a vertex in the neighborhood of already visited vertices.

The rule of "how we choose the next visited vertex" defines the search method.

If no such rule exists $\rightarrow$ **generic search**.
Observe: every search method is also a generic search.



Breadth First Search

Idea of BFS:

initialize queue $Q = \{s\}$
iteratively take top vertex v from Q
add all non-queue neighbors of v to end of Q

```
Q = {s}; i=1;
foreach v ∈ Q do
    σ(i) ← v; i++;
    foreach unvisited neighbor w of v with
      w ∉ Q do
        append w to Q
    end
end
```

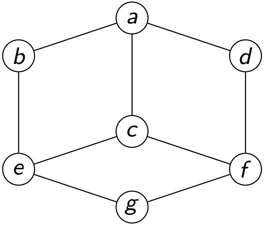
Breadth First Search

Idea of BFS:

initialize queue $Q = \{s\}$
iteratively take top vertex v from Q
add all non-queue neighbors of v to end of Q

Q = {s}; i=1;
foreach $v \in Q$ **do**
  $\sigma(i) \leftarrow v$; i++;
  foreach *unvisited neighbor* w *of* v *with* $w \notin Q$ **do**
    append w to Q
  end
end

Example



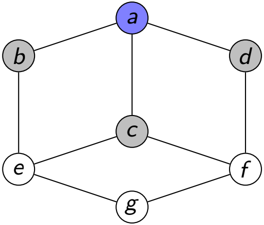
Breadth First Search

Idea of BFS:

initialize queue $Q = \{s\}$
iteratively take top vertex v from Q
add all non-queue neighbors of v to end of Q

$Q = \{s\}$; i=1;
foreach $v \in Q$ **do**
  $\sigma(i) \leftarrow v$; i++;
  foreach *unvisited neighbor* w *of* v *with* $w \notin Q$ **do**
    append w to Q
  end
end

Example



$Q = [a]\ b, d, c$

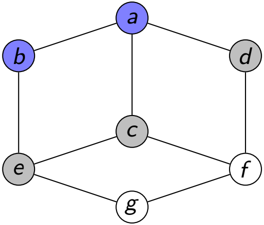
Breadth First Search

Idea of BFS:

initialize queue $Q = \{s\}$
iteratively take top vertex v from Q
add all non-queue neighbors of v to end of Q

$Q = \{s\}$; i=1;
foreach $v \in Q$ **do**
    $\sigma(i) \leftarrow v$; i++;
    foreach *unvisited neighbor* w *of* v *with* $w \notin Q$ **do**
        append w to Q
    end
end

Example



$Q = [a, b]\ d, c, e$

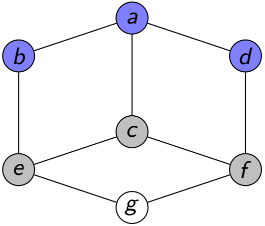
Breadth First Search

Idea of BFS:

initialize queue $Q = \{s\}$
iteratively take top vertex v from Q
add all non-queue neighbors of v to end of Q

```
Q = {s}; i=1;
foreach v ∈ Q do
    σ(i) ← v; i++;
    foreach unvisited neighbor w of v with
      w ∉ Q do
        append w to Q
    end
end
```

Example



$Q = [a, b, d]\ c, e, f$

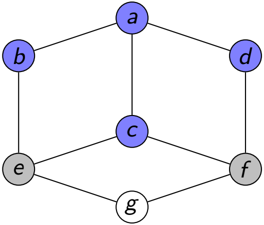
Breadth First Search

Idea of BFS:

initialize queue $Q = \{s\}$
iteratively take top vertex v from Q
add all non-queue neighbors of v to end of Q

$Q = \{s\}$; i=1;
foreach $v \in Q$ **do**
  $\sigma(i) \leftarrow v$; i++;
  foreach *unvisited neighbor* w *of* v *with* $w \notin Q$ **do**
    append w to Q
  end
end

Example



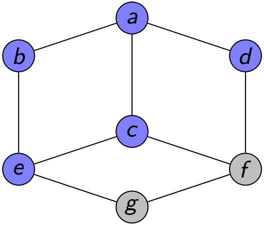
$Q = [a, b, d, c]\ e, f$

Idea of BFS:

initialize queue $Q = \{s\}$
iteratively take top vertex v from Q
add all non-queue neighbors of v to end of Q

$Q = \{s\}$; i=1;
foreach $v \in Q$ **do**
    $\sigma(i) \leftarrow v$; i++;
    foreach *unvisited neighbor* w *of* v *with* $w \notin Q$ **do**
        append w to Q
    end
end

Example



$Q = [a, b, d, c, e]\ f, g$

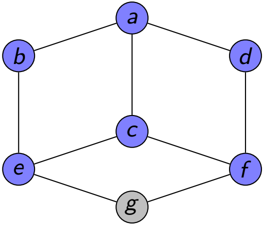
Breadth First Search

Idea of BFS:

initialize queue $Q = \{s\}$
iteratively take top vertex v from Q
add all non-queue neighbors of v to end of Q

```
Q = {s}; i=1;
foreach v ∈ Q do
    σ(i) ← v; i++;
    foreach unvisited neighbor w of v with
      w ∉ Q do
        append w to Q
    end
end
```

Example



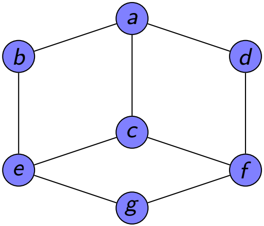
$Q = [a, b, d, c, e, f]\, g$

Idea of BFS:

initialize queue $Q = \{s\}$
iteratively take top vertex v from Q
add all non-queue neighbors of v to end of Q

$Q = \{s\}$; i=1;
foreach $v \in Q$ **do**
  $\sigma(i) \leftarrow v$; i++;
  foreach *unvisited neighbor* w *of* v *with* $w \notin Q$ **do**
    append w to Q
  end
end

Example



$Q = [a, b, d, c, e, f, g]$

Depth First Search

Idea of DFS:

initialize queue $Q = \{s\}$
among vertices in $N(s)$ visit
one that has a neighbor in Q visited
as-later-as-possible
initialize stack $S = N(s)$
iteratively take top vertex v from S
add all non-queue neighbors of v to
beginning of S

DFS(G,s)
append s to Q
foreach $v \in N(s)$ *that is not in* Q **do**
  DFS(G,v)
end

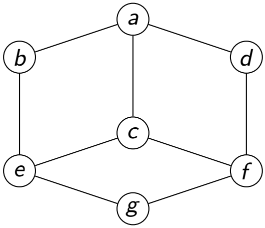
init();
$Q = []$;
DFS(G,s);

Depth First Search

Idea of DFS:

initialize queue $Q = \{s\}$
among vertices in $N(s)$ visit
one that has a neighbor in Q visited
as-later-as-possible
initialize stack $S = N(s)$
iteratively take top vertex v from S
add all non-queue neighbors of v to
beginning of S

Example



DFS(G,s)
append s to Q
foreach $v \in N(s)$ *that is not in* Q **do**
  | DFS(G,v)
end

init();
$Q = []$;
DFS(G,s);

Depth First Search

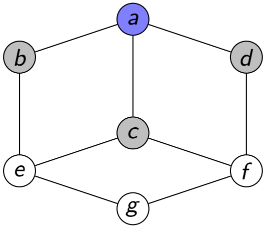
Idea of DFS:

initialize queue $Q = \{s\}$
among vertices in $N(s)$ visit
one that has a neighbor in Q visited
as-later-as-possible
initialize stack $S = N(s)$
iteratively take top vertex v from S
add all non-queue neighbors of v to
beginning of S

DFS(G,s)
append s to Q
foreach $v \in N(s)$ *that is not in* Q **do**
| DFS(G,v)
end

init();
$Q = []$;
DFS(G,s);

Example



- $Q = [a]$
- $S = [b, d, c]$

Depth First Search

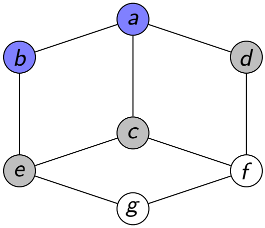
Idea of DFS:

initialize queue $Q = \{s\}$
among vertices in $N(s)$ visit
one that has a neighbor in Q visited
as-later-as-possible
initialize stack $S = N(s)$
iteratively take top vertex v from S
add all non-queue neighbors of v to
beginning of S

DFS(G,s)
append s to Q
foreach $v \in N(s)$ *that is not in* Q **do**
   DFS(G,v)
end

init();
$Q = []$;
DFS(G,s);

Example



- $Q = [a, b]$
- $S = [e, d, c]$

Depth First Search

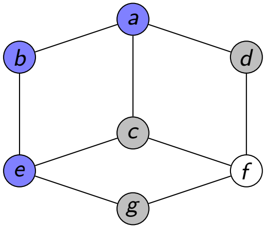
Idea of DFS:

initialize queue $Q = \{s\}$
among vertices in $N(s)$ visit
one that has a neighbor in Q visited
as-later-as-possible
initialize stack $S = N(s)$
iteratively take top vertex v from S
add all non-queue neighbors of v to
beginning of S

DFS(G,s)
append s to Q
foreach $v \in N(s)$ *that is not in* Q **do**
  DFS(G,v)
end

init();
$Q = []$;
DFS(G,s);

Example



- $Q = [a, b, e]$
- $S = [c, g, d]$

Depth First Search

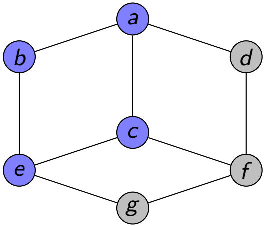
Idea of DFS:

initialize queue $Q = \{s\}$
among vertices in $N(s)$ visit
one that has a neighbor in Q visited
as-later-as-possible
initialize stack $S = N(s)$
iteratively take top vertex v from S
add all non-queue neighbors of v to
beginning of S

DFS(G,s)
append s to Q
foreach $v \in N(s)$ *that is not in* Q **do**
  DFS(G,v)
end

init();
$Q = []$;
DFS(G,s);

Example



- $Q = [a, b, e, c]$
- $S = [f, g, d]$

Depth First Search

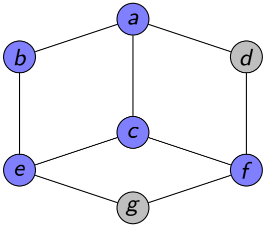
Idea of DFS:

initialize queue $Q = \{s\}$
among vertices in $N(s)$ visit
one that has a neighbor in Q visited
as-later-as-possible
initialize stack $S = N(s)$
iteratively take top vertex v from S
add all non-queue neighbors of v to
beginning of S

DFS(G,s)
append s to Q
foreach $v \in N(s)$ *that is not in* Q **do**
  DFS(G,v)
end

init();
$Q = []$;
DFS(G,s);

Example



- $Q = [a, b, e, c, f]$
- $S = [g, d]$

Depth First Search

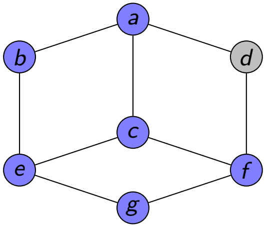
Idea of DFS:

initialize queue $Q = \{s\}$
among vertices in $N(s)$ visit
one that has a neighbor in Q visited
as-later-as-possible
initialize stack $S = N(s)$
iteratively take top vertex v from S
add all non-queue neighbors of v to
beginning of S

DFS(G,s)
append s to Q
foreach $v \in N(s)$ *that is not in* Q **do**
| DFS(G,v)
end

init();
$Q = []$;
DFS(G,s);

Example



- $Q = [a, b, e, c, f, g]$
- $S = [d]$

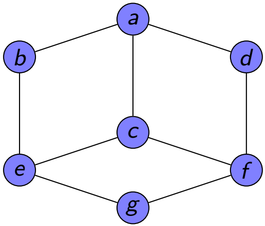
Depth First Search

Idea of DFS:

initialize queue $Q = \{s\}$
among vertices in $N(s)$ visit
one that has a neighbor in Q visited
as-later-as-possible
initialize stack $S = N(s)$
iteratively take top vertex v from S
add all non-queue neighbors of v to
beginning of S

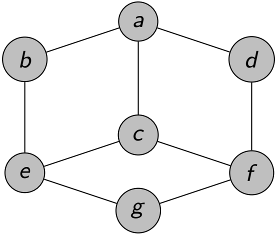
```
DFS(G,s)
append s to Q
foreach v ∈ N(s) that is not in Q do
 |  DFS(G,v)
end

init();
Q = [];
DFS(G,s);
```

Example



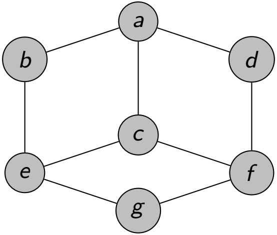
- $Q = [a, b, e, c, f, g, d]$
- $S = []$

Usually the outcome of the graph search is a *search order* = a sequence of the vertices in the order they are visited.



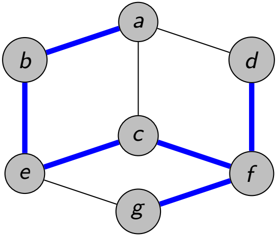
$$Q = [a, b, e, c, f, g, d]$$

Usually the outcome of the graph search is a *search order* = a sequence of the vertices in the order they are visited.



$$Q = [a, b, e, c, f, g, d]$$

Another closely related structure produced as outcome of the search: **the search tree**.



$$Q = [a, b, e, c, f, g, d]$$

We connect a vertex with some previously visited neighbor. Which one?

Why the trees are important?

- the tree obtained by a BFS contains the shortest paths from the root r to all other vertices in the graphs
- the trees generated by DFS can be used for fast planarity testing of graphs
- using trees we can fing Hamiltonian path in a co-comparability graph
- multiple runs of graph searches give a strong insight into graph structure

Why the trees are important?

- the tree obtained by a BFS contains the shortest paths from the root r to all other vertices in the graphs
- the trees generated by DFS can be used for fast planarity testing of graphs
- using trees we can fing Hamiltonian path in a co-comparability graph
- multiple runs of graph searches give a strong insight into graph structure

There seems to be some hidden structural properties.

Why the trees are important?

- the tree obtained by a BFS contains the shortest paths from the root r to all other vertices in the graphs
- the trees generated by DFS can be used for fast planarity testing of graphs
- using trees we can fing Hamiltonian path in a co-comparability graph
- multiple runs of graph searches give a strong insight into graph structure

There seems to be some hidden structural properties.

Question: whether a given tree can be a search tree of a particular search?

How the search tree is defined?

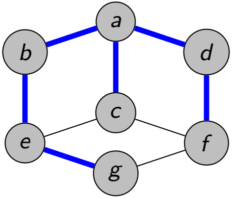
When visiting vertex v, we connect it with one already visited neighbor of v.

BFS: we connect it with neighbor of v that appeared **first** in the BFS order.
DFS: we connect it with neighbor of v that was visited **last** before v.

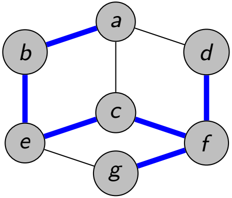
How the search tree is defined?

When visiting vertex v, we connect it with one already visited neighbor of v.

BFS: we connect it with neighbor of v that appeared **first** in the BFS order.
DFS: we connect it with neighbor of v that was visited **last** before v.

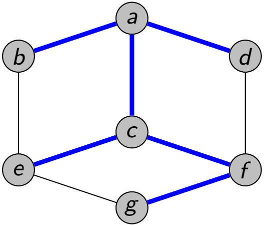


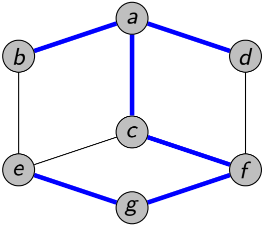
$Q = [a, b, d, c, e, f, g]$



$Q = [a, b, e, c, f, g, d]$

Is T an BFS tree of G?





$\mathcal{F}$ and $\mathcal{L}$ search trees

For BFS and DFS it is clear how to connect a vertex with some previously visited neighbor. In general, it is not clear, so we have a freedom to choose

- first visited neighbor
- last visited neighbor
- any other visited neighbor

$\mathcal{F}$ and $\mathcal{L}$ search trees

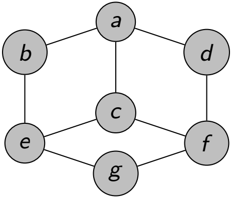
For BFS and DFS it is clear how to connect a vertex with some previously visited neighbor. In general, it is not clear, so we have a freedom to choose

- first visited neighbor
- last visited neighbor
- any other visited neighbor

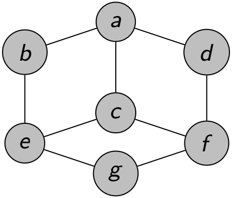
Given a search method on the graph G that produces an ordering σ, we consider two types of search trees

- $\mathcal{F}$-search tree: we construct a search tree so that we add edges connecting a current vertex with a **first** visited vertex.
- $\mathcal{L}$-search tree: we construct a search tree so that we add edges connecting a current vertex with a **last** visited vertex.

Assume we have a graph G and an ordering of vertices $\sigma = [a, c, b, e, f, d, g]$.

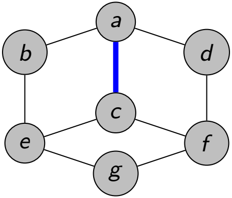


type $\mathcal{F}$

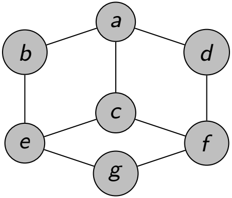


type $\mathcal{L}$

Assume we have a graph G and an ordering of vertices $\sigma = [a, c, b, e, f, d, g]$.



type $\mathcal{F}$

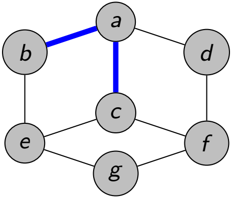


type $\mathcal{L}$

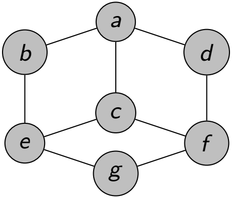
$\mathcal{F}$-Tree ($\mathcal{L}$-Tree) Recognition Problem

Instance: A connected graph $G = (V, E)$ and a spanning tree T.
Task: Decide whether there is a graph search of the given type such that T is its $\mathcal{F}$-tree ($\mathcal{L}$-tree) of G.

Assume we have a graph G and an ordering of vertices $\sigma = [a, c, b, e, f, d, g]$.



type $\mathcal{F}$

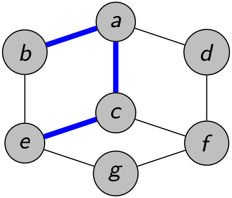


type $\mathcal{L}$

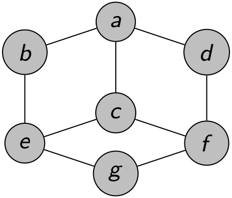
$\mathcal{F}$-Tree ($\mathcal{L}$-Tree) Recognition Problem

Instance: A connected graph $G = (V, E)$ and a spanning tree T.
Task: Decide whether there is a graph search of the given type such that T is its $\mathcal{F}$-tree ($\mathcal{L}$-tree) of G.

Assume we have a graph G and an ordering of vertices $\sigma = [a, c, b, e, f, d, g]$.



type $\mathcal{F}$

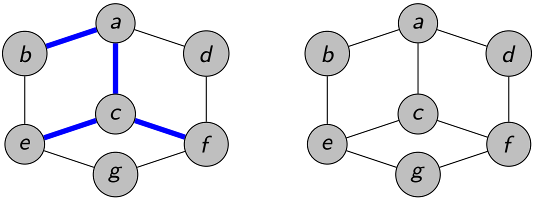


type $\mathcal{L}$

$\mathcal{F}$-Tree ($\mathcal{L}$-Tree) Recognition Problem

Instance: A connected graph $G = (V, E)$ and a spanning tree T.
Task: Decide whether there is a graph search of the given type such that T is its $\mathcal{F}$-tree ($\mathcal{L}$-tree) of G.

Assume we have a graph G and an ordering of vertices $\sigma = [a, c, b, e, f, d, g]$.

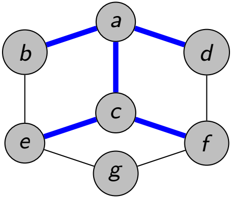


type $\mathcal{F}$ type $\mathcal{L}$

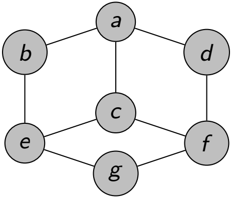
$\mathcal{F}$-Tree ($\mathcal{L}$-Tree) Recognition Problem

Instance: A connected graph $G = (V, E)$ and a spanning tree T.
Task: Decide whether there is a graph search of the given type such that T is its $\mathcal{F}$-tree ($\mathcal{L}$-tree) of G.

Assume we have a graph G and an ordering of vertices $\sigma = [a, c, b, e, f, d, g]$.



type $\mathcal{F}$

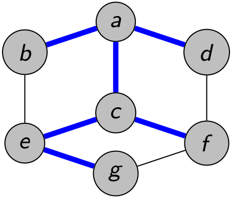


type $\mathcal{L}$

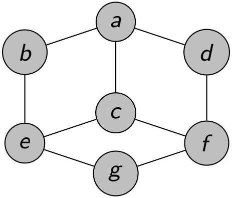
$\mathcal{F}$-Tree ($\mathcal{L}$-Tree) Recognition Problem

Instance: A connected graph $G = (V, E)$ and a spanning tree T.
Task: Decide whether there is a graph search of the given type such that T is its $\mathcal{F}$-tree ($\mathcal{L}$-tree) of G.

Assume we have a graph G and an ordering of vertices $\sigma = [a, c, b, e, f, d, g]$.



type $\mathcal{F}$

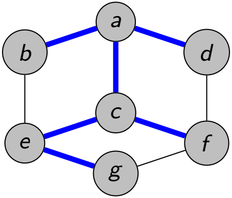


type $\mathcal{L}$

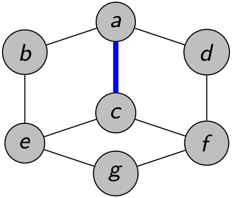
$\mathcal{F}$-Tree ($\mathcal{L}$-Tree) Recognition Problem

Instance: A connected graph $G = (V, E)$ and a spanning tree T.
Task: Decide whether there is a graph search of the given type such that T is its $\mathcal{F}$-tree ($\mathcal{L}$-tree) of G.

Assume we have a graph G and an ordering of vertices $\sigma = [a, c, b, e, f, d, g]$.



type $\mathcal{F}$

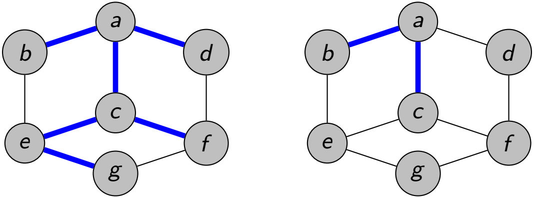


type $\mathcal{L}$

$\mathcal{F}$-Tree ($\mathcal{L}$-Tree) Recognition Problem

Instance: A connected graph $G = (V, E)$ and a spanning tree T.
Task: Decide whether there is a graph search of the given type such that T is its $\mathcal{F}$-tree ($\mathcal{L}$-tree) of G.

Assume we have a graph G and an ordering of vertices $\sigma = [a, c, b, e, f, d, g]$.

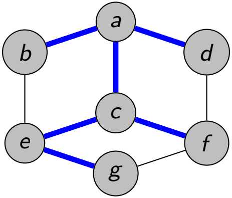


type $\mathcal{F}$ type $\mathcal{L}$

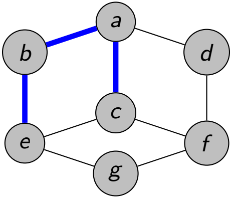
$\mathcal{F}$-Tree ($\mathcal{L}$-Tree) Recognition Problem

Instance: A connected graph $G = (V, E)$ and a spanning tree T.
Task: Decide whether there is a graph search of the given type such that T is its $\mathcal{F}$-tree ($\mathcal{L}$-tree) of G.

Assume we have a graph G and an ordering of vertices $\sigma = [a, c, b, e, f, d, g]$.



type $\mathcal{F}$

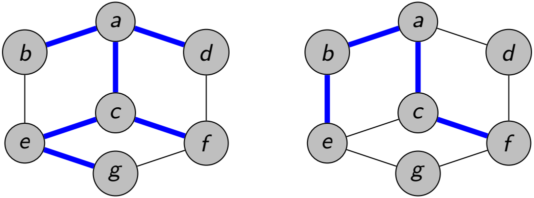


type $\mathcal{L}$

$\mathcal{F}$-Tree ($\mathcal{L}$-Tree) Recognition Problem

Instance: A connected graph $G = (V, E)$ and a spanning tree T.
Task: Decide whether there is a graph search of the given type such that T is its $\mathcal{F}$-tree ($\mathcal{L}$-tree) of G.

Assume we have a graph G and an ordering of vertices $\sigma = [a, c, b, e, f, d, g]$.

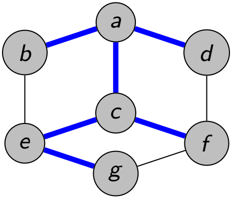


type $\mathcal{F}$ type $\mathcal{L}$

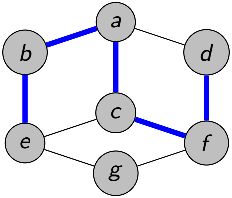
$\mathcal{F}$-Tree ($\mathcal{L}$-Tree) Recognition Problem

Instance: A connected graph $G = (V, E)$ and a spanning tree T.
Task: Decide whether there is a graph search of the given type such that T is its $\mathcal{F}$-tree ($\mathcal{L}$-tree) of G.

Assume we have a graph G and an ordering of vertices $\sigma = [a, c, b, e, f, d, g]$.



type $\mathcal{F}$

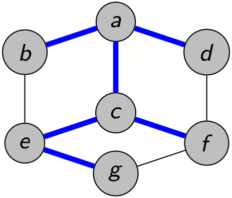


type $\mathcal{L}$

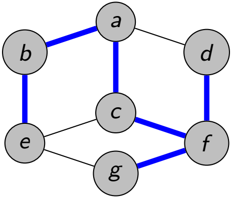
$\mathcal{F}$-Tree ($\mathcal{L}$-Tree) Recognition Problem

Instance: A connected graph $G = (V, E)$ and a spanning tree T.
Task: Decide whether there is a graph search of the given type such that T is its $\mathcal{F}$-tree ($\mathcal{L}$-tree) of G.

Assume we have a graph G and an ordering of vertices $\sigma = [a, c, b, e, f, d, g]$.



type $\mathcal{F}$



type $\mathcal{L}$

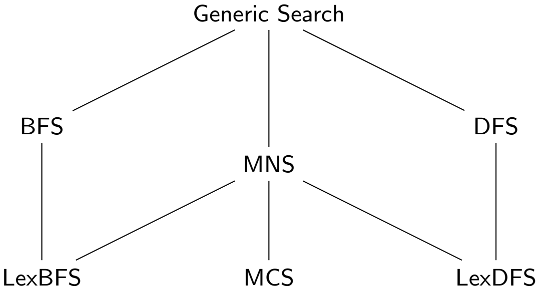
$\mathcal{F}$-Tree ($\mathcal{L}$-Tree) Recognition Problem

Instance: A connected graph $G = (V, E)$ and a spanning tree T.
Task: Decide whether there is a graph search of the given type such that T is its $\mathcal{F}$-tree ($\mathcal{L}$-tree) of G.

The problem is defined on graphs. So lets talk about graphs.

- A tree is a graph without induced cycles.
- The first point of a tree is its root.
- Given a graph $G = (V, E)$ and a tree T, we say that T is a spanning tree of G if $V(T) = V$ and $E(T) \subseteq E$.
- A graph G is a split graph: vertices of G can be partitioned into sets C and I, with C being a clique and I an independent set.
- A graph G isweakly chordal: G and $\overline{G}$ have no induced cycles of length greater than 4.

On the search for the right search



Lexicographic BFS

Idea of LexBFS:

iteratively select vertex with lexicogr. largest label;
selected vertex appends number i to label of neighbors

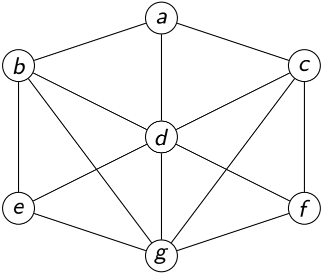
foreach $v \in V$ **do** label(v) $= \emptyset$;
label(s) $= \{0\}$; $n = |V|$
for $i \leftarrow n$ **to** 1 **do**
    $v \leftarrow$ unnumbered vertex with lexic. largest label $l(v)$;
    $\sigma(n-i) \leftarrow v$;
    foreach *unnumb. neighbor w of v* **do**
        append i to $l(w)$
    end
end

Idea of LBFS:

```
foreach v ∈ V do label(v) = ∅;
label(s) = {0}; n = |V|
for i ← n to 1 do
    v ← unnumbered vertex with
      lexicographically largest label l(v);
    σ(n − i) ← v;
    foreach unnumbered neighbor w of v do
        append i to l(w)
    end
end
```

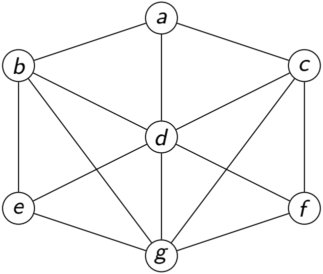
Idea of LBFS:

foreach $v \in V$ **do** label(v) $= \emptyset$;
label(s) $= \{0\}$; $n = |V|$
for $i \leftarrow n$ **to** 1 **do**
  $v \leftarrow$ unnumbered vertex with lexicographically largest label $l(v)$;
  $\sigma(n-i) \leftarrow v$;
  foreach *unnumbered neighbor w of v* **do**
    append i to $l(w)$
  end
end



Idea of LBFS:

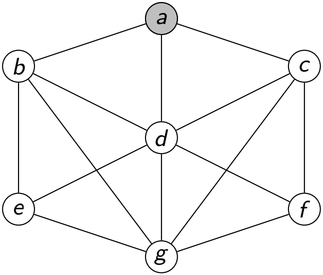
foreach $v \in V$ **do** label(v) $= \emptyset$;
label(s) $= \{0\}$; $n = |V|$
for $i \leftarrow n$ **to** 1 **do**
  $v \leftarrow$ unnumbered vertex with lexicographically largest label $l(v)$;
  $\sigma(n - i) \leftarrow v$;
  foreach *unnumbered neighbor* w *of* v **do**
    append i to $l(w)$
  end
end



a	
b	
c	
d	
e	
f	
g	

Idea of LBFS:

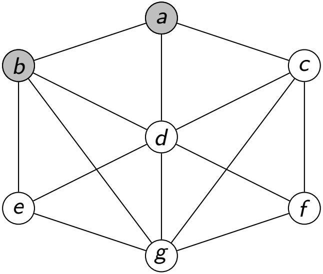
foreach $v \in V$ **do** label(v) $= \emptyset$;
label(s) $= \{0\}$; $n = |V|$
for $i \leftarrow n$ **to** 1 **do**
    $v \leftarrow$ unnumbered vertex with
     lexicographically largest label $l(v)$;
    $\sigma(n-i) \leftarrow v$;
    foreach *unnumbered neighbor w of v* **do**
        append i to $l(w)$
    end
end



a	7
b	7
c	7
d	7
e	
f	
g	

Idea of LBFS:

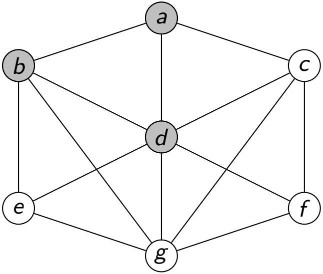
foreach $v \in V$ **do** label(v) $= \emptyset$;
label(s) $= \{0\}$; $n = |V|$
for $i \leftarrow n$ **to** 1 **do**
    $v \leftarrow$ unnumbered vertex with lexicographically largest label $l(v)$;
    $\sigma(n-i) \leftarrow v$;
    foreach *unnumbered neighbor* w *of* v **do**
        append i to $l(w)$
    end
end



a	7	7
b	7	6
c	7	7
d	7	76
e		6
f		
g		6

Idea of LBFS:

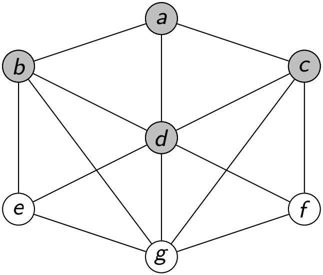
foreach $v \in V$ **do** label(v) $= \emptyset$;
label(s) $= \{0\}$; $n = |V|$
for $i \leftarrow n$ **to** 1 **do**
    $v \leftarrow$ unnumbered vertex with
     lexicographically largest label $l(v)$;
    $\sigma(n-i) \leftarrow v$;
    foreach *unnumbered neighbor w of v* **do**
        append i to $l(w)$
    end
end



a	7	7	7
b	7	6	6
c	7	7	75
d	7	76	5
e		6	65
f			5
g		6	65

Idea of LBFS:

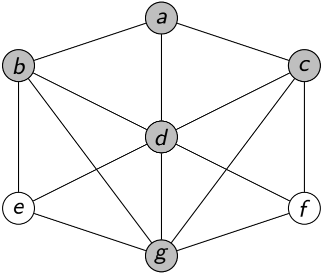
```
foreach v ∈ V do label(v) = ∅;
label(s) = {0}; n = |V|
for i ← n to 1 do
    v ← unnumbered vertex with
      lexicographically largest label l(v);
    σ(n − i) ← v;
    foreach unnumbered neighbor w of v do
        append i to l(w)
    end
end
```



a	7	7	7	7
b	7	6	6	6
c	7	7	75	4
d	7	76	5	5
e		6	65	65
f			5	54
g		6	65	654

Idea of LBFS:

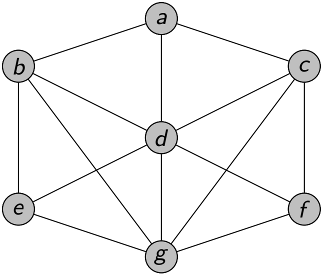
foreach $v \in V$ **do** label(v) $= \emptyset$;
label(s) $= \{0\}$; $n = |V|$
for $i \leftarrow n$ **to** 1 **do**
  $v \leftarrow$ unnumbered vertex with lexicographically largest label $l(v)$;
  $\sigma(n-i) \leftarrow v$;
  foreach *unnumbered neighbor w of v* **do**
    append i to $l(w)$
  end
end



a	7	7	7	7	7
b	7	6	6	6	6
c	7	7	75	4	4
d	7	76	5	5	5
e		6	65	65	653
f			5	54	543
g		6	65	654	3

Idea of LBFS:

foreach $v \in V$ **do** label(v) $= \emptyset$;
label(s) $= \{0\}$; $n = |V|$
for $i \leftarrow n$ **to** 1 **do**
  $v \leftarrow$ unnumbered vertex with
   lexicographically largest label $l(v)$;
  $\sigma(n-i) \leftarrow v$;
  foreach *unnumbered neighbor* w *of* v **do**
   append i to $l(w)$
  end
end

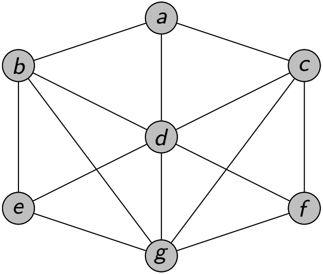


a	7	7	7	7	7	7
b	7	6	6	6	6	6
c	7	7	75	4	4	4
d	7	76	5	5	5	5
e		6	65	65	653	2
f			5	54	543	1
g		6	65	654	3	3

Idea of LBFS:

foreach $v \in V$ **do** label(v) $= \emptyset$;
label(s) $= \{0\}$; $n = |V|$
for $i \leftarrow n$ **to** 1 **do**
  $v \leftarrow$ unnumbered vertex with
  lexicographically largest label $l(v)$;
  $\sigma(n-i) \leftarrow v$;
  foreach *unnumbered neighbor* w *of* v **do**
    append i to $l(w)$
  end
end

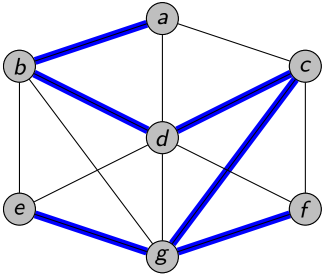
$\sigma = (a, b, d, c, g, e, f)$



a	7	7	7	7	7	7
b	7	6	6	6	6	6
c	7	7	75	4	4	4
d	7	76	5	5	5	5
e		6	65	65	653	2
f			5	54	543	1
g		6	65	654	3	3

Idea of LBFS:

foreach $v \in V$ **do** label(v) $= \emptyset$;
label(s) $= \{0\}$; $n = |V|$
for $i \leftarrow n$ **to** 1 **do**
    $v \leftarrow$ unnumbered vertex with
     lexicographically largest label $l(v)$;
    $\sigma(n - i) \leftarrow v$;
    foreach *unnumbered neighbor* w *of* v **do**
        append i to $l(w)$
    end
end

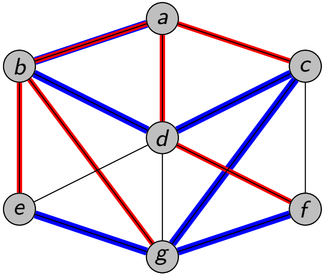


$\sigma = (a, b, d, c, g, e, f)$
$\mathcal{F}$-tree
$\mathcal{L}$-tree

a	7	7	7	7	7	7
b	7	6	6	6	6	6
c	7	7	75	4	4	4
d	7	76	5	5	5	5
e		6	65	65	653	2
f			5	54	543	1
g		6	65	654	3	3

Idea of LBFS:

foreach $v \in V$ **do** label(v) $= \emptyset$;
label(s) $= \{0\}$; $n = |V|$
for $i \leftarrow n$ **to** 1 **do**
    $v \leftarrow$ unnumbered vertex with lexicographically largest label $l(v)$;
    $\sigma(n-i) \leftarrow v$;
    foreach *unnumbered neighbor w of v* **do**
        append i to $l(w)$
    end
end



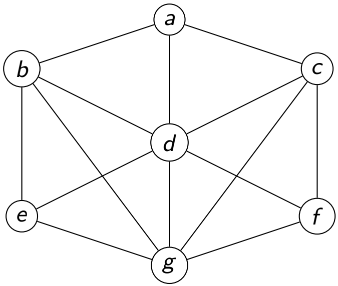
$\sigma = (a, b, d, c, g, e, f)$
$\mathcal{F}$-tree
$\mathcal{L}$-tree

a	7	7	7	7	7	7
b	7	6	6	6	6	6
c	7	7	75	4	4	4
d	7	76	5	5	5	5
e		6	65	65	653	2
f			5	54	543	1
g		6	65	654	3	3

Lexicographic DFS

Idea of LDFS:

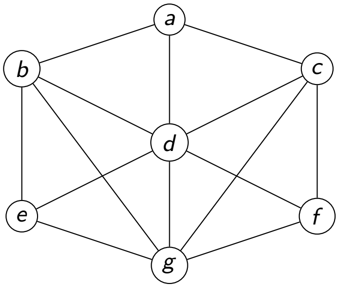
foreach $v \in V$ **do** label(v) $= \emptyset$;
label(s) $= \{0\}$; $n = |V|$
for $i \leftarrow 1$ **to** n **do**
  $v \leftarrow$ unnumbered vertex with lexicographically largest label $l(v)$;
  $\sigma(i) \leftarrow v$;
  foreach *unnumbered neighbor* w *of* v **do**
    prepend i to $l(w)$
  end
end



Lexicographic DFS

Idea of LDFS:

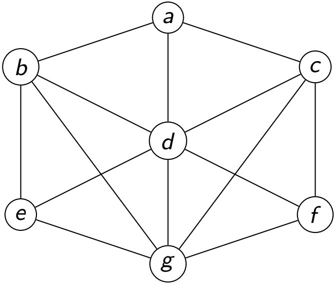
foreach $v \in V$ **do** label(v) $= \emptyset$;
label(s) $= \{0\}$; $n = |V|$
for $i \leftarrow 1$ **to** n **do**
    $v \leftarrow$ unnumbered vertex with lexicographically largest label $l(v)$;
    $\sigma(i) \leftarrow v$;
    foreach *unnumbered neighbor w of v* **do**
        prepend i to $l(w)$
    end
end



Lexicographic DFS

Idea of LDFS:

foreach $v \in V$ **do** label(v) $= \emptyset$;
label(s) $= \{0\}$; $n = |V|$
for $i \leftarrow 1$ **to** n **do**
    $v \leftarrow$ unnumbered vertex with lexicographically largest label $l(v)$;
    $\sigma(i) \leftarrow v$;
    foreach *unnumbered neighbor* w *of* v **do**
        prepend i to $l(w)$
    end
end

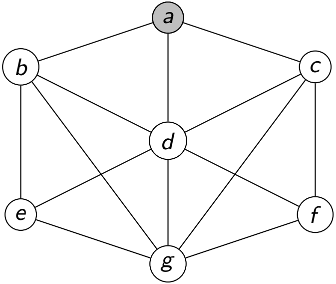


a
b
c
d
e
f
g

Lexicographic DFS

Idea of LDFS:

foreach $v \in V$ **do** label(v) $= \emptyset$;
label(s) $= \{0\}$; $n = |V|$
for $i \leftarrow 1$ **to** n **do**
    $v \leftarrow$ unnumbered vertex with lexicographically largest label $l(v)$;
    $\sigma(i) \leftarrow v$;
    foreach *unnumbered neighbor* w *of* v **do**
        prepend i to $l(w)$
    end
end

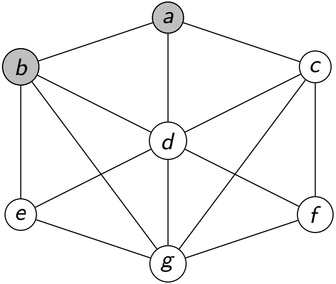


a	1
b	1
c	1
d	1
e	
f	
g	

Lexicographic DFS

Idea of LDFS:

foreach $v \in V$ **do** label(v) $= \emptyset$;
label(s) $= \{0\}$; $n = |V|$
for $i \leftarrow 1$ **to** n **do**
  $v \leftarrow$ unnumbered vertex with lexicographically largest label $l(v)$;
  $\sigma(i) \leftarrow v$;
  foreach *unnumbered neighbor w of v* **do**
    prepend i to $l(w)$
  end
end

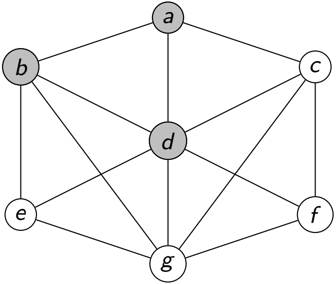


a	1	1
b	1	2
c	1	1
d	1	21
e		2
f		
g		2

Lexicographic DFS

Idea of LDFS:

foreach $v \in V$ **do** label(v) $= \emptyset$;
label(s) $= \{0\}$; $n = |V|$
for $i \leftarrow 1$ **to** n **do**
  $v \leftarrow$ unnumbered vertex with lexicographically largest label $l(v)$;
  $\sigma(i) \leftarrow v$;
  foreach *unnumbered neighbor* w *of* v **do**
    prepend i to $l(w)$
  end
end

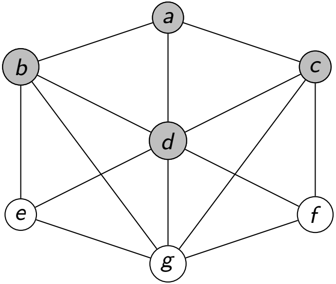


a	1	1	1
b	1	2	2
c	1	1	321
d	1	21	3
e		2	32
f			3
g		2	3

Lexicographic DFS

Idea of LDFS:

foreach $v \in V$ **do** label(v) $= \emptyset$;
label(s) $= \{0\}$; $n = |V|$
for $i \leftarrow 1$ **to** n **do**
  $v \leftarrow$ unnumbered vertex with lexicographically largest label $l(v)$;
  $\sigma(i) \leftarrow v$;
  foreach *unnumbered neighbor* w *of* v **do**
    prepend i to $l(w)$
  end
end

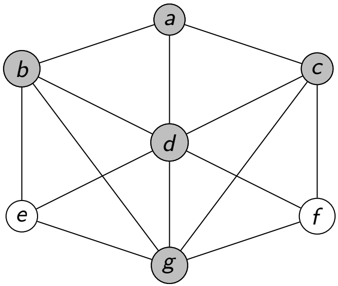


a	1	1	1	1
b	1	2	2	2
c	1	1	321	4
d	1	21	3	3
e		2	32	32
f			3	43
g		2	3	432

Lexicographic DFS

Idea of LDFS:

foreach $v \in V$ **do** label(v) $= \emptyset$;
label(s) $= \{0\}$; $n = |V|$
for $i \leftarrow 1$ **to** n **do**
  $v \leftarrow$ unnumbered vertex with lexicographically largest label $l(v)$;
  $\sigma(i) \leftarrow v$;
  foreach *unnumbered neighbor* w *of* v **do**
    prepend i to $l(w)$
  end
end

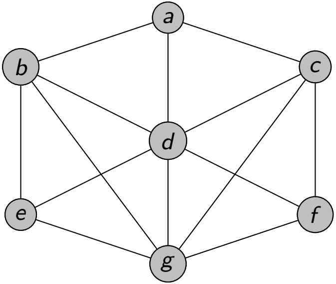


a	1	1	1	1	1
b	1	2	2	2	2
c	1	1	321	4	4
d	1	21	3	3	3
e		2	32	32	32
f			3	43	43
g		2	3	432	5

Lexicographic DFS

Idea of LDFS:

foreach $v \in V$ **do** label(v) $= \emptyset$;
label(s) $= \{0\}$; $n = |V|$
for $i \leftarrow 1$ **to** n **do**
  $v \leftarrow$ unnumbered vertex with lexicographically largest label $l(v)$;
  $\sigma(i) \leftarrow v$;
  foreach *unnumbered neighbor w of v* **do**
    prepend i to $l(w)$
  end
end



a	1	1	1	1	1	7
b	1	2	2	2	2	6
c	1	1	321	4	4	4
d	1	21	3	3	3	5
e		2	32	32	32	2
f			3	43	43	1
g		2	3	432	5	3

Input: a graph $G = (V, E)$ and a distinguished vertex $v \in V$.
Result: an MNS ordering of G.

Maximal Neighborhood Search - MNS

Input: a graph $G = (V, E)$ and a distinguished vertex $v \in V$.
Result: an MNS ordering of G.

- for each vertex we will have some labels and numbers,

Maximal Neighborhood Search - MNS

Input: a graph $G = (V, E)$ and a distinguished vertex $v \in V$.
Result: an MNS ordering of G.

- for each vertex we will have some labels and numbers,
- we start with vertex v, assign a number 1 to it,

Maximal Neighborhood Search - MNS

Input: a graph $G = (V, E)$ and a distinguished vertex $v \in V$.
Result: an MNS ordering of G.

- for each vertex we will have some labels and numbers,
- we start with vertex v, assign a number 1 to it,
- we add 1 to the label set of each neighbor of v,

Maximal Neighborhood Search - MNS

Input: a graph $G = (V, E)$ and a distinguished vertex $v \in V$.
Result: an MNS ordering of G.

- for each vertex we will have some labels and numbers,
- we start with vertex v, assign a number 1 to it,
- we add 1 to the label set of each neighbor of v,
- we choose the unnumbered vertex in V with the largest label under set inclusion, assign a number 2 to it,

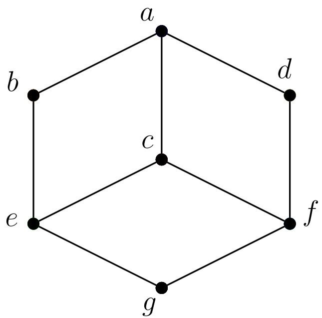
Maximal Neighborhood Search - MNS

Input: a graph $G = (V, E)$ and a distinguished vertex $v \in V$.
Result: an MNS ordering of G.

- for each vertex we will have some labels and numbers,
- we start with vertex v, assign a number 1 to it,
- we add 1 to the label set of each neighbor of v,
- we choose the unnumbered vertex in V with the largest label under set inclusion, assign a number 2 to it,
- continue until all vertices are numbered.

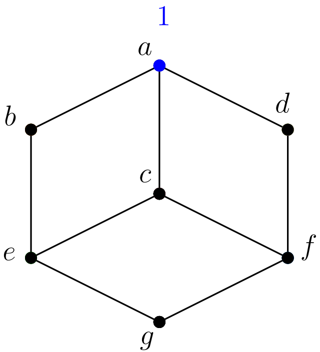
Example of MNS

assign the label $\emptyset$ to all vertices;
label(s) $\leftarrow \{n+1\}$;
foreach $i \leftarrow 1$ *to* n **do**
  pick an unnumbered vertex v with maximal label under set inclusion;
  $\sigma(i) \leftarrow v$;
  foreach *unnumbered vertex* $w \in N(v)$ **do**
    add i to label(w);
  end
end



Example of MNS

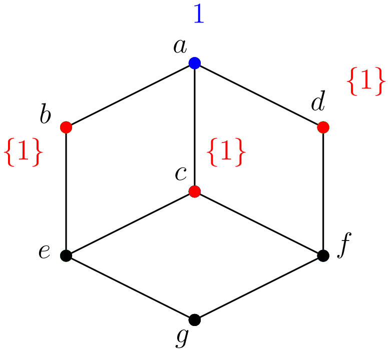
assign the label $\emptyset$ to all vertices;
label(s) $\leftarrow \{n+1\}$;
foreach $i \leftarrow 1$ *to* n **do**
  pick an unnumbered vertex v with maximal label under set inclusion;
  $\sigma(i) \leftarrow v$;
  foreach *unnumbered vertex* $w \in N(v)$ **do**
    add i to label(w);
  end
end



$\sigma = (a)$

Example of MNS

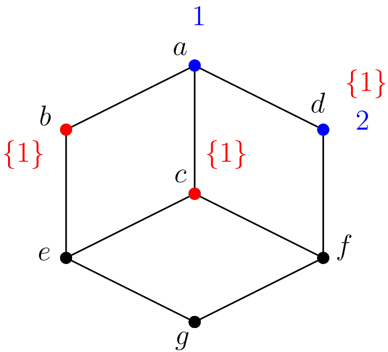
assign the label $\emptyset$ to all vertices;
label(s) $\leftarrow \{n+1\}$;
foreach $i \leftarrow 1$ *to* n **do**
  pick an unnumbered vertex v with maximal label under set inclusion;
  $\sigma(i) \leftarrow v$;
  foreach *unnumbered vertex* $w \in N(v)$ **do**
    add i to label(w);
  end
end



$\sigma = (a)$

Example of MNS

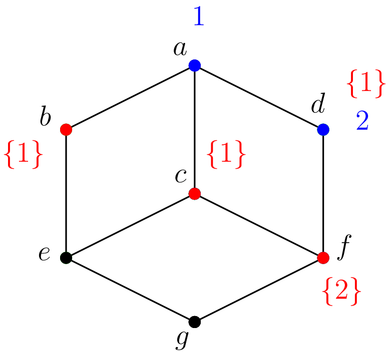
assign the label $\emptyset$ to all vertices;
label(s) $\leftarrow \{n+1\}$;
foreach $i \leftarrow 1$ *to* n **do**
    pick an unnumbered vertex v with maximal label under set inclusion;
    $\sigma(i) \leftarrow v$;
    foreach *unnumbered vertex* $w \in N(v)$ **do**
        add i to label(w);
    end
end



$$\sigma = (a, d)$$

Example of MNS

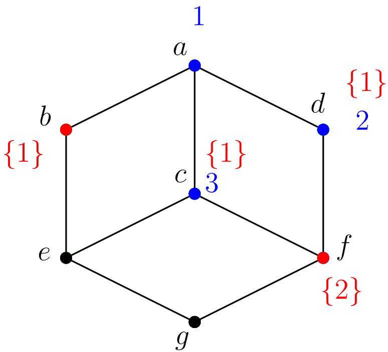
assign the label $\emptyset$ to all vertices;
label(s) $\leftarrow \{n+1\}$;
foreach $i \leftarrow 1$ *to* n **do**
    pick an unnumbered vertex v with maximal label under set inclusion;
    $\sigma(i) \leftarrow v$;
    foreach *unnumbered vertex* $w \in N(v)$ **do**
        add i to label(w);
    end
end



$$\sigma = (a, d)$$

Example of MNS

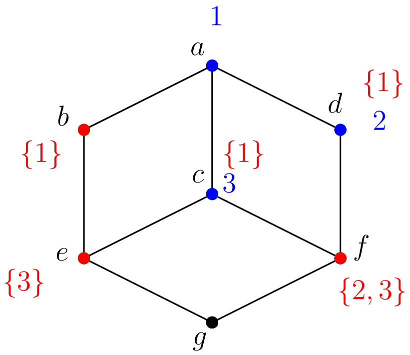
assign the label $\emptyset$ to all vertices;
label(s) $\leftarrow \{n+1\}$;
foreach $i \leftarrow 1$ *to* n **do**
  pick an unnumbered vertex v with maximal label under set inclusion;
  $\sigma(i) \leftarrow v$;
  foreach *unnumbered vertex* $w \in N(v)$ **do**
    add i to label(w);
  end
end



$$\sigma = (a, d, c)$$

Example of MNS

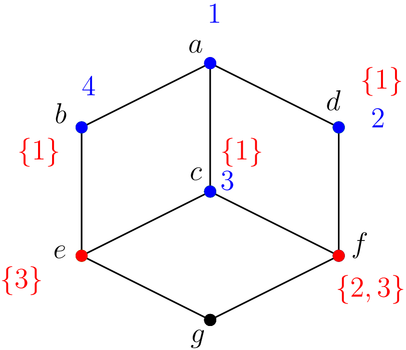
assign the label $\emptyset$ to all vertices;
label(s) $\leftarrow \{n+1\}$;
foreach $i \leftarrow 1$ *to* n **do**
  pick an unnumbered vertex v with maximal label under set inclusion;
  $\sigma(i) \leftarrow v$;
  foreach *unnumbered vertex* $w \in N(v)$ **do**
    add i to label(w);
  end
end



$$\sigma = (a, d, c)$$

Example of MNS

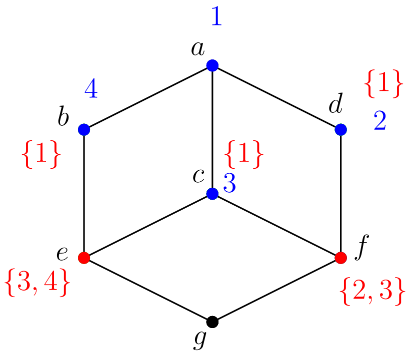
assign the label $\emptyset$ to all vertices;
label(s) $\leftarrow \{n+1\}$;
foreach $i \leftarrow 1$ *to* n **do**
    pick an unnumbered vertex v with maximal label under set inclusion;
    $\sigma(i) \leftarrow v$;
    foreach *unnumbered vertex* $w \in N(v)$ **do**
        add i to label(w);
    end
end



$$\sigma = (a, d, c, b)$$

Example of MNS

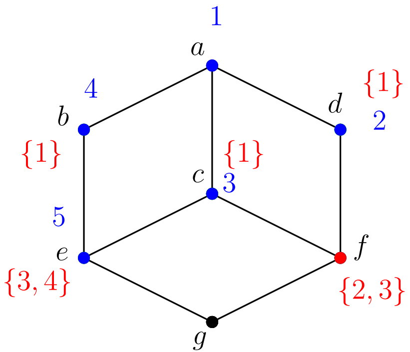
assign the label $\emptyset$ to all vertices;
label(s) $\leftarrow \{n+1\}$;
foreach $i \leftarrow 1$ *to* n **do**
    pick an unnumbered vertex v with maximal label under set inclusion;
    $\sigma(i) \leftarrow v$;
    foreach *unnumbered vertex* $w \in N(v)$ **do**
        add i to label(w);
    end
end



$$\sigma = (a, d, c, b)$$

Example of MNS

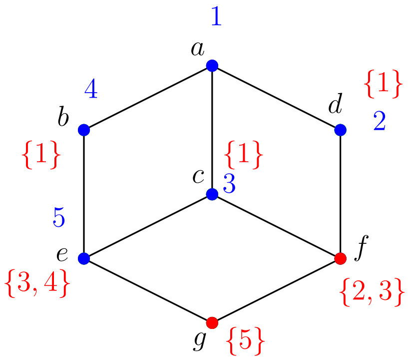
assign the label $\emptyset$ to all vertices;
label(s) $\leftarrow \{n+1\}$;
foreach $i \leftarrow 1$ *to* n **do**
  pick an unnumbered vertex v with maximal label under set inclusion;
  $\sigma(i) \leftarrow v$;
  foreach *unnumbered vertex* $w \in N(v)$ **do**
    add i to label(w);
  end
end



$\sigma = (a, d, c, b, e)$

Example of MNS

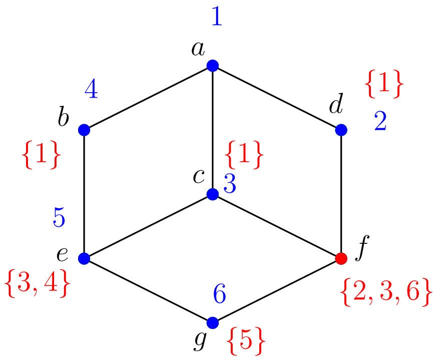
assign the label $\emptyset$ to all vertices;
label(s) $\leftarrow \{n+1\}$;
foreach $i \leftarrow 1$ *to* n **do**
    pick an unnumbered vertex v with maximal label under set inclusion;
    $\sigma(i) \leftarrow v$;
    foreach *unnumbered vertex* $w \in N(v)$ **do**
        add i to label(w);
    end
end



$\sigma = (a, d, c, b, e)$

Example of MNS

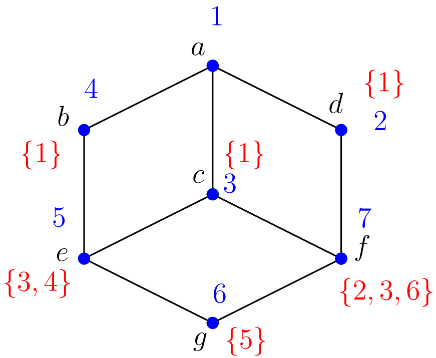
assign the label $\emptyset$ to all vertices;
label(s) $\leftarrow \{n+1\}$;
foreach $i \leftarrow 1$ *to* n **do**
    pick an unnumbered vertex v with maximal label under set inclusion;
    $\sigma(i) \leftarrow v$;
    foreach *unnumbered vertex* $w \in N(v)$ **do**
        add i to label(w);
    end
end



$\sigma = (a, d, c, b, e, g)$

Example of MNS

assign the label $\emptyset$ to all vertices;
label(s) $\leftarrow \{n+1\}$;
foreach $i \leftarrow 1$ *to* n **do**
    pick an unnumbered vertex v with maximal label under set inclusion;
    $\sigma(i) \leftarrow v$;
    foreach *unnumbered vertex* $w \in N(v)$ **do**
        add i to label(w);
    end
end



$\sigma = (a, d, c, b, e, g, f)$

Maximum Cardinality Search - MCS

Input: a graph $G = (V, E)$ and a distinguished vertex $v \in V$.
Result: an MCS ordering of G.

Maximum Cardinality Search - MCS

Input: a graph $G = (V, E)$ and a distinguished vertex $v \in V$.
Result: an MCS ordering of G.

- for each vertex we will have some labels and numbers,

Input: a graph $G = (V, E)$ and a distinguished vertex $v \in V$.
Result: an MCS ordering of G.

- for each vertex we will have some labels and numbers,
- we start with vertex v, assign a number 1 to it,

Maximum Cardinality Search - MCS

Input: a graph $G = (V, E)$ and a distinguished vertex $v \in V$.
Result: an MCS ordering of G.

- for each vertex we will have some labels and numbers,
- we start with vertex v, assign a number 1 to it,
- we add 1 to the label set of each neighbor of v,

Maximum Cardinality Search - MCS

Input: a graph $G = (V, E)$ and a distinguished vertex $v \in V$.
Result: an MCS ordering of G.

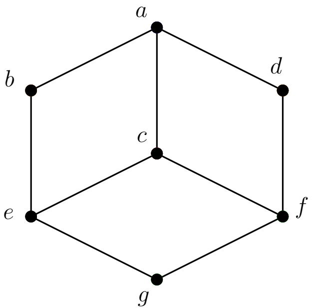
- for each vertex we will have some labels and numbers,
- we start with vertex v, assign a number 1 to it,
- we add 1 to the label set of each neighbor of v,
- we choose the unnumbered vertex in V with the largest label under set cardinality, assign a number 2 to it,

Input: a graph $G = (V, E)$ and a distinguished vertex $v \in V$.
Result: an MCS ordering of G.

- for each vertex we will have some labels and numbers,
- we start with vertex v, assign a number 1 to it,
- we add 1 to the label set of each neighbor of v,
- we choose the unnumbered vertex in V with the largest label under set cardinality, assign a number 2 to it,
- continue until all vertices are numbered.

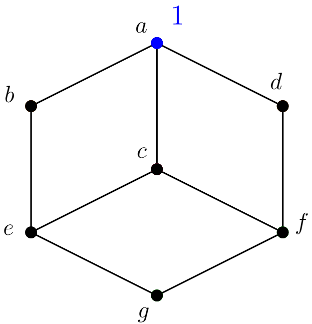
Example of MCS

assign the label $\emptyset$ to all vertices;
label(s) $\leftarrow \{n+1\}$;
foreach $i \leftarrow 1$ *to* n **do**
    pick an unnumbered vertex v with maximum label under set cardinality;
    $\sigma(i) \leftarrow v$;
    foreach *unnumbered vertex* $w \in N(v)$ **do**
        add i to label(w);
    end
end



Example of MCS

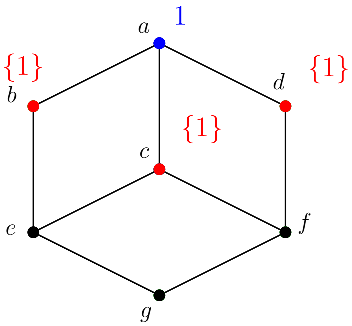
assign the label $\emptyset$ to all vertices;
label(s) $\leftarrow \{n+1\}$;
foreach $i \leftarrow 1$ *to* n **do**
    pick an unnumbered vertex v with maximum label under set cardinality;
    $\sigma(i) \leftarrow v$;
    foreach *unnumbered vertex* $w \in N(v)$ **do**
        add i to label(w);
    end
end



$$\sigma = (a)$$

Example of MCS

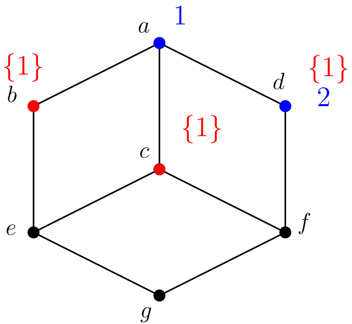
assign the label $\emptyset$ to all vertices;
label(s) $\leftarrow \{n+1\}$;
foreach $i \leftarrow 1$ *to* n **do**
  pick an unnumbered vertex v with maximum label under set cardinality;
  $\sigma(i) \leftarrow v$;
  foreach *unnumbered vertex* $w \in N(v)$ **do**
    add i to label(w);
  end
end



$$\sigma = (a)$$

Example of MCS

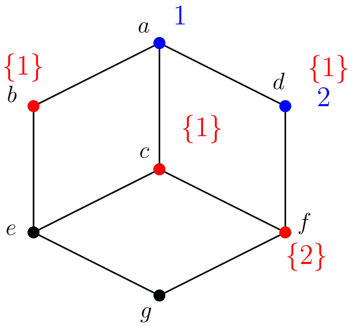
assign the label $\emptyset$ to all vertices;
label(s) $\leftarrow \{n+1\}$;
foreach $i \leftarrow 1$ *to n* **do**
    pick an unnumbered vertex v with maximum label under set cardinality;
    $\sigma(i) \leftarrow v$;
    foreach *unnumbered vertex* $w \in N(v)$ **do**
        add i to label(w);
    end
end



$$\sigma = (a, d)$$

Example of MCS

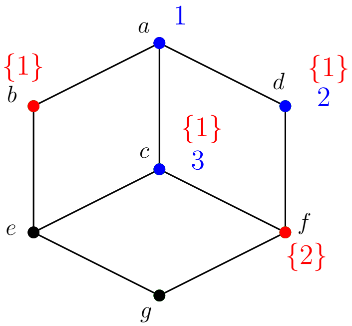
assign the label $\emptyset$ to all vertices;
label(s) $\leftarrow \{n+1\}$;
foreach $i \leftarrow 1$ *to* n **do**
  pick an unnumbered vertex v with maximum label under set cardinality;
  $\sigma(i) \leftarrow v$;
  foreach *unnumbered vertex* $w \in N(v)$ **do**
    add i to label(w);
  end
end



$$\sigma = (a, d)$$

Example of MCS

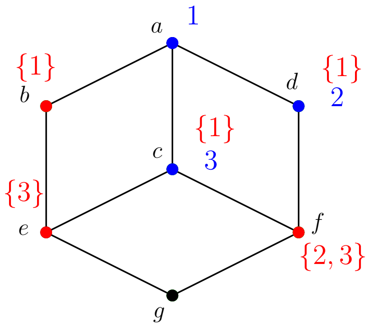
assign the label $\emptyset$ to all vertices;
label(s) $\leftarrow \{n+1\}$;
foreach $i \leftarrow 1$ *to* n **do**
  pick an unnumbered vertex v with maximum label under set cardinality;
  $\sigma(i) \leftarrow v$;
  foreach *unnumbered vertex* $w \in N(v)$ **do**
    add i to label(w);
  end
end



$$\sigma = (a, d, c)$$

Example of MCS

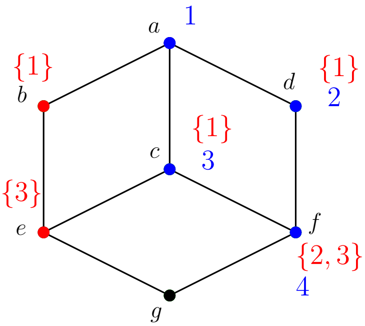
assign the label $\emptyset$ to all vertices;
label(s) $\leftarrow \{n+1\}$;
foreach $i \leftarrow 1$ *to* n **do**
    pick an unnumbered vertex v with maximum label under set cardinality;
    $\sigma(i) \leftarrow v$;
    foreach *unnumbered vertex* $w \in N(v)$ **do**
        add i to label(w);
    end
end



$$\sigma = (a, d, c)$$

Example of MCS

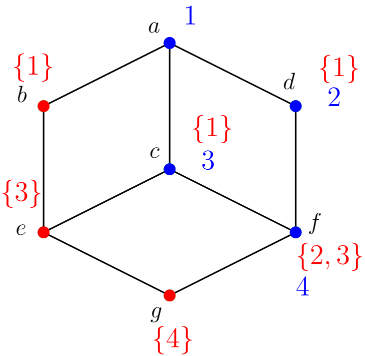
assign the label $\emptyset$ to all vertices;
label(s) $\leftarrow \{n+1\}$;
foreach $i \leftarrow 1$ *to* n **do**
    pick an unnumbered vertex v with maximum label under set cardinality;
    $\sigma(i) \leftarrow v$;
    foreach *unnumbered vertex* $w \in N(v)$ **do**
        add i to label(w);
    end
end



$$\sigma = (a, d, c, f)$$

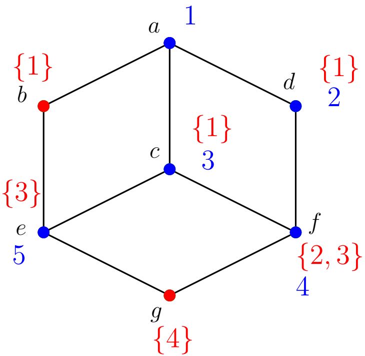
Example of MCS

assign the label $\emptyset$ to all vertices;
label(s) $\leftarrow \{n+1\}$;
foreach $i \leftarrow 1$ *to* n **do**
    pick an unnumbered vertex v with maximum label under set cardinality;
    $\sigma(i) \leftarrow v$;
    foreach *unnumbered vertex* $w \in N(v)$ **do**
        add i to label(w);
    end
end



$\sigma = (a, d, c, f)$

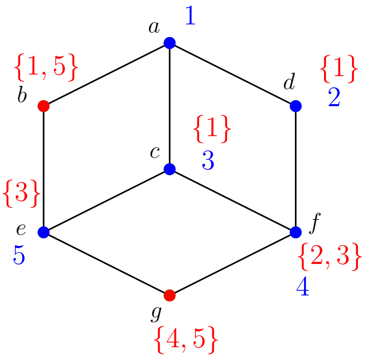
assign the label $\emptyset$ to all vertices;
label(s) $\leftarrow \{n+1\}$;
foreach $i \leftarrow 1$ *to* n **do**
    pick an unnumbered vertex v with maximum label under set cardinality;
    $\sigma(i) \leftarrow v$;
    foreach *unnumbered vertex* $w \in N(v)$ **do**
        add i to label(w);
    end
end



$\sigma = (a, d, c, f, e)$

Example of MCS

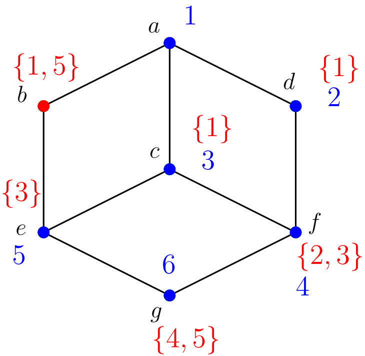
assign the label $\emptyset$ to all vertices;
label(s) $\leftarrow \{n+1\}$;
foreach $i \leftarrow 1$ *to* n **do**
    pick an unnumbered vertex v with maximum label under set cardinality;
    $\sigma(i) \leftarrow v$;
    foreach *unnumbered vertex* $w \in N(v)$ **do**
        add i to label(w);
    end
end



$$\sigma = (a, d, c, f, e)$$

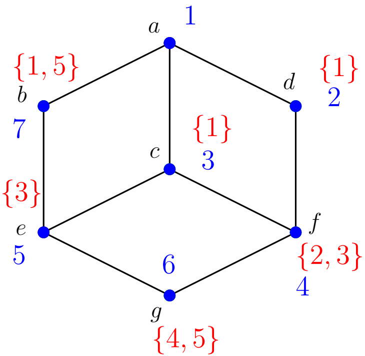
Example of MCS

assign the label $\emptyset$ to all vertices;
label(s) $\leftarrow \{n+1\}$;
foreach $i \leftarrow 1$ *to* n **do**
    pick an unnumbered vertex v with maximum label under set cardinality;
    $\sigma(i) \leftarrow v$;
    foreach *unnumbered vertex* $w \in N(v)$ **do**
        add i to label(w);
    end
end



$\sigma = (a, d, c, f, e, g)$

assign the label $\emptyset$ to all vertices;
label(s) $\leftarrow \{n+1\}$;
foreach $i \leftarrow 1$ *to* n **do**
    pick an unnumbered vertex v with maximum label under set cardinality;
    $\sigma(i) \leftarrow v$;
    foreach *unnumbered vertex* $w \in N(v)$ **do**
        add i to label(w);
    end
end



$\sigma = (a, d, c, f, e, g, b)$

Known results

Theorem (Korach and Ostfeld, 1989)

The DFS-tree recognition problem is solvable in polynomial time.

Known results

Theorem (Korach and Ostfeld, 1989)

The DFS-tree recognition problem is solvable in polynomial time.

Theorem (Manber, 1990)

The BFS-tree recognition problem is solvable in linear time.

Overview of results

Tree results	$\mathcal{F}$-BFS	$\mathcal{F}$-LBFS	$\mathcal{L}$-DFS	$\mathcal{L}$-LDFS	$\mathcal{F}$-MCS	$\mathcal{F}$-MNS
All Graphs	**L**	**NPC**	**L**	**P**	**NPC**	**NPC**
Weakly Chordal	L	**NPC**	L	**P**	**NPC**	**NPC**
Chordal	L	?	L	**P**	?	?
Split	L	**L**	L	**P**	**L**	**L**

- Hagerup and Nowak, 1985; Korach and Ostfeld, 1989
- Manber, 1990

Lemma (Tarjan, 1972)

Let $G = (V, E)$ be a graph and let T be an $\mathcal{L}$-tree of G generated by DFS. For each $uv \in E$ it holds that either $uv \in E(T)$ or, without loss of generality u is an ancestor of v in T.

A consequence of result by Korach and Ostfeld:

Lemma

Let $G = (V, E)$ be a graph with spanning tree T. Let G_i be an induced subgraph of G with a spanning tree T_i which is the restriction of T to G_i. If T is an $\mathcal{L}$-tree of LDFS on G, then T_i is an $\mathcal{L}$-tree of LDFS on G_i. In particular, if T is rooted in r, and $r \in T(V_i)$, then T_i is also rooted in r.

Lemma (Tarjan, 1972)

Let $G = (V, E)$ be a graph and let T be an $\mathcal{L}$-tree of G generated by DFS. For each $uv \in E$ it holds that either $uv \in E(T)$ or, without loss of generality u is an ancestor of v in T.

A consequence of result by Korach and Ostfeld:

Lemma

Let $G = (V, E)$ be a graph with spanning tree T. Let G_i be an induced subgraph of G with a spanning tree T_i which is the restriction of T to G_i. If T is an $\mathcal{L}$-tree of LDFS on G, then T_i is an $\mathcal{L}$-tree of LDFS on G_i. In particular, if T is rooted in r, and $r \in T(V_i)$, then T_i is also rooted in r.

Result: a polynomial algorithm for LDFS.

A polynomial algorithm for LDFS

Since this is DFS-like search $\rightarrow$ $\mathcal{L}$-tree.

Observe: the recognition of last visited vertex in LDFS is hard!

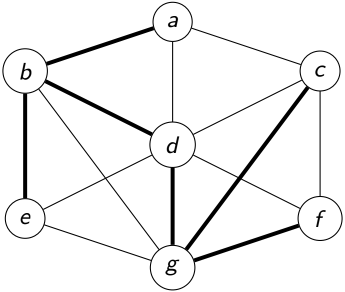
Theorem

The $\mathcal{L}$-tree recognition problem for LDFS can be solved in polynomial time.

Idea:

- we check for every vertex $v \in G$ whether there is LBFS starting at v that produces T
- start LBFS at vertex r
- after visiting u, choose a vertex v with lex.largest label s.t. $uv \in E(T)$
- prepend a number of u to label of its neigbbors

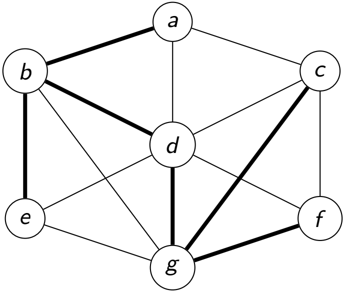
Example of algorithm execution



Check whether we can start LDFS in a.

- first we visit a,

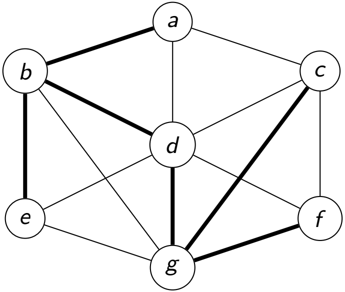
Example of algorithm execution



Check whether we can start LDFS in a.

- first we visit a,
- we choose b since $ab \in E(T)$ and b has lexicographically largest label,

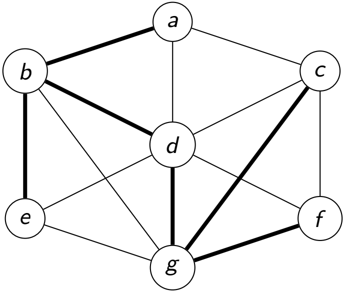
Example of algorithm execution



Check whether we can start LDFS in a.

- first we visit a,
- we choose b since $ab \in E(T)$ and b has lexicographically largest label,
- now we choose d,

Example of algorithm execution



Check whether we can start LDFS in a.

- first we visit a,
- we choose b since $ab \in E(T)$ and b has lexicographically largest label,
- now we choose d,
- now we should choose c, but $dc \notin E(T) \Rightarrow$ **contradiction!**

NP-completeness for LBFS

Since BFS-like search $\rightarrow$ $\mathcal{F}$-tree.

Theorem

The $\mathcal{F}$-tree recognition problem for LDFS is NP-complete for weakly chordal graphs.

Since BFS-like search $\rightarrow$ $\mathcal{F}$-tree.

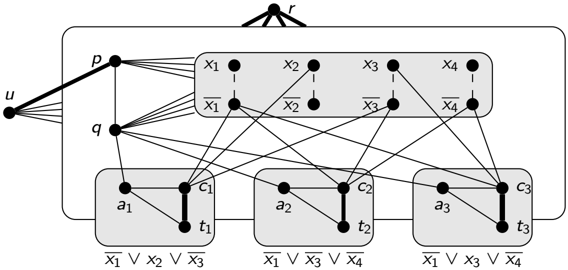
Theorem

The $\mathcal{F}$-tree recognition problem for LDFS is NP-complete for weakly chordal graphs.

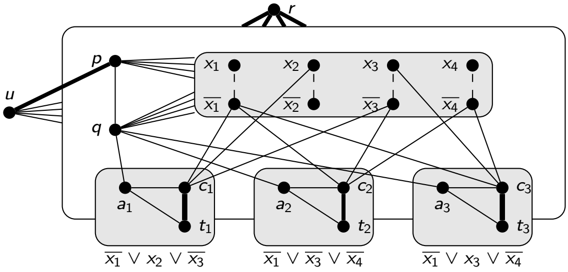
Proof.

Polynomial reduction from 3-SAT.
Assume $\mathcal{I} = (x_1, \ldots, x_n, C_1, \ldots, C_m)$ is an instance of 3-SAT. $\square$

Polynomial reduction



Polynomial reduction



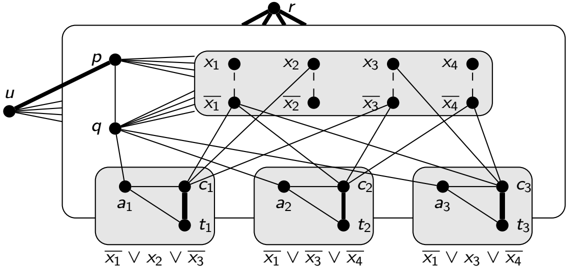
Proposition

$\mathcal{I}$ admits a satisfying assignment if and only if $T(\mathcal{I})$ is an $\mathcal{F}$-tree of LBFS on $G(\mathcal{I})$.

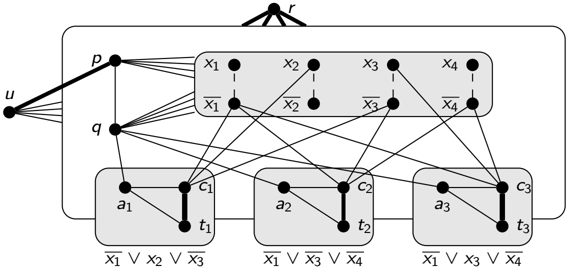
Proposition

For all $\mathcal{I}$, the graph $G(\mathcal{I})$ is weakly chordal.

Polynomial reduction



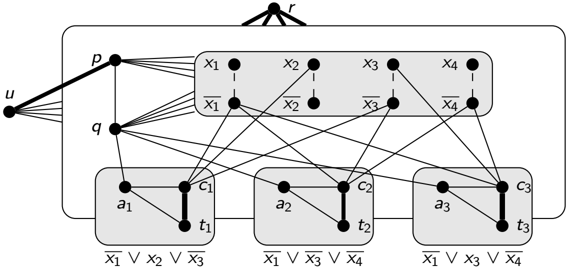
Polynomial reduction



$\Rightarrow$ If we have a satisfying assignment $\mathcal{A}$, we do the search as follows

- visit r, and then p
- visit literals from $\mathcal{A}$
- visit q, and then the remaining of X
- visit u and then visit the clause vertices c_i
- visit a_i and then t_i

Polynomial reduction



Assume now that $\mathcal{I}$ has no satisfying assignment. Observe:

- LBFS must start in r
- we must choose p (otherwise $pu \notin E(T)$)
- if we choose q: a_i visited before $c_i \to t_i a_i \in E(T)$, so we visit something in X,
- visit some literal, then q, and then the neqation of literal, so we visit some assignment before q
- since not satisfiable, one a_i visited before $c_i \Rightarrow T$ is not a corresponding tree.

NP-completeness for MCS and MNS

Since BFS-like search $\rightarrow$ $\mathcal{F}$-tree.

Theorem

The $\mathcal{F}$-tree recognition problem for MNS and MCS is NP-complete for weakly chordal graphs.

NP-completeness for MCS and MNS

Since BFS-like search $\rightarrow$ $\mathcal{F}$-tree.

Theorem

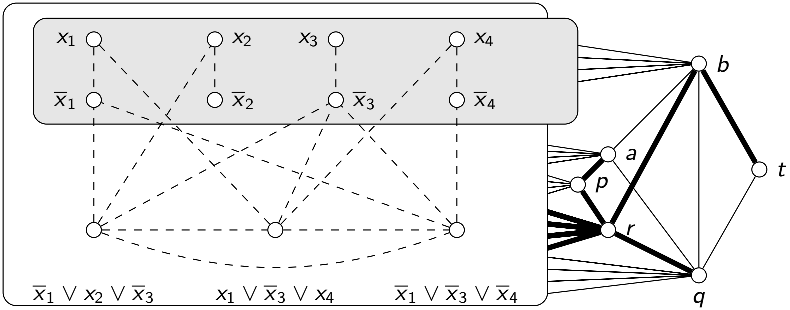
The $\mathcal{F}$-tree recognition problem for MNS and MCS is NP-complete for weakly chordal graphs.

Proof.

Polynomial reduction from 3-SAT.

Assume $\mathcal{I} = (x_1, \ldots, x_n, C_1, \ldots, C_m)$ is an instance of 3-SAT.

Polyomial reduction



Split graphs

In general: $\mathcal{F}$-trees for BFS, MNS, MCS are not the same.

Split graphs

In general: $\mathcal{F}$-trees for BFS, MNS, MCS are not the same.

Theorem

A tree T is an $\mathcal{F}$-tree of BFS on a split graph G if and only if it is an $\mathcal{F}$-tree of MNS (MCS, LBFS, LDFS).

Split graphs

In general: $\mathcal{F}$-trees for BFS, MNS, MCS are not the same.

Theorem

A tree T is an $\mathcal{F}$-tree of BFS on a split graph G if and only if it is an $\mathcal{F}$-tree of MNS (MCS, LBFS, LDFS).

Theorem (Manber)

The $\mathcal{F}$-tree problem can be solved in linear time for BFS on split graphs.

Split graphs

In general: $\mathcal{F}$-trees for BFS, MNS, MCS are not the same.

Theorem

A tree T is an $\mathcal{F}$-tree of BFS on a split graph G if and only if it is an $\mathcal{F}$-tree of MNS (MCS, LBFS, LDFS).

Theorem (Manber)

The $\mathcal{F}$-tree problem can be solved in linear time for BFS on split graphs.

Corollary

The $\mathcal{F}$-tree problem of MNS, MCS, LBFS and LDFS can be solved in linear time.

Split graphs

In general: $\mathcal{F}$-trees for BFS, MNS, MCS are not the same.

Theorem

A tree T is an $\mathcal{F}$-tree of BFS on a split graph G if and only if it is an $\mathcal{F}$-tree of MNS (MCS, LBFS, LDFS).

Theorem (Manber)

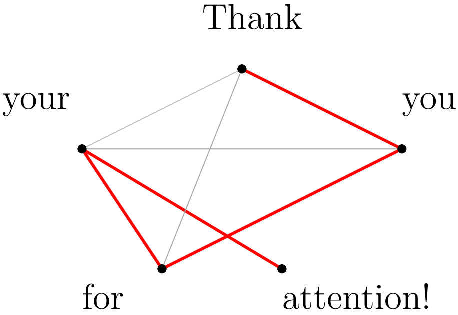
The $\mathcal{F}$-tree problem can be solved in linear time for BFS on split graphs.

Corollary

The $\mathcal{F}$-tree problem of MNS, MCS, LBFS and LDFS can be solved in linear time.

Theorem

The $\mathcal{L}$-tree problem of MNS, MCS, LBFS and LDFS can be solved in linear time.


Thank
you
your
for
attention!