On the Power of graph searching

Michel Habib
habib@liafa.jussieu.fr
http://www.liafa.jussieu.fr/~habib

Evaluation INRIA,
Paris, March 2012

Schedule

- Generic Search
- Lexicographic Breadth First Search LexBFS
- Lexicographic Depth First Search LexDFS
- Importance of 4 points conditions for graph recognition
- Lego with basic graph searches
- Principle of a Composition of Searches
- Some applications
- Some Perspectives for Gang

Joint work with D. Corneil (Toronto), P. Crescenzi (Florence), J. Dusart (PhD GANG), R. Grossi (Pise)
and many others F. de Montgolfier, D. Fortin ...

Graph searches are very well known and used

1. Euler (1735) for solving the famous walk problem in Kœnisberg

Graph searches are very well known and used

1. Euler (1735) for solving the famous walk problem in Kœnisberg
2. Tremaux (1882) and Tarry (1895) using DFS to solve maze problems

Graph searches are very well known and used

1. Euler (1735) for solving the famous walk problem in Kœnisberg
2. Tremaux (1882) and Tarry (1895) using DFS to solve maze problems
3. Computer scientists from 1950 ....

Graph searches are very well known and used

1. Euler (1735) for solving the famous walk problem in Kœnisberg
2. Tremaux (1882) and Tarry (1895) using DFS to solve maze problems
3. Computer scientists from 1950 ....
4. But also : "Fil d'ariane" in the Greek mythology.

Tarry

LE PROBLÈME DES LABYRINTHES;

PAR M. G. TARRY.

Tout labyrinthe peut être parcouru en une seule course, en passant deux fois en sens contraire par chacune des allées, sans qu'il soit nécessaire d'en connaître le plan.

Pour résoudre ce problème, il suffit d'observer cette règle unique :

Ne reprendre l'allée initiale qui a conduit à un carrefour pour la première fois que lorsqu'on ne peut pas faire autrement.

Nous ferons d'abord quelques remarques.

A un moment quelconque, avant d'arriver à un car-

Umberto Eco, "Il nome della rosa", Roman, 1980

« Pour trouver la sortie d'un labyrinthe, récita en effet Guillaume, il n'y a qu'un moyen. A chaque nœud nouveau, autrement dit jamais visité avant, le parcours d'arrivée sera marqué de trois signes. Si, à cause de signes précédents sur l'un des chemins du nœud, on voit que ce nœud a déjà été visité, on placera un seul signe sur le parcours d'arrivée. Si tous les passages ont été déjà marqués, alors il faudra reprendre la même voie, en revenant en arrière. Mais si un ou deux passages du nœud sont encore sans signes, on en choisira un quelconque, pour y apposer deux signes. Quand on s'achemine par un passage qui porte un seul signe, on en appposera deux autres, de façon que ce passage en porte trois dorénavant. Toutes les parties du labyrinthe devraient avoir été parcourues si, en arrivant à un nœud, on ne prend jamais le passage avec trois signes, sauf si d'autres passages sont encore sans signes.

– Comment le savez-vous? Vous êtes expert en labyrinthes?

– Non, je récite un extrait d'un texte antique que j'ai lu autrefois.

– Et selon cette règle, on sort?

– Presque jamais, que je sache. Mais nous tenterons quand même. Et puis dans les prochains jours j'aurai des verres et j'aurai le temps de mieux me pencher sur les livres. Il se peut que là où le parcours des cartouches nous embrouille, celui des livres nous donne une règle.

Three main aspects for a graph search :

1. its principle or its algorithm
   (i.e. the description of the tie-break rules for the choice of the next vertex (edge) to be explored)

Three main aspects for a graph search :

1. its principle or its algorithm
   (i.e. the description of the tie-break rules for the choice of the next vertex (edge) to be explored)
2. The study of the underlying tree structure

Three main aspects for a graph search :

1. its principle or its algorithm
   (i.e. the description of the tie-break rules for the choice of the next vertex (edge) to be explored)
2. The study of the underlying tree structure
3. The complexity analysis and its implementation or its program

We will focus here on a fourth one :

- Here we consider a graph search as an operator producing of a total ordering on the set of vertices (the order in which the vertices are visited by the search)

We will focus here on a fourth one :

- Here we consider a graph search as an operator producing of a total ordering on the set of vertices (the order in which the vertices are visited by the search)
- I will try to convince you that this viewpoint can be helpful

- Building bottom up graph algorithms form well-known

- Building bottom up graph algorithms form well-known
- Develop basic theoretic tools for the structural analysis of graphs

- Building bottom up graph algorithms form well-known
- Develop basic theoretic tools for the structural analysis of graphs
- No need to store sophisticated data structures, just a labeling of the vertices, can be used for huge graphs

- Building bottom up graph algorithms form well-known
- Develop basic theoretic tools for the structural analysis of graphs
- No need to store sophisticated data structures, just a labeling of the vertices, can be used for huge graphs
- Fundamental Question :
  Which kind of knowledge can we learn about a graph via graph searching (i.e. with one or a series of successive graph searches) ?

Problem

For an undirected graph $G = (V, E)$,
explore the vertices of G "traversing or following" the edges.

Problem

For an undirected graph $G = (V, E)$,
explore the vertices of G "traversing or following" the edges.

Result

- a tree structure rooted at the first visited vertex

Problem

For an undirected graph $G = (V, E)$,
explore the vertices of G "traversing or following" the edges.

Result

- a tree structure rooted at the first visited vertex
- an ordering σ of the vertices

Problem

For an undirected graph $G = (V, E)$,
explore the vertices of G "traversing or following" the edges.

Result

- a tree structure rooted at the first visited vertex
- an ordering σ of the vertices

Questions

- Under which conditions an ordering σ of the vertices corresponds to a search ?

Problem

For an undirected graph $G = (V, E)$,
explore the vertices of G "traversing or following" the edges.

Result

- a tree structure rooted at the first visited vertex
- an ordering σ of the vertices

Questions

- Under which conditions an ordering σ of the vertices corresponds to a search ?
- What are the properties of these orderings ?

Important reference for this today lecture :

These easy questions have been only recently systematically considered :
D.G. Corneil et R. M. Krueger, A unified view of graph searching, SIAM J. Discrete Math, 22, Num 4 (2008) 1259-1276

Generic Search

Lexicographic Breadth First Search LexBFS

Lexicographic Depth First Search LexDFS

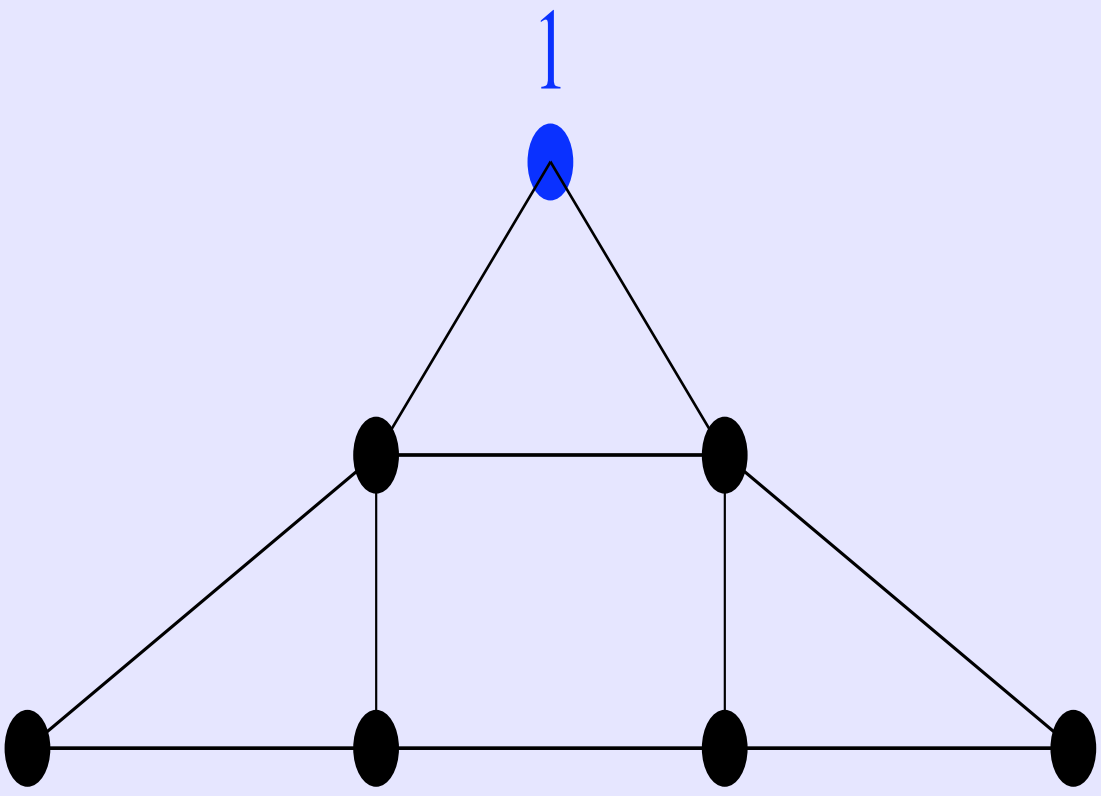
Importance of 4 points conditions for graph recognition

Lego with basic graph searches

Principle of a Composition of Searches

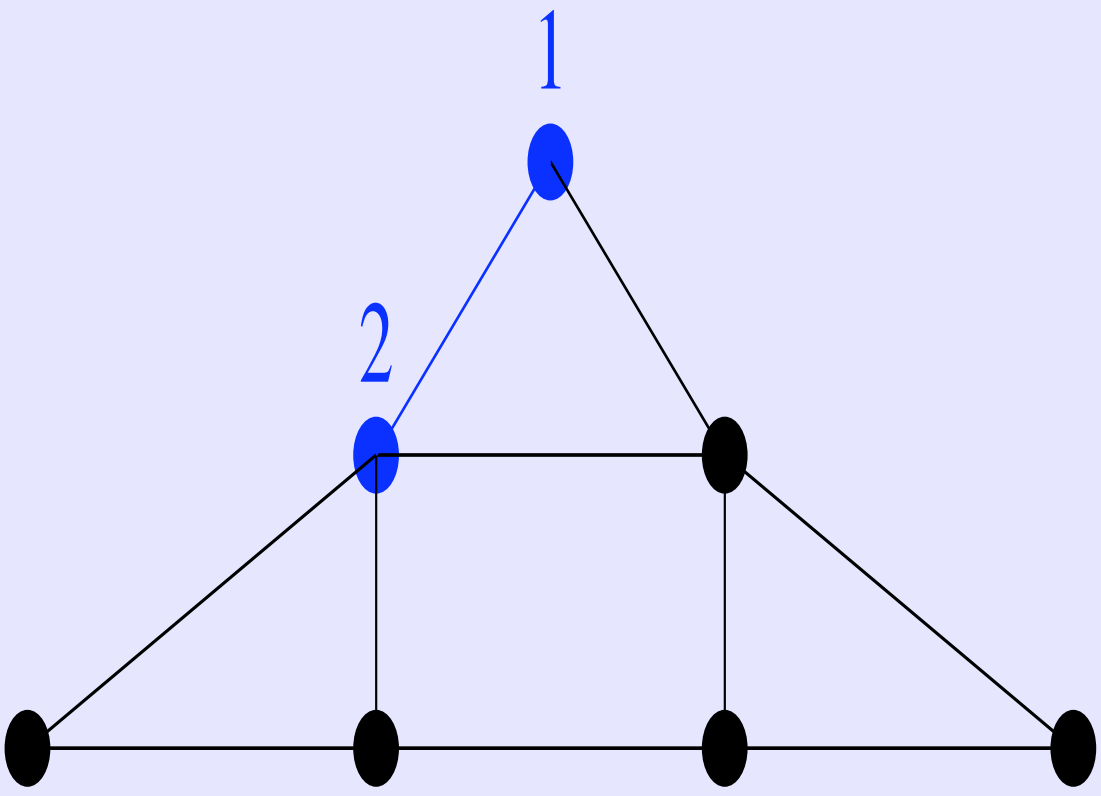
Some applications

Some Perspectives for Gang



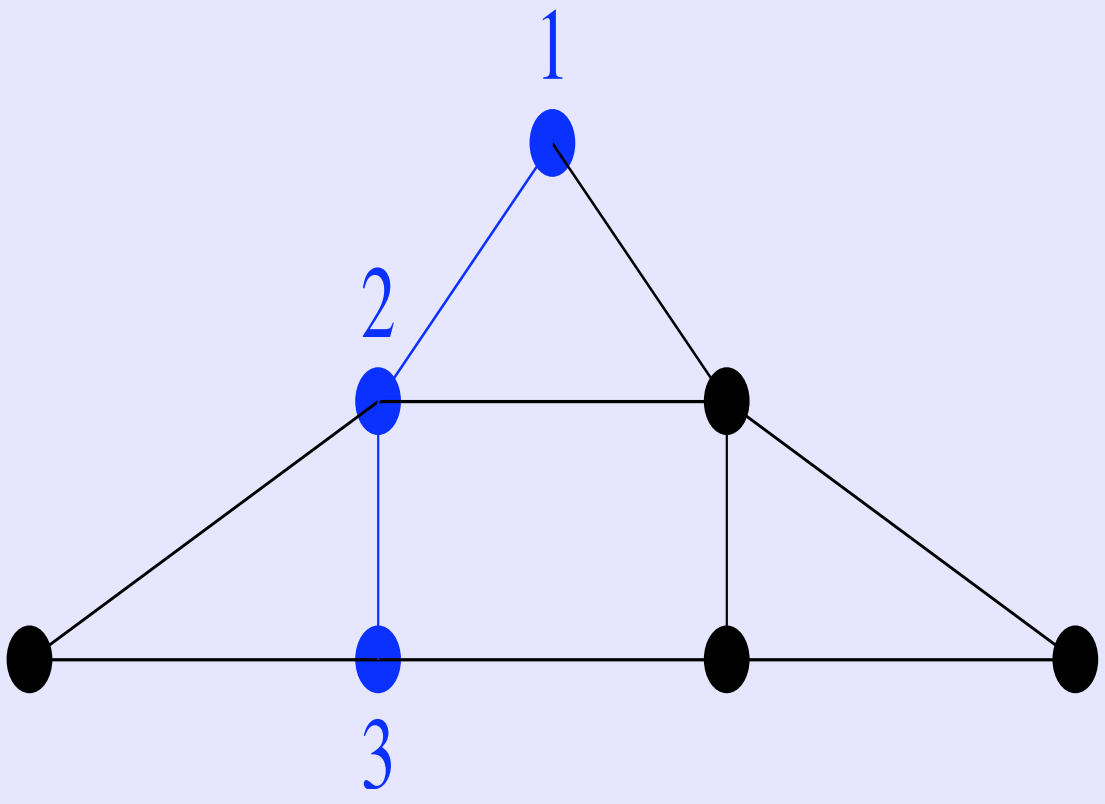
Invariant

At each step, an edge between a visited vertex and a unvisited one is selected



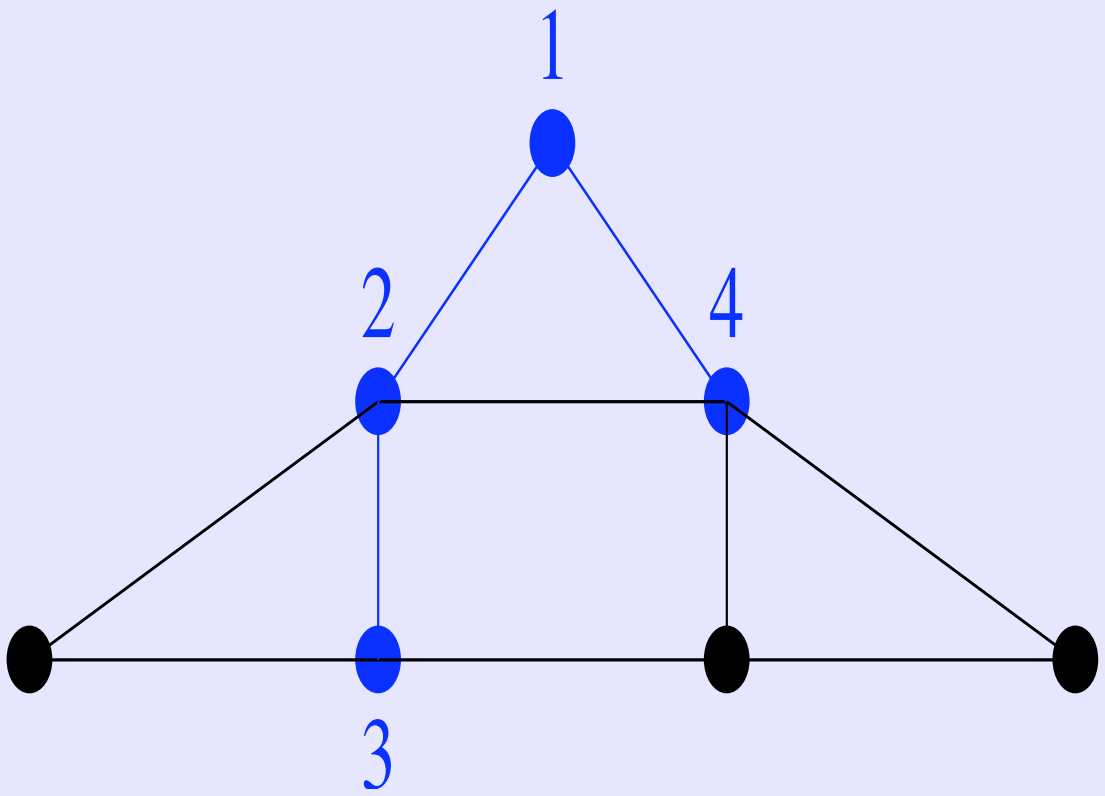
Invariant

At each step, an edge between a visited vertex and a unvisited one is selected



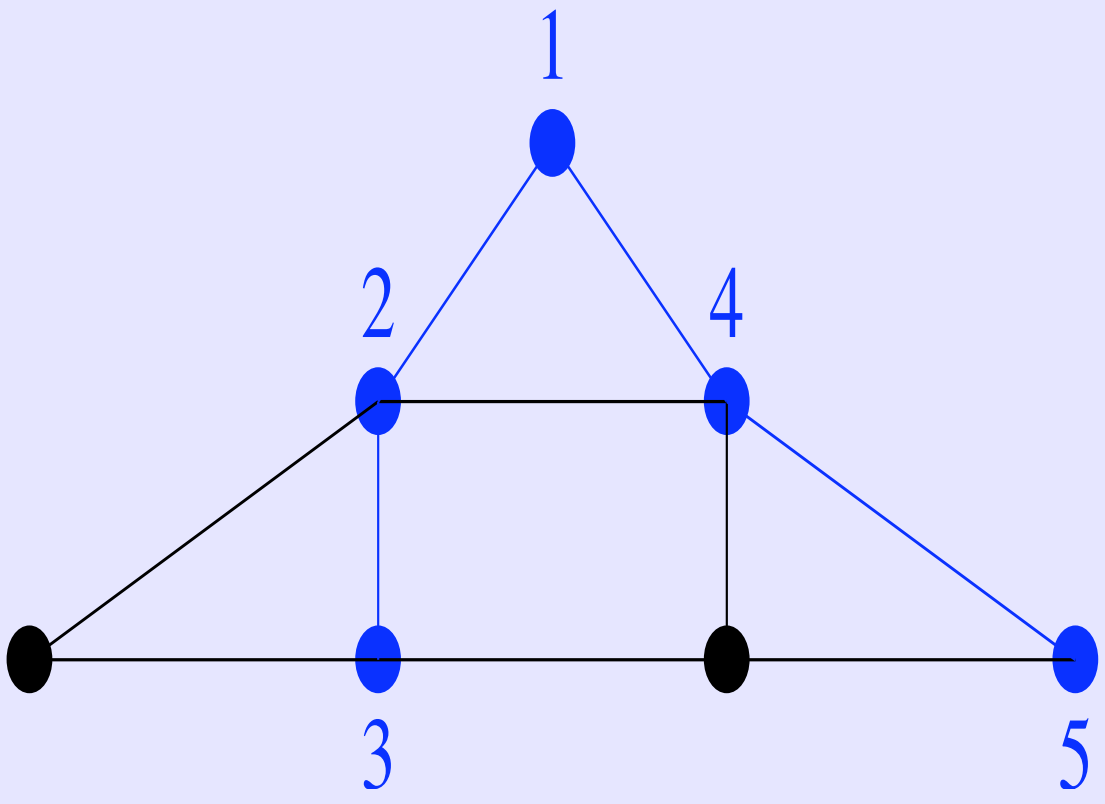
Invariant

At each step, an edge between a visited vertex and a unvisited one is selected



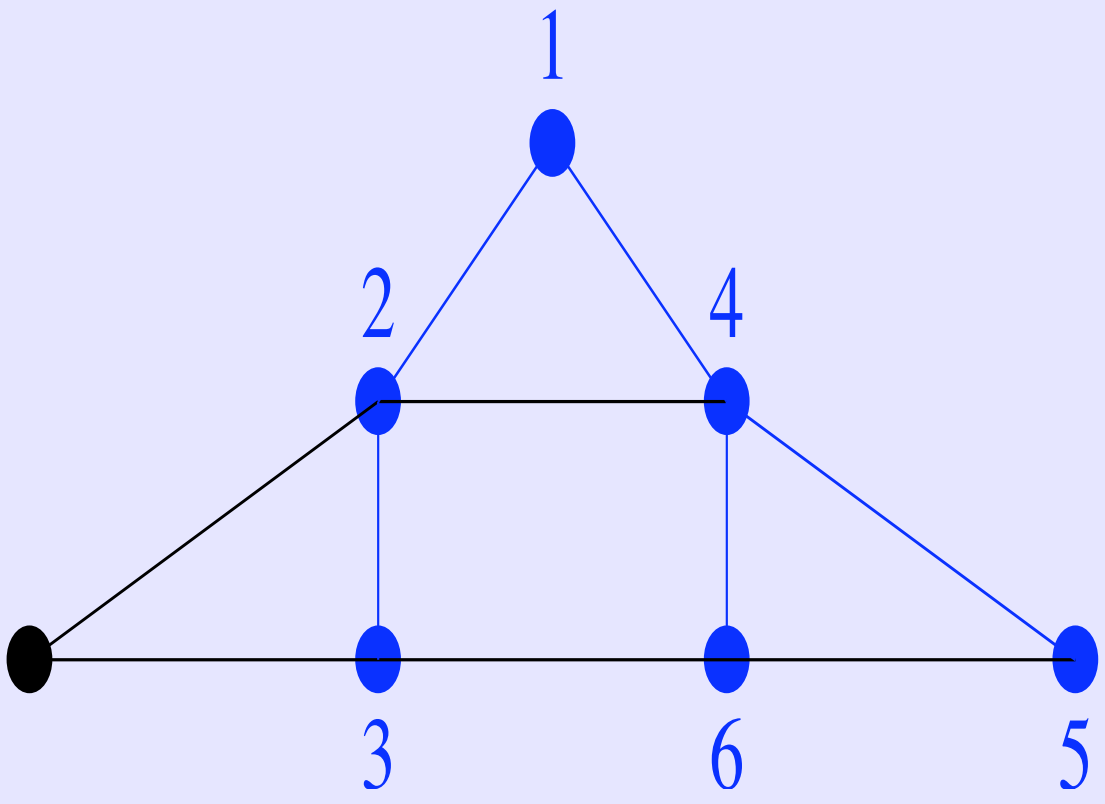
Invariant

At each step, an edge between a visited vertex and a unvisited one is selected



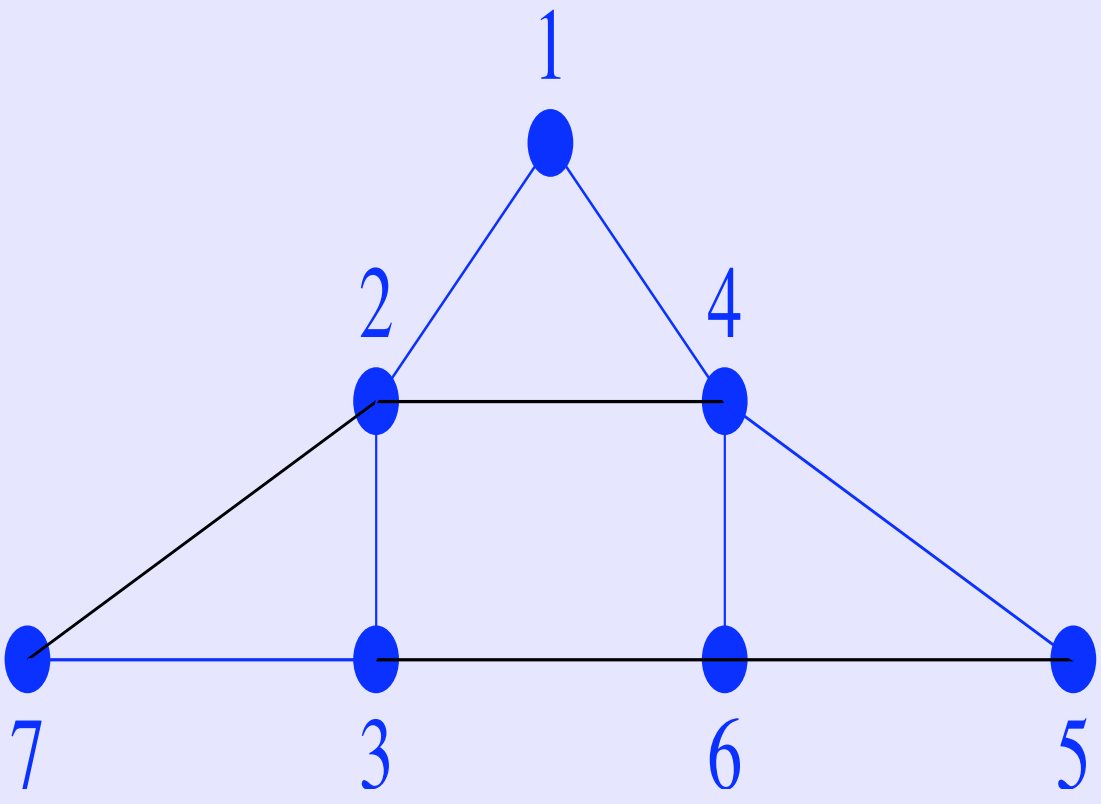
Invariant

At each step, an edge between a visited vertex and a unvisited one is selected



Invariant

At each step, an edge between a visited vertex and a unvisited one is selected



Invariant

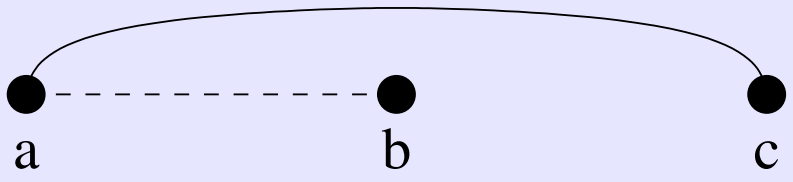
At each step, an edge between a visited vertex and a unvisited one is selected

Generic search

$S \leftarrow \{s\}$
for $i \leftarrow 1$ **à** n **do**
    Pick an unumbered vertex v of S
    $\sigma(i) \leftarrow v$
    foreach *unumbered vertex* $w \in N(v)$ **do**
        if $w \notin S$ **then**
            Add w to S
        end
    end
end

Generic question ?

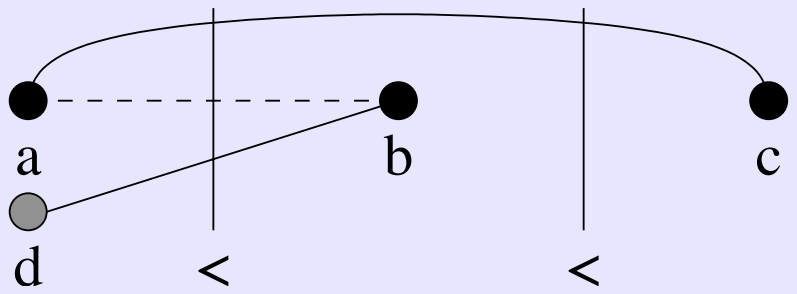
Let a, b et c be 3 vertices such that $ab \notin E$ et $ac \in E$.



Under which condition could we visit first a then b and last c ?

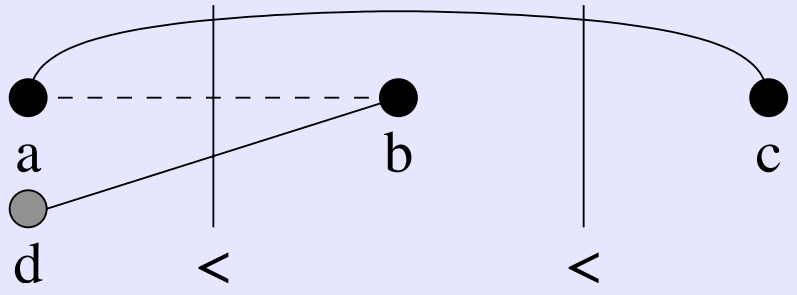
Property (Gen)

For an ordering σ on V, if $a < b < c$ and $ac \in E$ and $ab \notin E$, then it must exist a vertex d such that $d < b$ et $db \in E$



Property (Gen)

For an ordering σ on V, if $a < b < c$ and $ac \in E$ and $ab \notin E$, then it must exist a vertex d such that $d < b$ et $db \in E$



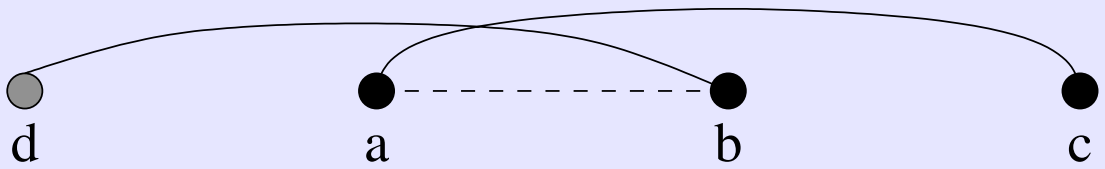
Theorem

For a graph $G = (V, E)$, an ordering σ sur V is a generic search of G iff σ satisfies property (Gen).

Most of the searches that we will study are refinement of this generic search
i.e. we just add new rules to follow for the choice of the next vertex to be visited
Graph searches mainly differ by the management of the tie-break set

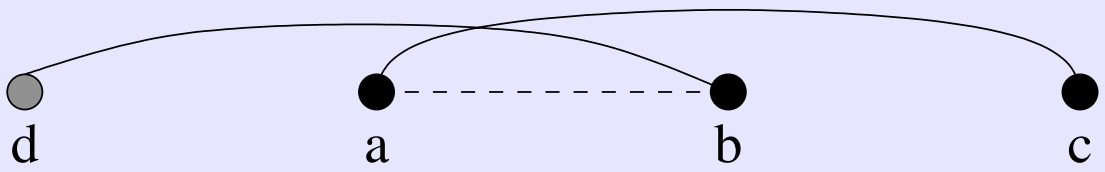
Property (B)

For an ordering σ on V, if $a < b < c$ and $ac \in E$ and $ab \notin E$, then it must exist a vertex d such that $d < a$ et $db \in E$



Property (B)

For an ordering σ on V, if $a < b < c$ and $ac \in E$ and $ab \notin E$, then it must exist a vertex d such that $d < a$ et $db \in E$

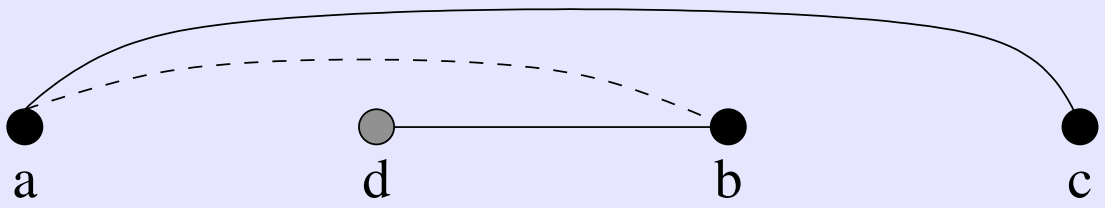


Theorem

For a graph $G = (V, E)$, an ordering σ sur V is a BFS of G iff σ satisfies property (B).

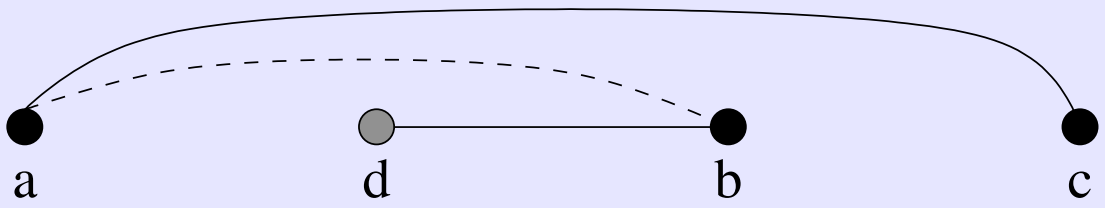
Proprerty (D)

For an ordering σ on V, if $a < b < c$ and $ac \in E$ and $ab \notin E$, then it must exist a vertex d such that $a < d < b$ and $db \in E$.



Proprerty (D)

For an ordering σ on V, if $a < b < c$ and $ac \in E$ and $ab \notin E$, then it must exist a vertex d such that $a < d < b$ and $db \in E$.



Theorem

For a graph $G = (V, E)$, an ordering σ sur V is a DFS of G iff σ satisfies property (D).

Applications of BFS

1. Distance computations (unit length), diameter and centers
2. BFS provides a useful layered structure of the graph
3. Using BFS to search an augmenting path provides a polynomial implementation of Ford-Fulkerson maximum flow algorithm.

Applications of DFS

Some applications

- Planarity testing.
- Computation of 2-connected (resp. strongly connected) components, 2-SAT solvers
- Computation of a linear extension (topological sorting) for an acyclic digraph, applications to inheritance mechanisms. . . .

Lexicographic Breadth First Search (LexBFS)

Data: a graph $G = (V, E)$ and a start vertex s

Result: an ordering σ of V

Assign the label $\emptyset$ to all vertices

$label(s) \leftarrow \{n\}$

for $i \leftarrow n$ **à** 1 **do**

    Pick an unumbered vertex v with lexicographically largest label

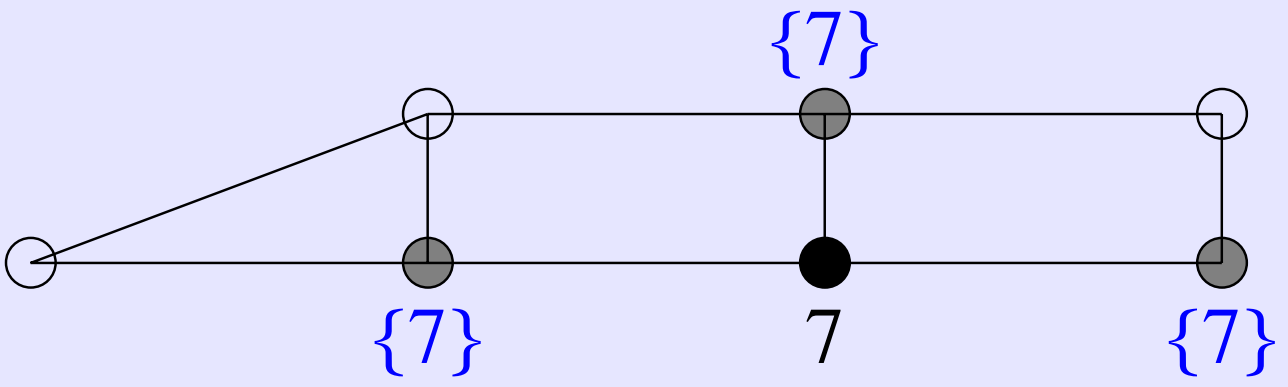
    $\sigma(i) \leftarrow v$

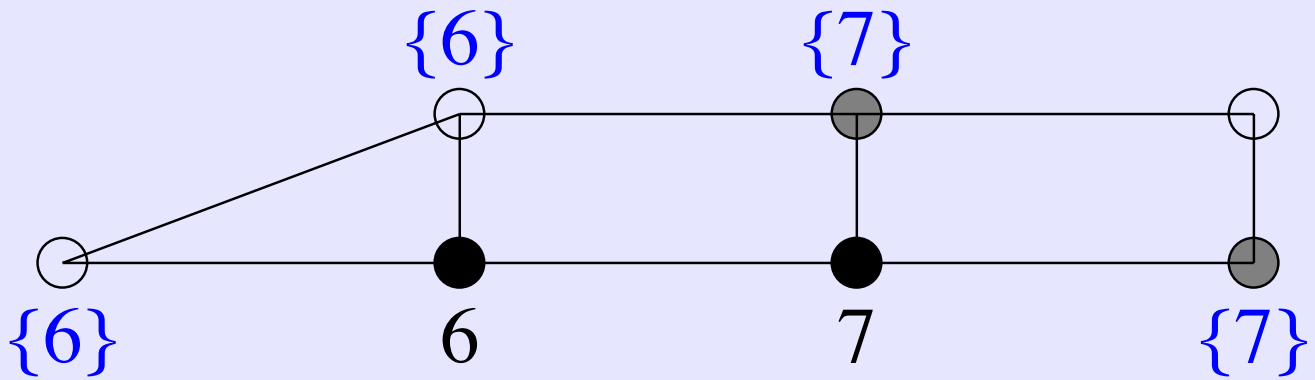
    foreach *unnumbered vertex* w *adjacent to* v **do**

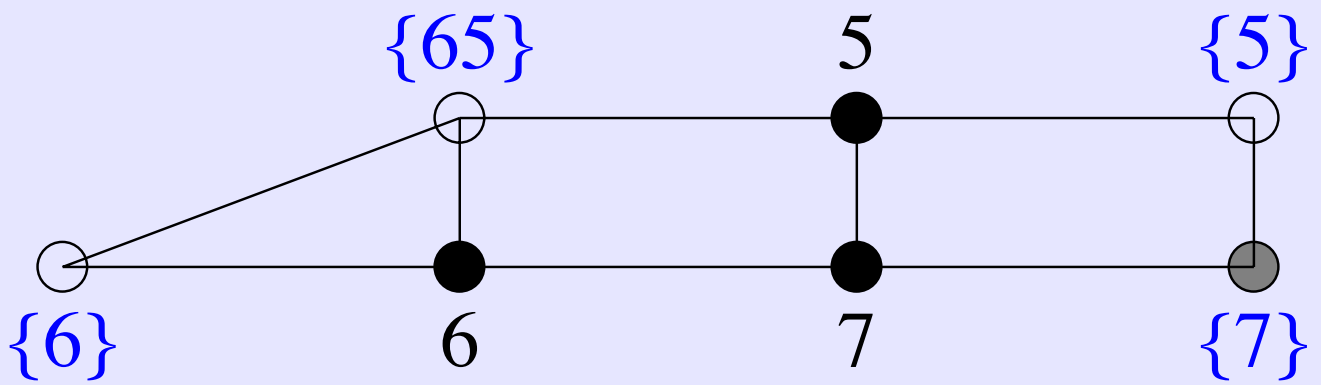
        $label(w) \leftarrow label(w).\{i\}$

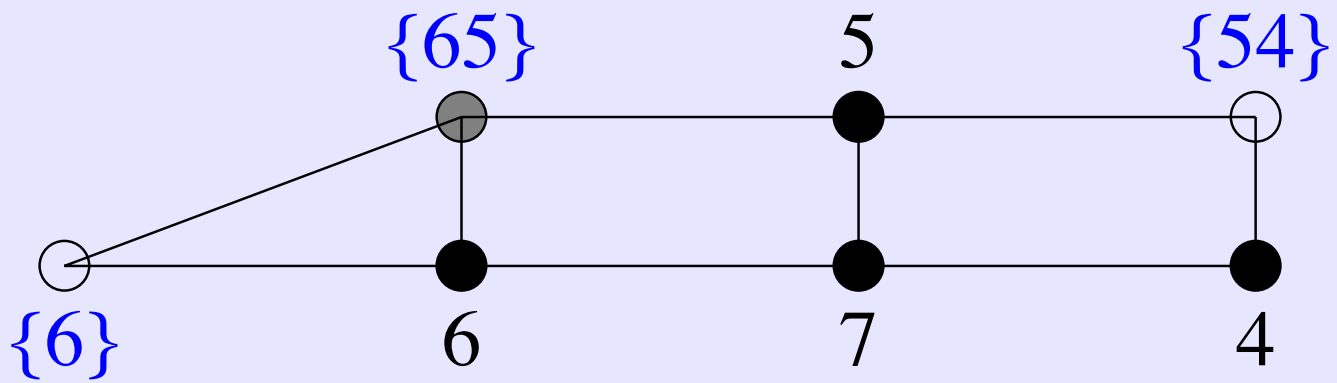
    end

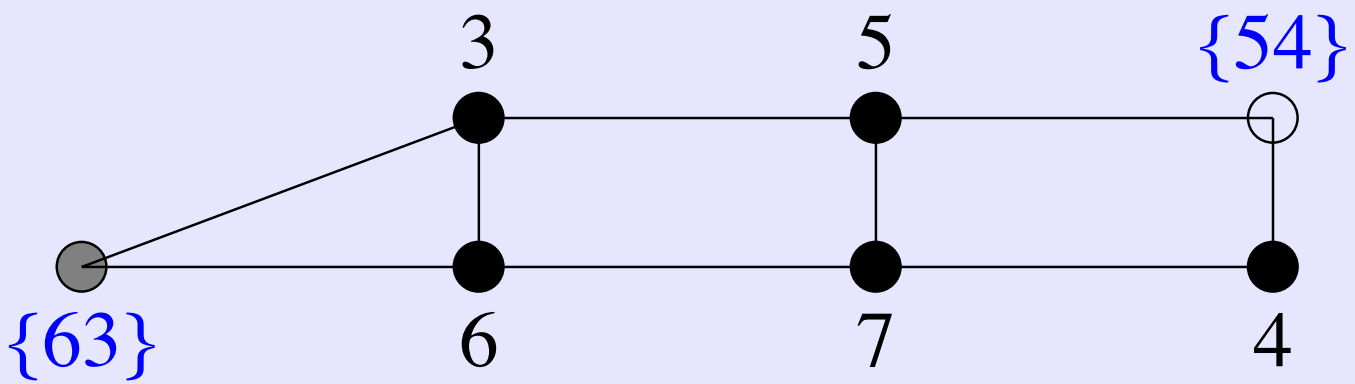
end

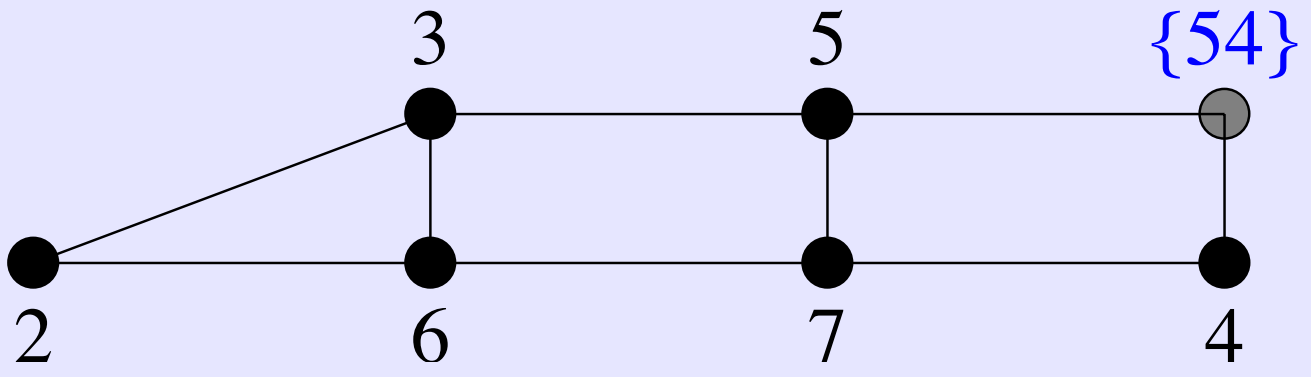

{7}
{7}
7
{7}

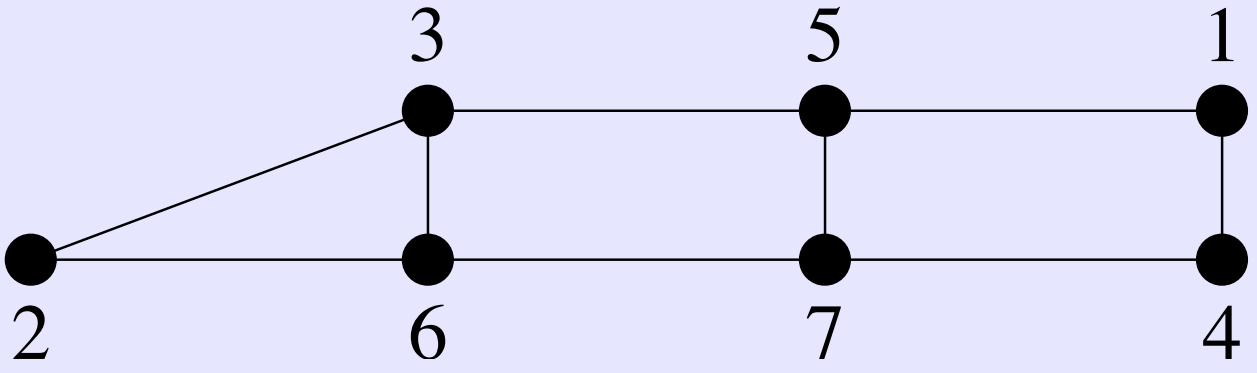

{6}
{7}
{6}
6
7
{7}


{65}
5
{5}
{6}
6
7
{7}


{65}
5
{54}
{6}
6
7
4


3
5
{54}
{63}
6
7
4


3
5
{54}
2
6
7
4

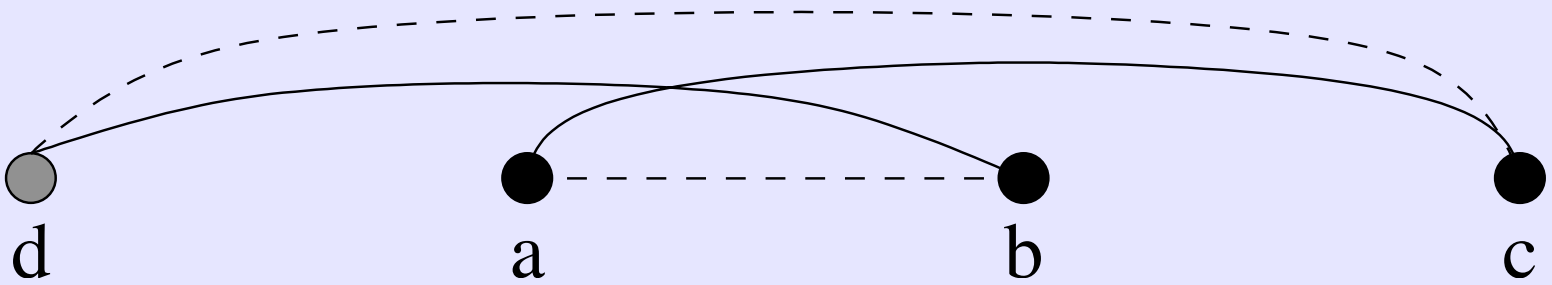

3
5
1
2
6
7
4

It is just a breadth first search with a tie break rule.

We are now considering a characterization of the order in which a LexBFS explores the vertices.

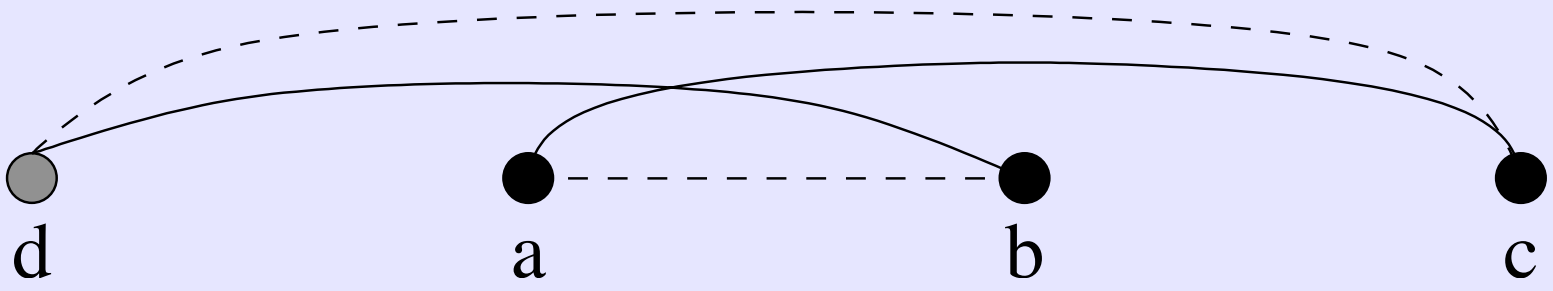
Property (LexB)

For an ordering σ on V, if $a < b < c$ and $ac \in E$ and $ab \notin E$, then it must exist a vertex d such that $d < a$ et $db \in E$ et $dc \notin E$.



Property (LexB)

For an ordering σ on V, if $a < b < c$ and $ac \in E$ and $ab \notin E$, then it must exist a vertex d such that $d < a$ et $db \in E$ et $dc \notin E$.



Theorem

For a graph $G = (V, E)$, an ordering σ sur V is a LexBFS of G iff σ satisfies property (LexB).

Why LexBFS behaves so nicely on well-structured graphs

A nice recursive property :
On every tie-break set S, LexBFS operates on $G(S)$ as a LexBFS.
Analogous properties are false for other classical searches.

Applications of LexBFS

1. Most famous one : Chordal graph recognition

Applications of LexBFS

1. Most famous one : Chordal graph recognition
2. For many classes of graphs using LexBFS ordering "backward" provides structural information on the graph.

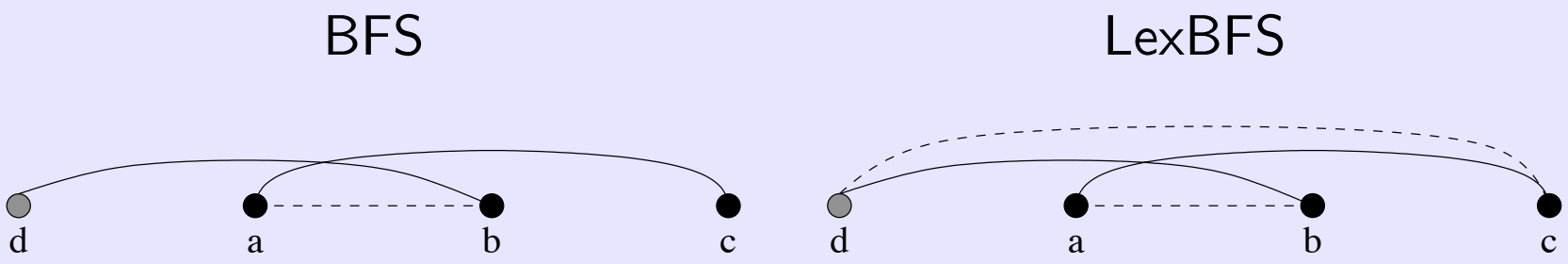
Applications of LexBFS

1. Most famous one : Chordal graph recognition
2. For many classes of graphs using LexBFS ordering "backward" provides structural information on the graph.
3. Last visited vertex (or clique) has some property (example simplicial for chordal graph)

Of course property LexB was known by authors such as M. Golumbic to study
chordal graphs but they did not noticed that it was a characterization
of LexBFS

LexDFS

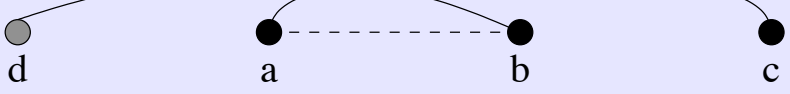
BFS vs LexBFS



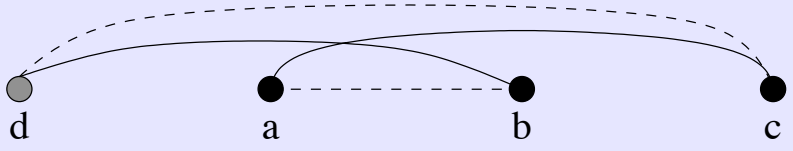
LexDFS

BFS vs LexBFS

BFS

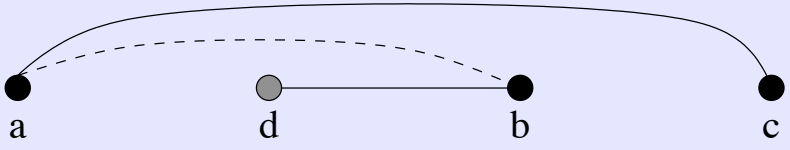


LexBFS



DFS vs LexDFS

DFS

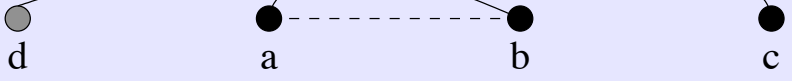


LexDFS

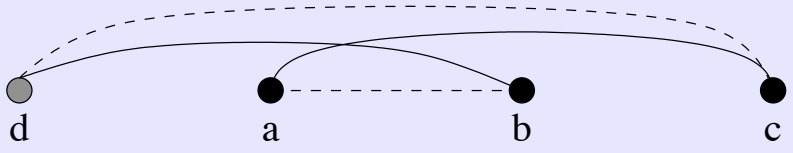
LexDFS

BFS vs LexBFS

BFS

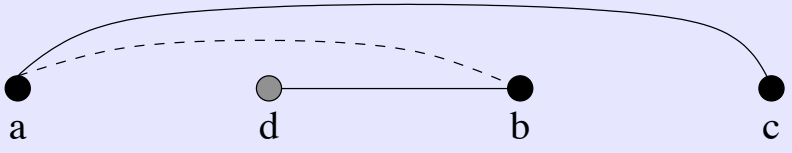


LexBFS

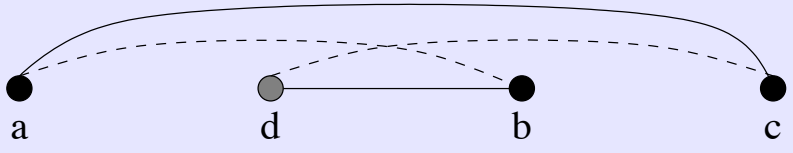


DFS vs LexDFS

DFS

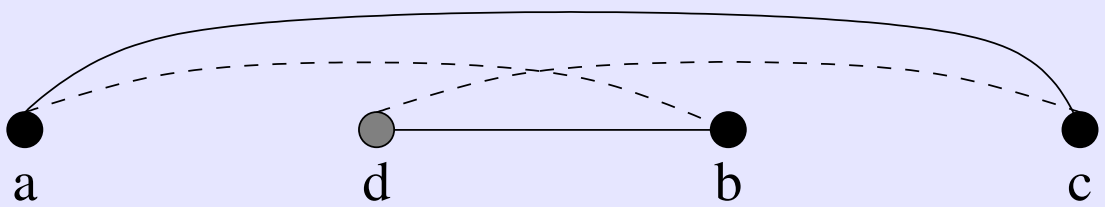


LexDFS



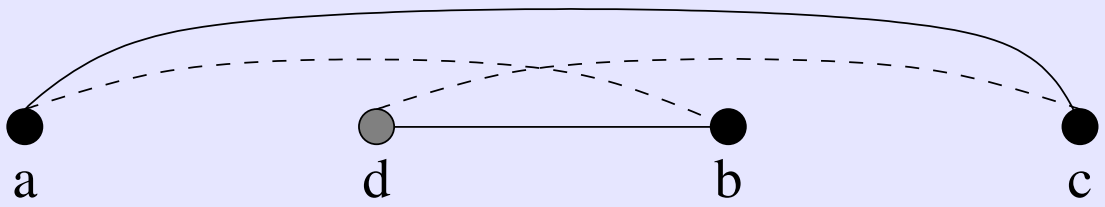
Property (LD)

For an ordering σ on V, if $a < b < c$ and $ac \in E$ and $ab \notin E$, then it must exist a vertex d such that $a < d < b$ and $db \in E$ and $dc \notin E$.



Property (LD)

For an ordering σ on V, if $a < b < c$ and $ac \in E$ and $ab \notin E$, then it must exist a vertex d such that $a < d < b$ and $db \in E$ and $dc \notin E$.



Theorem

For a graph $G = (V, E)$, an ordering σ sur V is a LexDFS of G iff σ satisfies property (LD).

Lexicographic Depth First Search (LexDFS)

Data: a graph $G = (V, E)$ and a start vertex s

Result: an ordering σ of V

Assign the label $\emptyset$ to all vertices

$label(s) \leftarrow \{0\}$

for $i \leftarrow 1$ **à** n **do**

    Pick an unumbered vertex v with lexicographically largest label

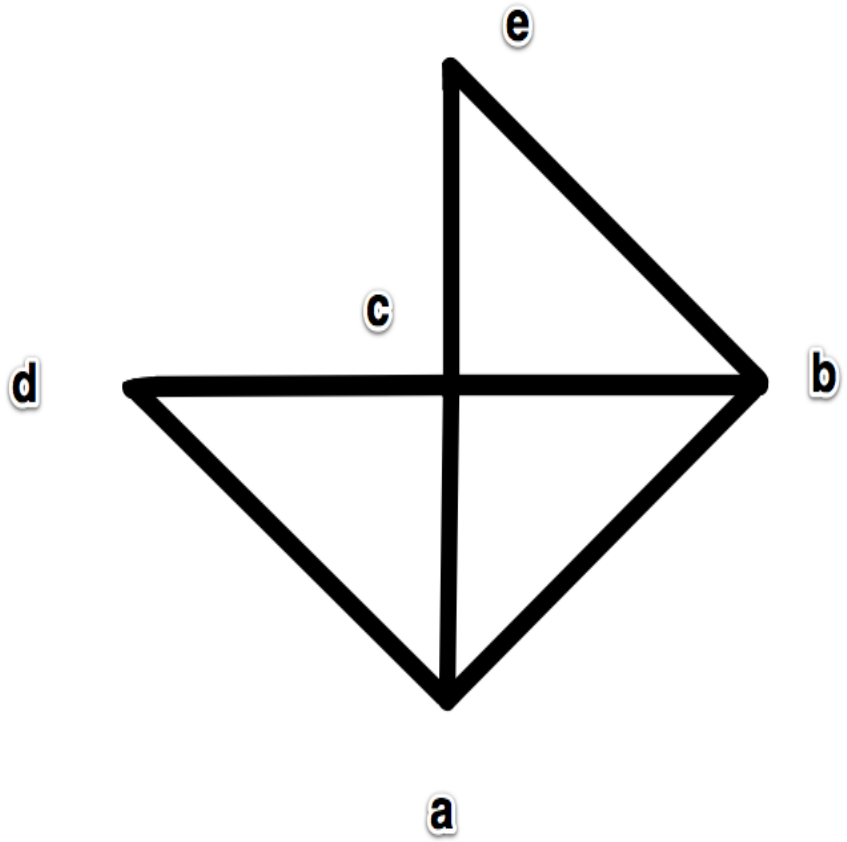
    $\sigma(i) \leftarrow v$

    foreach *unnumbered vertex* w *adjacent to* v **do**

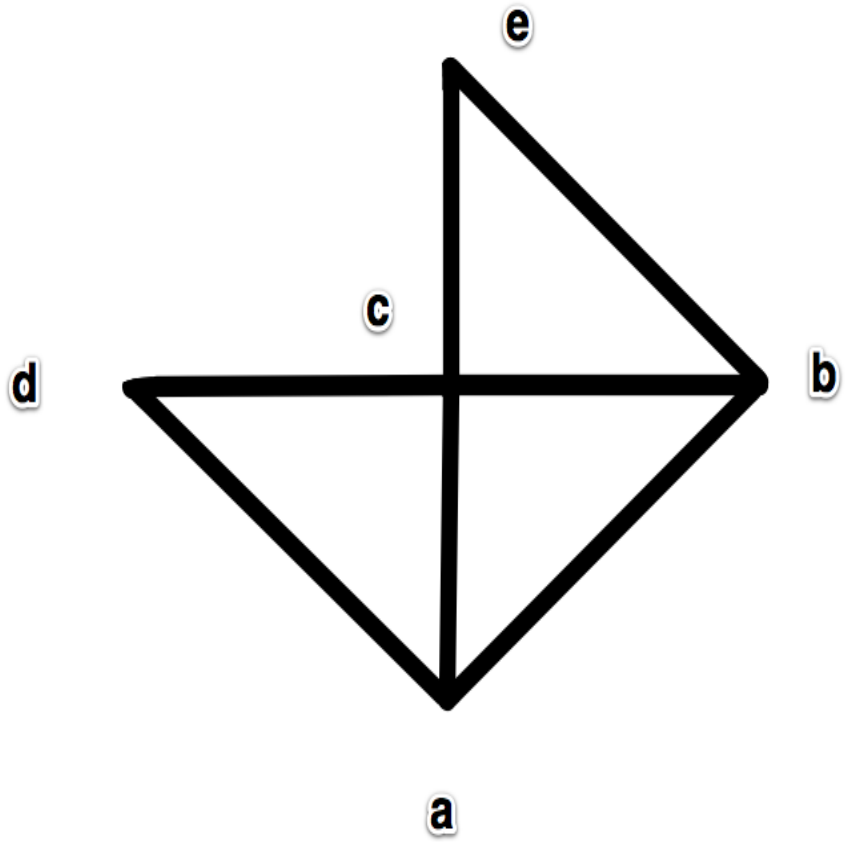
        $label(w) \leftarrow \{i\}.label(w)$

    end

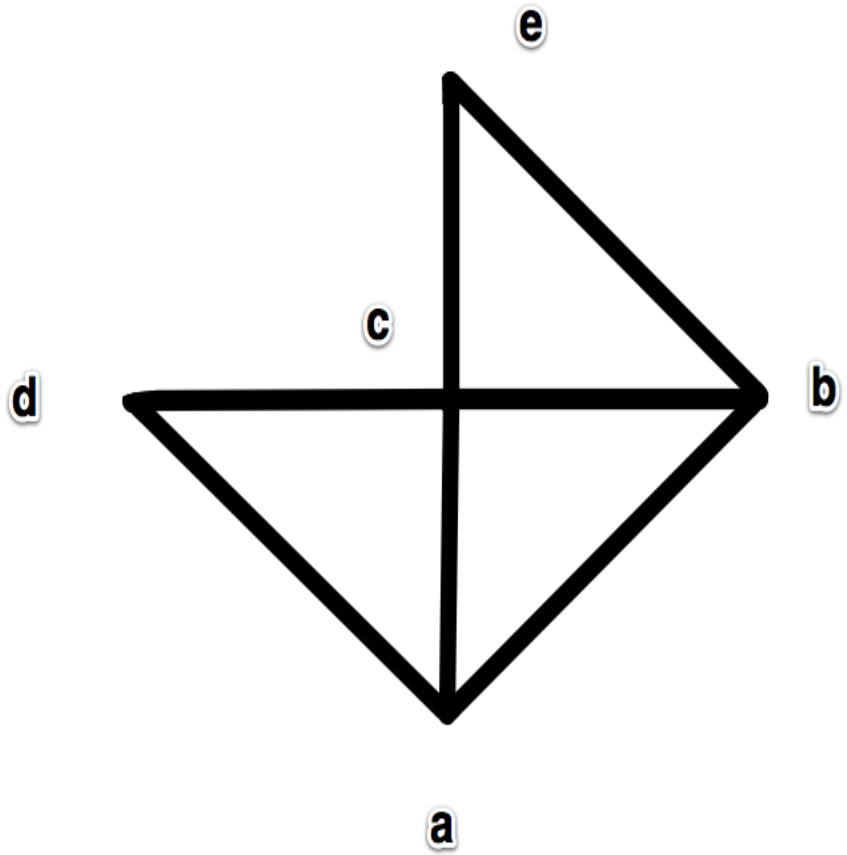
end



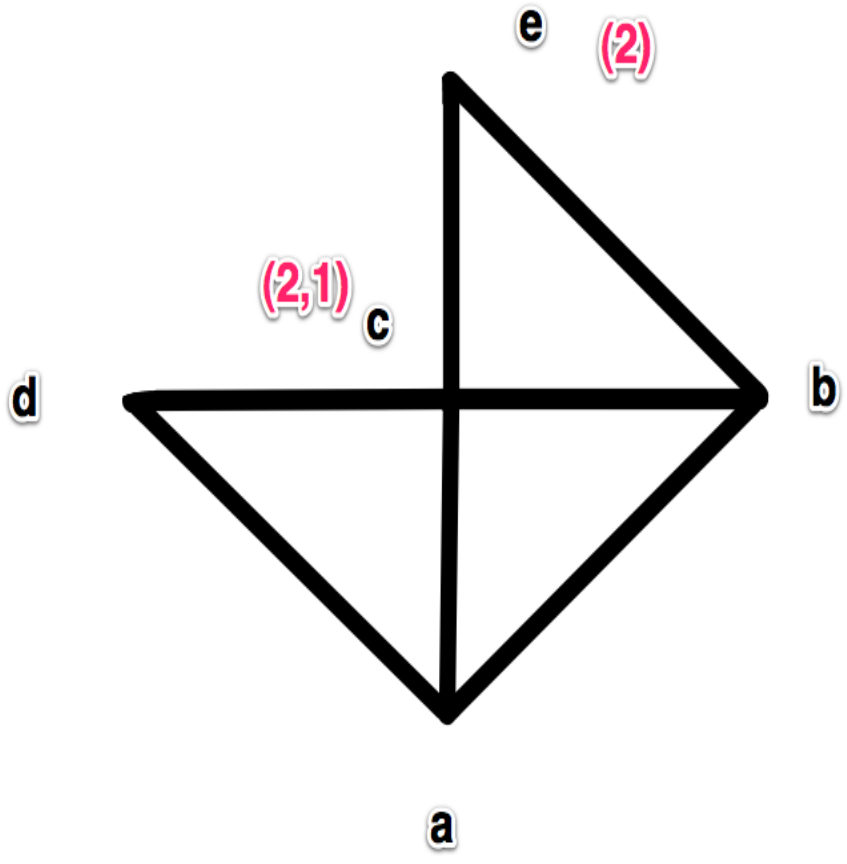
an example for LexDFS



an example for LexDFS

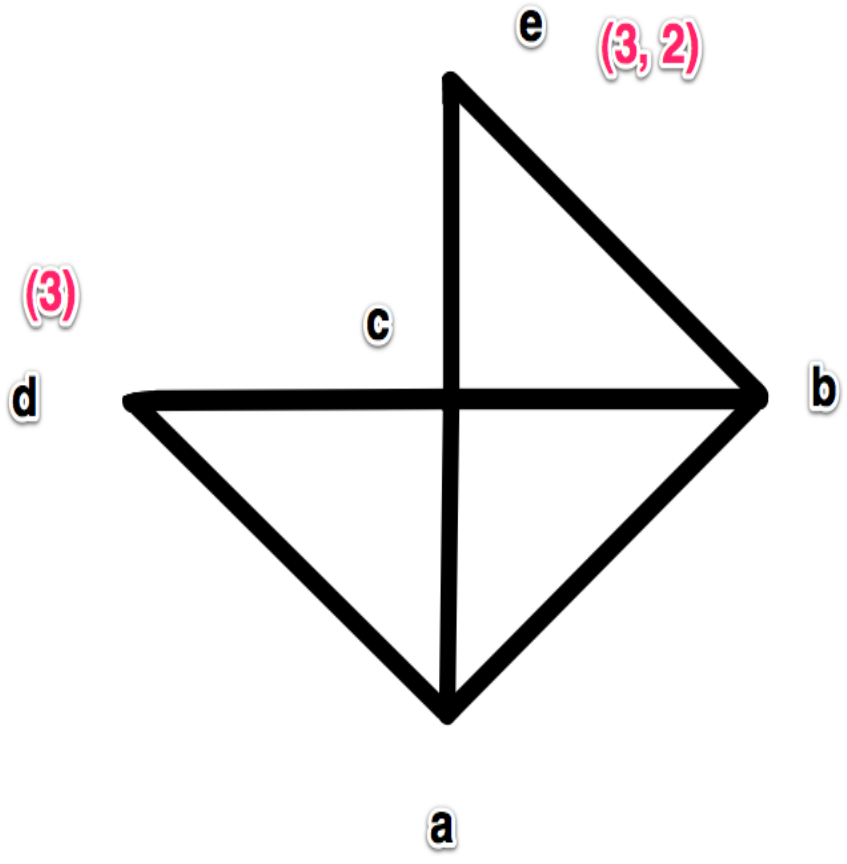


start with a and first visit b



start with a and first visit b

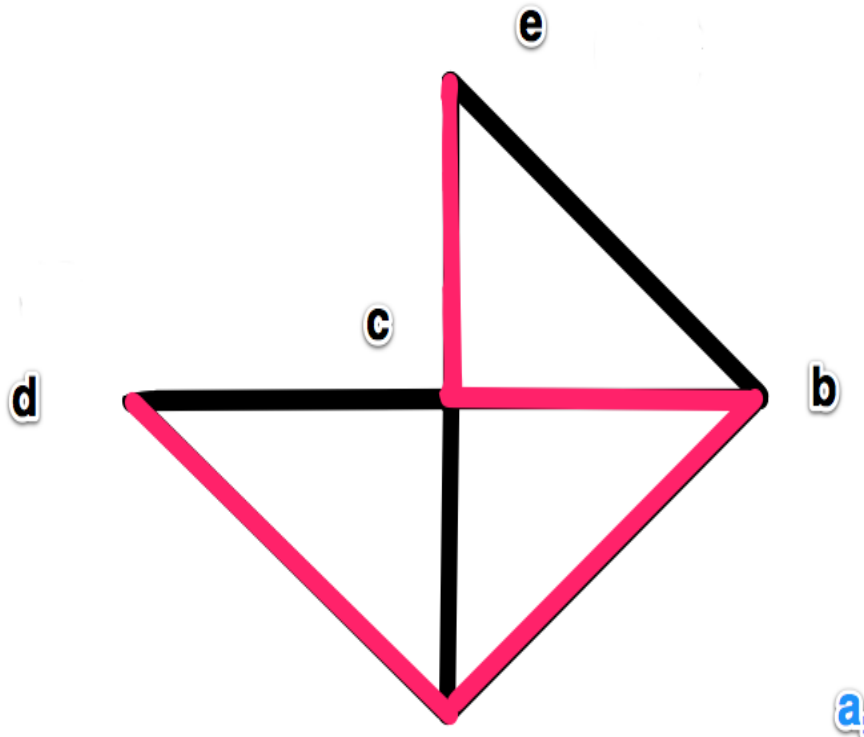
we must choose c next



start with a and first visit b

we must choose c next

then e is next and we finish in d



a, b, c, e, d

start with a and first visit b

we must choose c next

then e is next and we finish in d

LexDFS

Complexity

Is it possible to compute a LexDFS in $O(n+m)$?

LexDFS

Complexity

Is it possible to compute a LexDFS in $O(n+m)$?

Spinrad 2008

Best implementation so far needs $O(n+mloglogn)$ using Van der Boas trees.

LexDFS

Complexity

Is it possible to compute a LexDFS in $O(n+m)$?

Spinrad 2008

Best implementation so far needs $O(n+mloglogn)$ using Van der Boas trees.

First application : D. Corneil, B. Dalton, M. H. 2010

Hamiltonicity on co-comparability graphs via LexDFS.

Applications

- BFS to compute distances, diameter, centers
  Heuristics for diameter

Applications

- BFS to compute distances, diameter, centers
  Heuristics for diameter
- DFS planarity, strongly connected components, 2-SAT, ...

Applications

- BFS to compute distances, diameter, centers
  Heuristics for diameter
- DFS planarity, strongly connected components, 2-SAT, ...
- LexBFS, recognition of chordal graphs, interval graphs ...
  Heuristics for one consecutiveness property

Applications

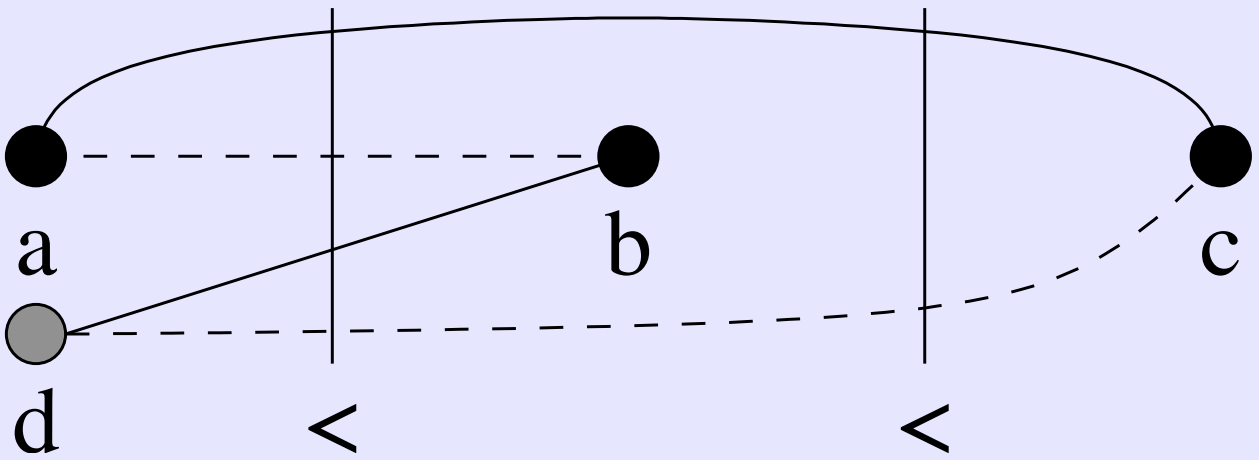
- BFS to compute distances, diameter, centers
  Heuristics for diameter
- DFS planarity, strongly connected components, 2-SAT, ...
- LexBFS, recognition of chordal graphs, interval graphs ...
  Heuristics for one consecutiveness property
- LexDFS, long paths, minimum path cover
  For co-comparability graphs LexDFS computes layered ordering of the complement partial order.
  Heuristics for graph clustering

MNS

Data: a graph $G = (V, E)$ and a start vertex s
Result: an ordering σ of V
$S \leftarrow \{s\}$
for $i \leftarrow 1$ **à** n **do**
    Pick an unumbered vertex $v \in S$ with a label maximal under inclusion
    $\sigma(i) \leftarrow v$
    foreach *unumbered vertex* $w \in N(v)$ **do**
        $label(w) \leftarrow \{i\} \cup label(w)$
    end
end

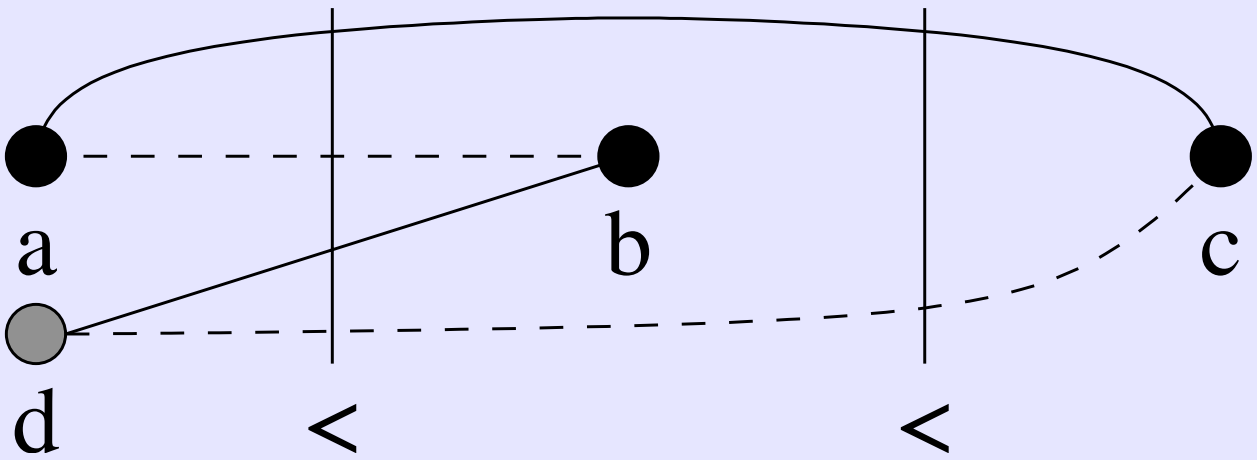
Property (MNS)

For an ordering σ on V, if $a < b < c$ and $ac \in E$ and $ab \notin E$, then it must exist a vertex d such that $d < b$, $db \in E$ et $dc \notin E$.



Property (MNS)

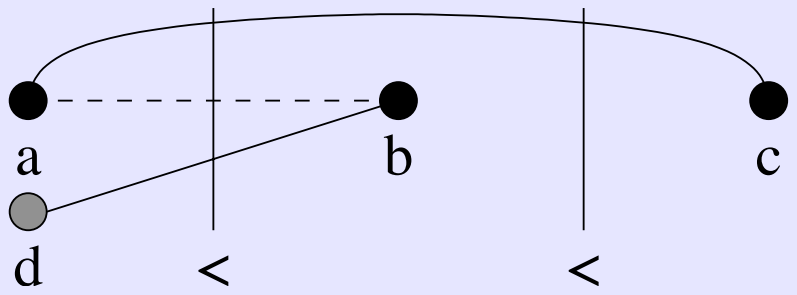
For an ordering σ on V, if $a < b < c$ and $ac \in E$ and $ab \notin E$, then it must exist a vertex d such that $d < b$, $db \in E$ et $dc \notin E$.



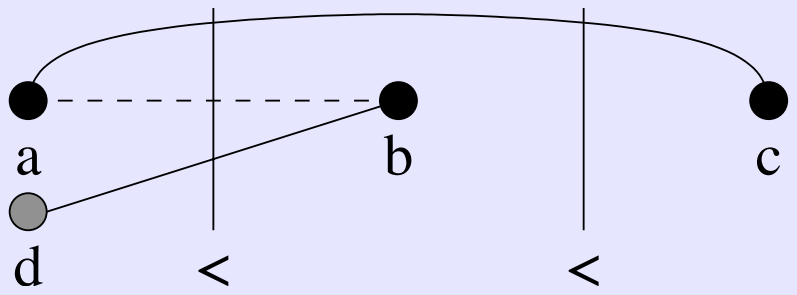
Theorem

For a graph $G = (V, E)$, an ordering σ sur V is a MNS of G iff σ satisfies property MNS.

Back to the generic search



Back to the generic search

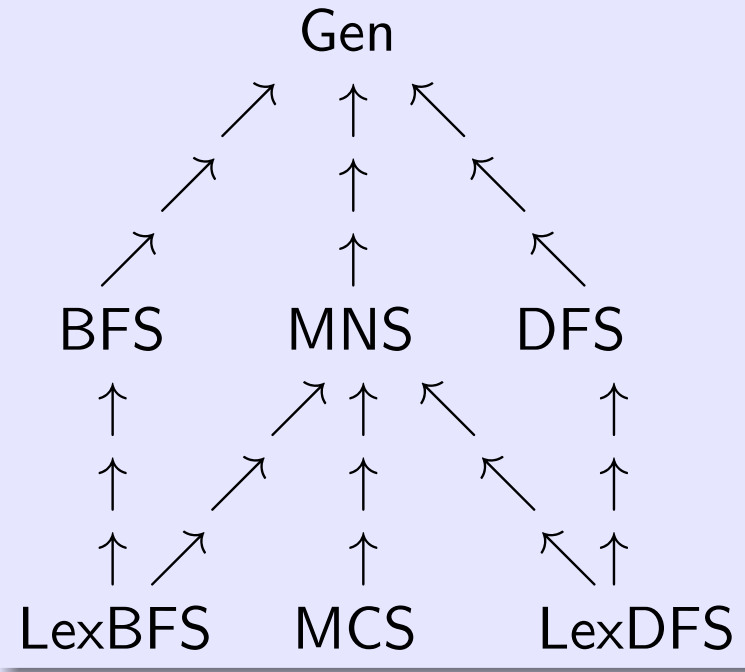


This search is a kind of Lex Generic Search (using analogy between BFS (resp. DFS) and LexBFS (resp. LexDFS). This is why MNS is sometimes named LexGen.

Conclusions

Using the 4-points configurations we have the following inclusion ordering between searches

Inclusions



Search classification

Search	Tie-break management
Generic search	none (random)
BFS	queue
DFS	stack
LexBFS	Lexicographic maximal
LexDFS	Lexicographic maximal
MNS	Maximal under inclusion
MCS	Maximal for the cardinality

Importance of 4 points conditions for graph recognition

Many classes of graphs or partial orders can be characterized by the existence of a particular ordering of the vertices with some forbidden configuration on three points.
Examples with forbidden configuration on three points :

1. Interval graphs : ordering of the left ends of the intervals.

Importance of 4 points conditions for graph recognition

Many classes of graphs or partial orders can be characterized by the existence of a particular ordering of the vertices with some forbidden configuration on three points.
Examples with forbidden configuration on three points :

1. Interval graphs : ordering of the left ends of the intervals.
2. Chordal : simplicial elimination ordering.

Importance of 4 points conditions for graph recognition

Many classes of graphs or partial orders can be characterized by the existence of a particular ordering of the vertices with some forbidden configuration on three points.
Examples with forbidden configuration on three points :

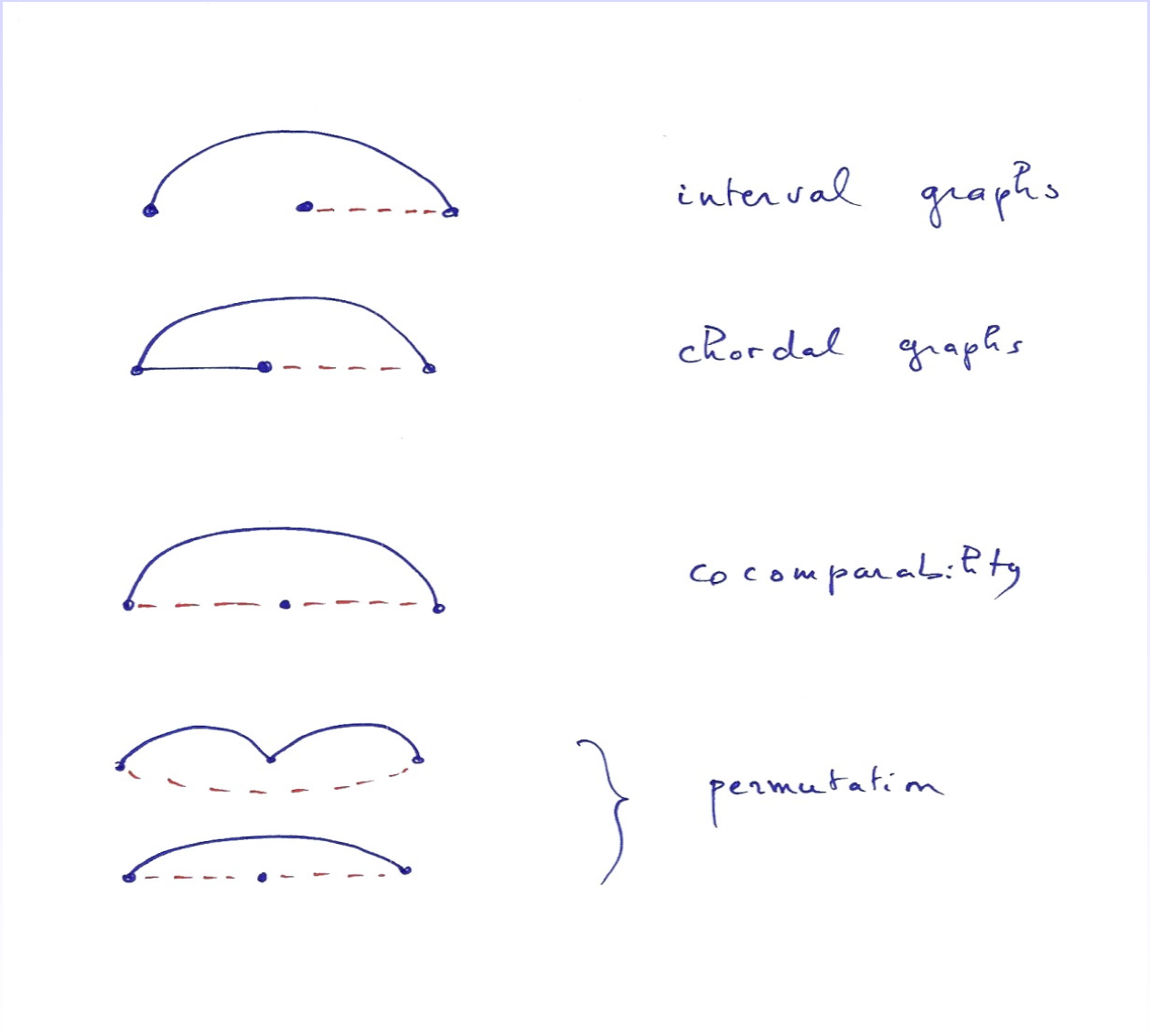
1. Interval graphs : ordering of the left ends of the intervals.
2. Chordal : simplicial elimination ordering.
3. Co-comparability : transitivity violation of the complement graph

Importance of 4 points conditions for graph recognition

Many classes of graphs or partial orders can be characterized by the existence of a particular ordering of the vertices with some forbidden configuration on three points.
Examples with forbidden configuration on three points :

1. Interval graphs : ordering of the left ends of the intervals.
2. Chordal : simplicial elimination ordering.
3. Co-comparability : transitivity violation of the complement graph
4. Permutation : transitivity violation of the graph and its complement.

Forbidden 3 points suborderings



Consequences

LexBFS is involved in many recognition algorithms for these classes of graphs.

- Apply a LexBFS on $\overline{G}$ giving an ordering σ

Consequences

LexBFS is involved in many recognition algorithms for these classes of graphs.

- Apply a LexBFS on $\overline{G}$ giving an ordering σ
- If G is a comparability graph the last vertex of σ, can be taken as a source in a transitive orientation of G.

Consequences

LexBFS is involved in many recognition algorithms for these classes of graphs.

- Apply a LexBFS on $\overline{G}$ giving an ordering σ
- If G is a comparability graph the last vertex of σ, can be taken as a source in a transitive orientation of G.
- The starting point for comparability and permutation graph recognition algorithms.

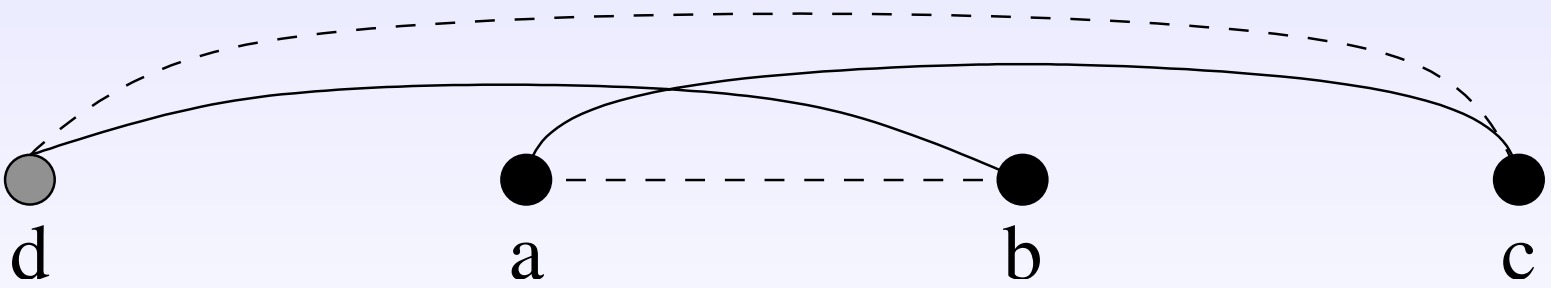
A kind of lego with simple searches

LexBFS

Data: a graph $G = (V, E)$ and a start vertex s
Result: an ordering σ of V
Assign the label $\emptyset$ to all vertices
$label(s) \leftarrow \{n\}$
for $i \leftarrow n$ **à** 1 **do**
    Pick an unumbered vertex v with lexicographically largest label
    $\sigma(i) \leftarrow v$
    foreach *unnumbered vertex w adjacent to v* **do**
        $label(w) \leftarrow label(w).\{i\}$
    end
end

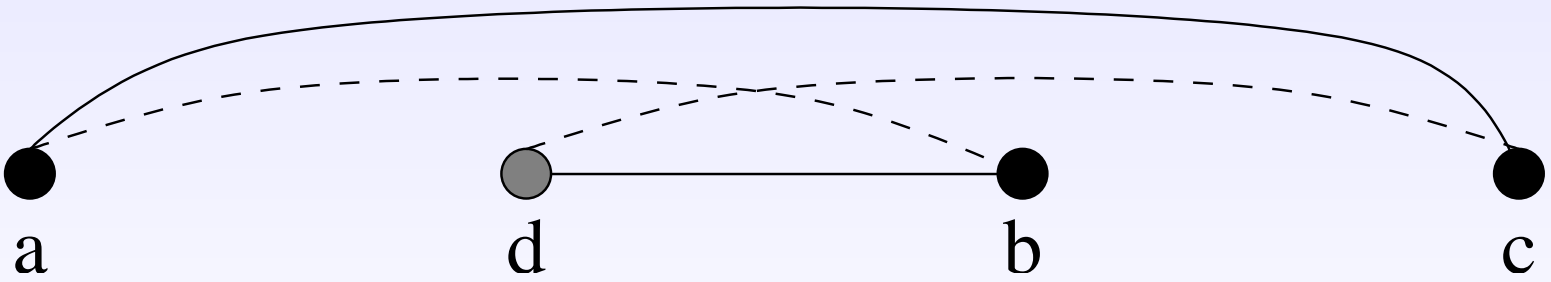
LexBFS

for $i \leftarrow n$ **à** 1 **do**
    Pick a lexicographic max
end
$label(w) \leftarrow label(w).\{i\}$



LexDFS

for $i \leftarrow 1$ **à** n **do**
    Pick a lexicographic max
end
$label(w) \leftarrow \{i\}.label(w)$



Using a remark by Berry, Krueger and Simonet 2011

co-LexBFS

for $i \leftarrow 1$ **à** n **do**
    Pick a lexicographic min
end
$label(w) \leftarrow label(w).\{i\}$

LexBFS on $\overline{G}$

co-LexDFS

for $i \leftarrow 1$ **à** n **do**
    Pick a lexicographic min
end
$label(w) \leftarrow \{i\}.label(w)$

LexDFS on $\overline{G}$

LexUP New !

for $i \leftarrow 1$ **à** n **do**
    Pick a lexicographic max
end
$label(w) \leftarrow label(w).\{i\}$

LexDown New !

for $i \leftarrow n$ **à** 1 **do**
    Pick a lexicographic max
end
$label(w) \leftarrow \{i\}.label(w)$

These two new searches LexUP and LexDown were characterized by Jérémie Dusart (Master MPRI 2010-2011) using forbidden configurations.

Since we focus on the ordering of the vertices as the result of a graph search, now we can compose graph searches in a natural way. Therefore we can denote by $M(G, x_0)$ the order of the vertices obtained by applying M on G starting from the vertex x_0.

Definition of the + Rule

Let M be a graph search and σ an ordering of the vertices of G, $M^+(G, \sigma)$ be the ordering of the vertices obtained by applying M on G starting from the vertex $\sigma(1)$ and tie-breaking using σ in decreasing order.

Why this Rule ?

The + Rule forces to keep the ordering of the previous sweep in case of tie-break

- Graph searches operate on total orderings :
  Step 0 : $\sigma = M(G, x_0)$
  Step 1 : $M(G, \sigma)$
  Step 2 : $M^2(G, \sigma) = M(G, M(G, \sigma))$
  ...
  Step i : $M^i(G, \sigma) = M(G, M^{i-1}(G, \sigma))$
  ...

- Graph searches operate on total orderings :
  Step 0 : $\sigma = M(G, x_0)$
  Step 1 : $M(G, \sigma)$
  Step 2 : $M^2(G, \sigma) = M(G, M(G, \sigma))$
  ...
  Step i : $M^i(G, \sigma) = M(G, M^{i-1}(G, \sigma))$
  ...
- For which search M and graph G does there exist fixed points ?

- Graph searches operate on total orderings :
  Step 0 : $\sigma = M(G, x_0)$
  Step 1 : $M(G, \sigma)$
  Step 2 : $M^2(G, \sigma) = M(G, M(G, \sigma))$
  ...
  Step i : $M^i(G, \sigma) = M(G, M^{i-1}(G, \sigma))$
  ...
- For which search M and graph G does there exist fixed points ?
- Unfortunately a formal study of this composition remains to be done !

- Graph searches operate on total orderings :
  Step 0 : $\sigma = M(G, x_0)$
  Step 1 : $M(G, \sigma)$
  Step 2 : $M^2(G, \sigma) = M(G, M(G, \sigma))$
  ...
  Step i : $M^i(G, \sigma) = M(G, M^{i-1}(G, \sigma))$
  ...
- For which search M and graph G does there exist fixed points ?
- Unfortunately a formal study of this composition remains to be done !
- Also called multisweep algorithms.

1. Such an idea was already used for planarity testing in some algorithm (de Fraysseix and Rosentiehl 1980) with 2 consecutive DFS.

1. Such an idea was already used for planarity testing in some algorithm (de Fraysseix and Rosentiehl 1980) with 2 consecutive DFS.
2. Algorithms for strongly connected components by Kosaraju 1978, Sharir 1981
   In our framework,
   1) $DFS(G)$
   2) $DFS(G^{-}, post^{d})$

1. Such an idea was already used for planarity testing in some algorithm (de Fraysseix and Rosentiehl 1980) with 2 consecutive DFS.
2. Algorithms for strongly connected components by Kosaraju 1978, Sharir 1981
   In our framework,
   1) $DFS(G)$
   2) $DFS(G^{-}, post^{d})$
3. To compute efficiently the diameter of a graph using successive BFS

Linear time recognition algorithms for interval graphs

- Booth and Lueker 1976, using PQ-trees.
- Korte and Mohring 1981 using LexBFS and Modified PQ-trees.
- Hsu and Ma 1995, using modular decomposition and a variation on Maximal Cardinality Search.
- Corneil, Olariu and Stewart SODA 1998, using a series of 5+1 special consecutive LexBFS, published in 2010.
- M.H, McConnell, Paul and Viennot 2000, using LexBFS and partition refinement on maximal cliques (a kind of LexBFS on cliques and minimal separators).
- New ! Peng Li, Yaokun Wu, using 3 +1 special LexBFS, 2011

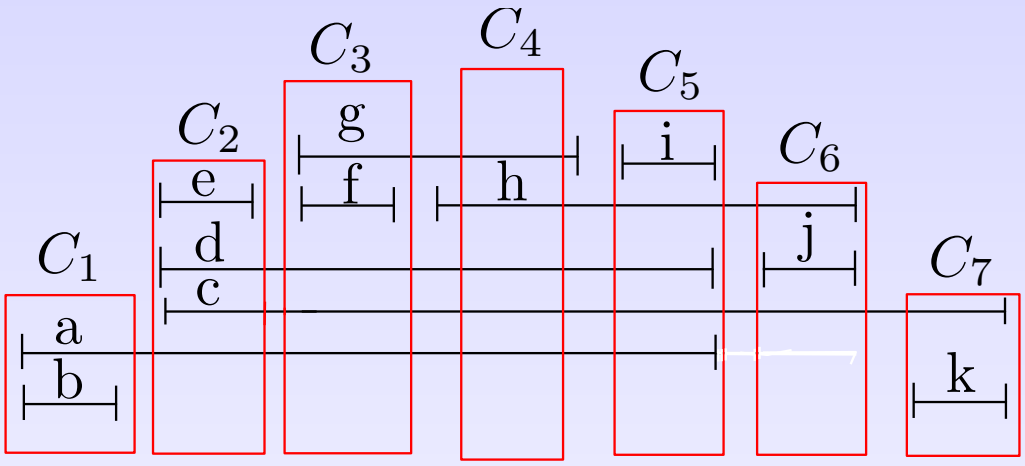
Repeated LexBFS

Require: $G = (V, E)$
Ensure: an ordering σ
  $\sigma \leftarrow$ LexBFS(G)
  for $i = 2$ **to** $|V|$ **do**
    $\sigma \leftarrow$ LexBFS$^+$(G,σ)
  end for

Algorithm 1: LexBFS$^+$ multi-sweep

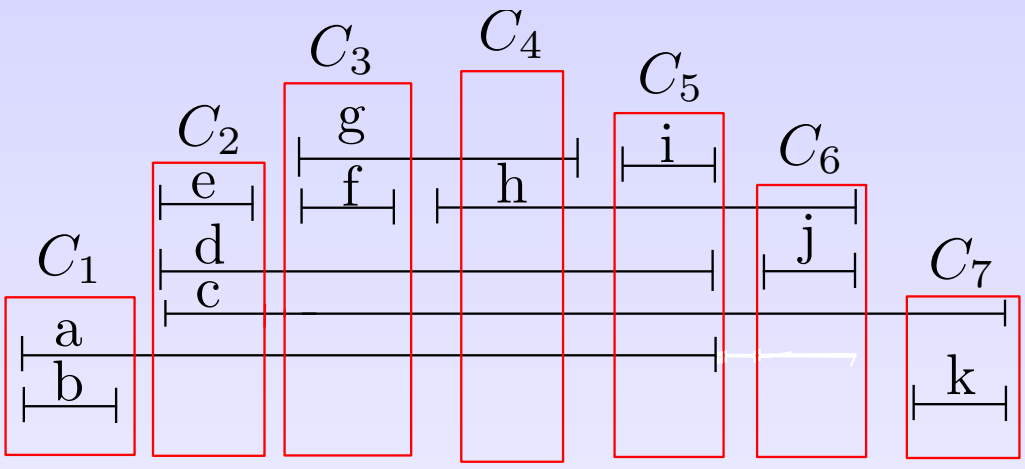
Property

If $G = (V, E)$ is an interval graph, that the previous algorithm finds a fixed point in less than $|V|$ iterations



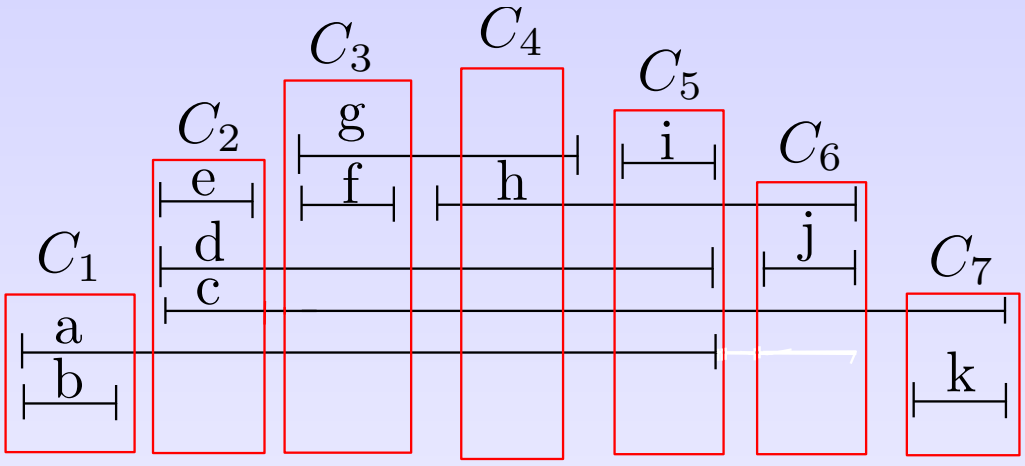
LexBFS(G, c) : c, d, e, a, f, g, h, i, j, k, b

LexBFS(G, c)	C_2 C_3 C_4 C_5 C_6 C_7	C_1



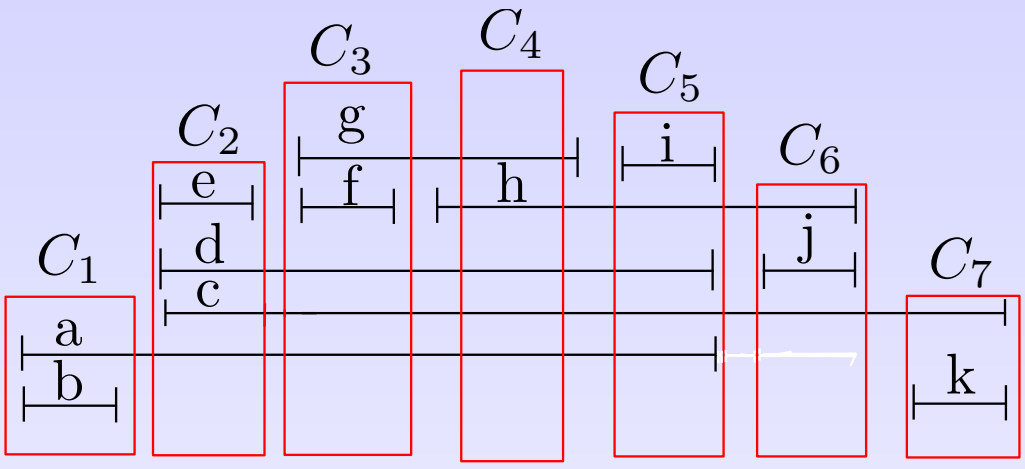
$LexBFS(G, c)$: c, d, e, a, f, g, h, i, j, k, b
$LexBFS^+(LexBFS(G, c))$: b, a, i, h, d, c, g, f, e, j, k

$LexBFS(G, c)$	C_2	C_3	C_4	C_5	C_6	C_7	C_1
$LexBFS^+(LexBFS(G, c))$	C_1	C_5	C_4	C_3	C_2	C_6	C_7



$LexBFS^+(LexBFS(G, c))$: b, a, i, h, d, c, g, f, e, j, k
$LexBFS^{3+}(G, c)$: k, c, j, h, g, d, a, i, f, e, b

$LexBFS(G, c)$	C_2	C_3	C_4	C_5	C_6	C_7	C_1
$LexBFS^{2+}(G, c)$	C_1	C_5	C_4	C_3	C_2	C_6	C_7
$LexBFS^{3+}(G, c)$	C_7	C_6	C_4	C_5	C_3	C_2	C_1



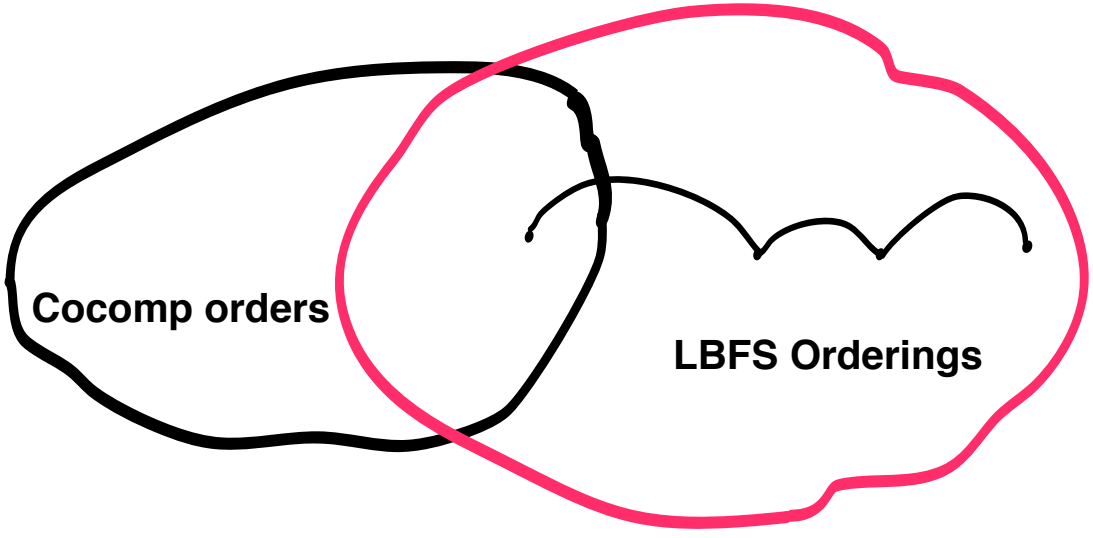
$LexBFS^{3+}(G, c)$: k, c, j, h, g, d, a, i, f, e, b
$LexBFS^{4+}(G, c)$: b, a, e, d, c, f, g, h, i, j, k

$LexBFS(G, c)$	C_2 C_3 C_4 C_5 C_6 C_7						C_1
$LexBFS^{2+}(G, c)$	C_1	C_5 C_4 C_3 C_2				C_6	C_7
$LexBFS^{3+}(G, c)$	C_7	C_6	C_4 C_5		C_3	C_2	C_1
$LexBFS^{4+}(G, c)$	C_1	C_2	C_3	C_4	C_5	C_6	C_7

Generalization to comparability graphs

Question

If $G = (V, E)$ is co-comparability graph, what can be said ?
Is the series $(LBFS^+(G))^i$ periodic (with a period less than $|V|$) ?
i.e. $\exists i < j <\leq n$ such that :
$(LBFS^+(G))^i = (LBFS^+(G))^j$.
Is $(LBFS^+(G))^i$ a cocomp ordering ?



Another way to produce a transitive orientation in linear time

Three applications :

1. Hamiltonicity on co-comparability graphs, D. Corneil, B. Dalton, M.H. 2010
2. New linear extensions for partial orders D. Corneil, M.H., E. Kőhler, 2012
3. Exact diameter computations using a series of BFS, P. Crescenzi, R. Grossi, M.H., L. Lanzi, A. Marino, 2011.

Hamiltonicity on co-comparability graphs

- For a co-comparability graph G find a Minimum Path Cover

Hamiltonicity on co-comparability graphs

- For a co-comparability graph G find a Minimum Path Cover
- Let P be a transitive orientation of $\overline{G}$
  our problem reduce to computing the bump number of P
  (Polynomial algorithm MH, Mohring, Steiner 1988)

Hamiltonicity on co-comparability graphs

- For a co-comparability graph G find a Minimum Path Cover
- Let P be a transitive orientation of $\overline{G}$
  our problem reduce to computing the bump number of P
  (Polynomial algorithm MH, Mohring, Steiner 1988)
- Another equivalent formulation as the 2-machine scheduling problem
  (Another polynomial algorithm Gabow, Tarjan 1985)

Our Algorithm

1. Start with σ any co-comparability ordering of G (a linear extension of P)

Our Algorithm

1. Start with σ any co-comparability ordering of G (a linear extension of P)
2. Apply $LDFS^{+}(\sigma)$ to produce an ordering τ.

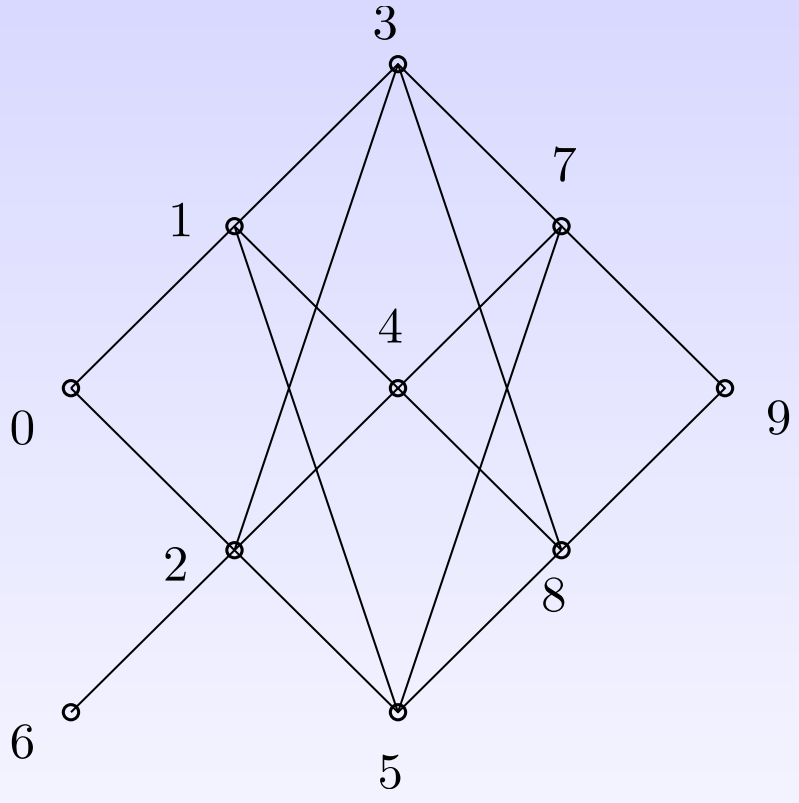
Our Algorithm

1. Start with σ any co-comparability ordering of G (a linear extension of P)
2. Apply $LDFS^{+}(\sigma)$ to produce an ordering τ.
3. Apply $RightMostNeighbour(\tau)$ which gives the path cover

Our Algorithm

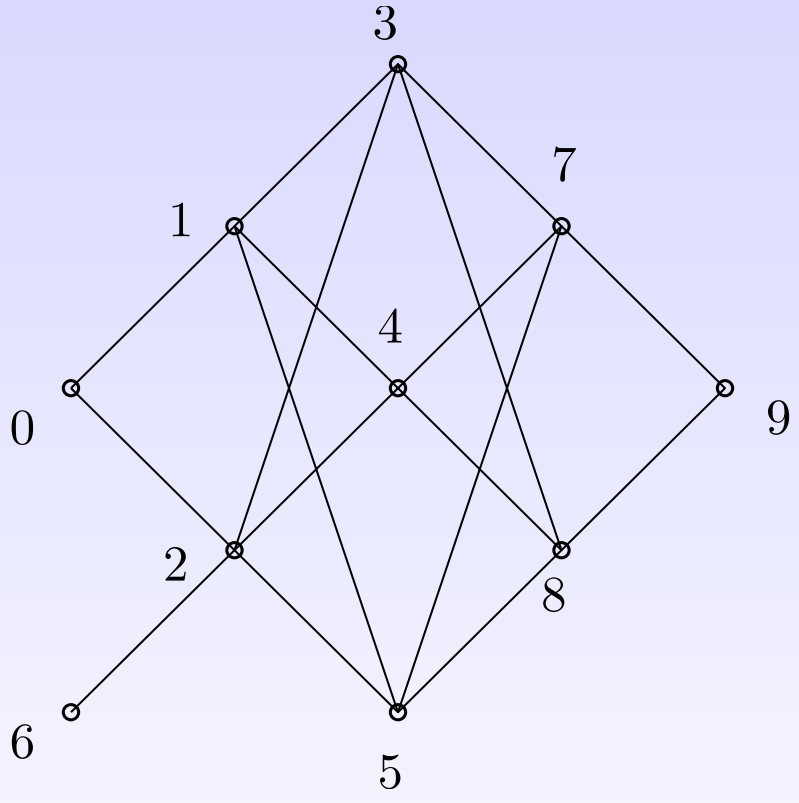
1. Start with σ any co-comparability ordering of G (a linear extension of P)
2. Apply $LDFS^+(\sigma)$ to produce an ordering τ.
3. Apply $RightMostNeighbour(\tau)$ which gives the path cover
4. Exhibit a certificate of minimality with a cut-set.

Let us take an example



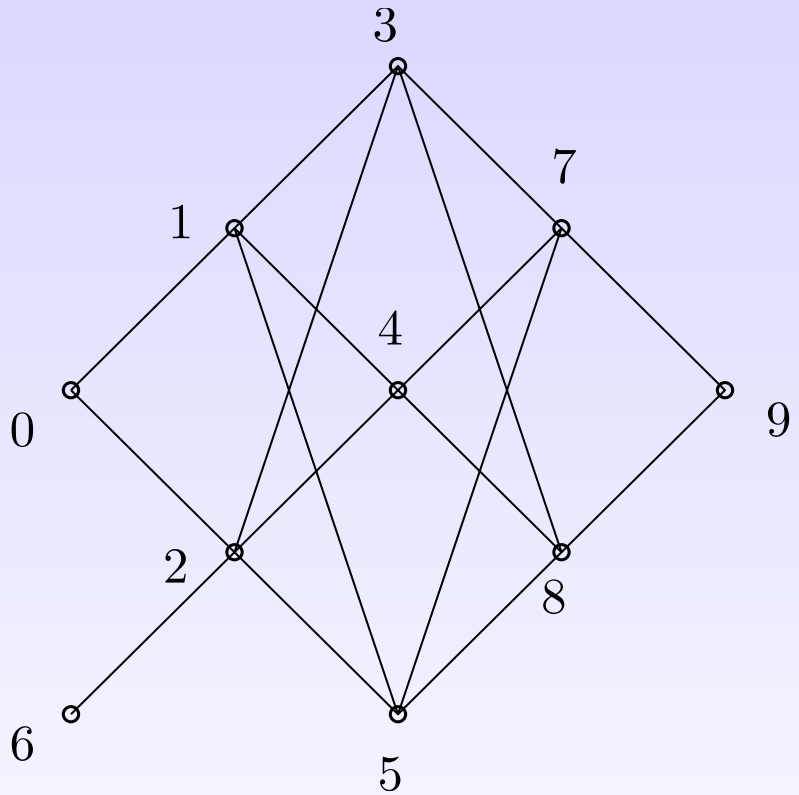
1. $\sigma = 2, 6, 0, 3, 4, 5, 1, 7, 8, 9$ a co-comparability ordering

Let us take an example



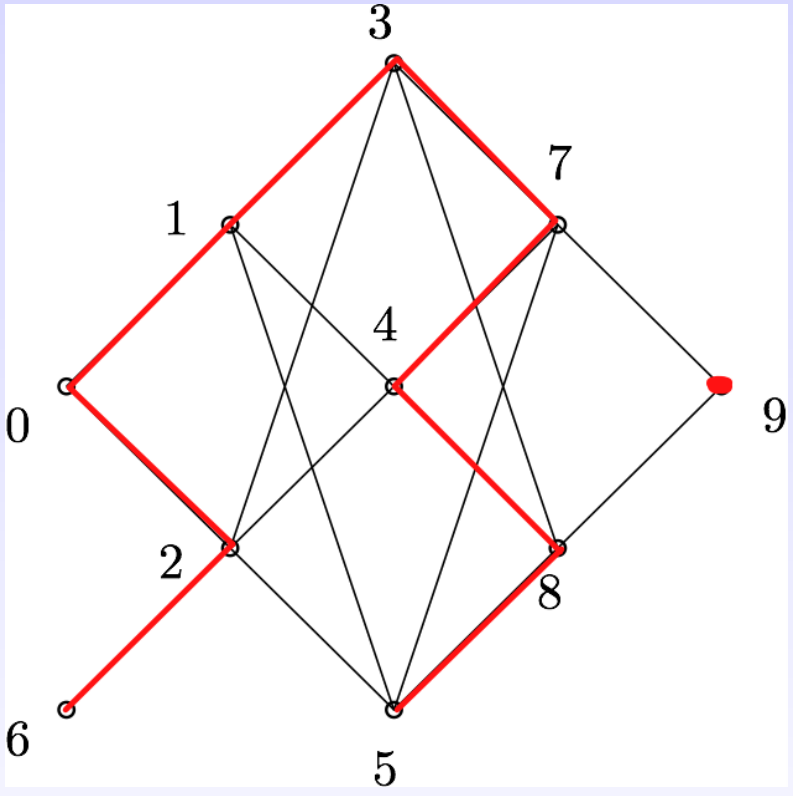
1. $\sigma = 2, 6, 0, 3, 4, 5, 1, 7, 8, 9$ a co-comparability ordering
2. $\tau = \mathit{LexDFS}^{+}(\sigma) = 9, 8, 5, 7, 4, 1, 3, 2, 0, 6$

Let us take an example



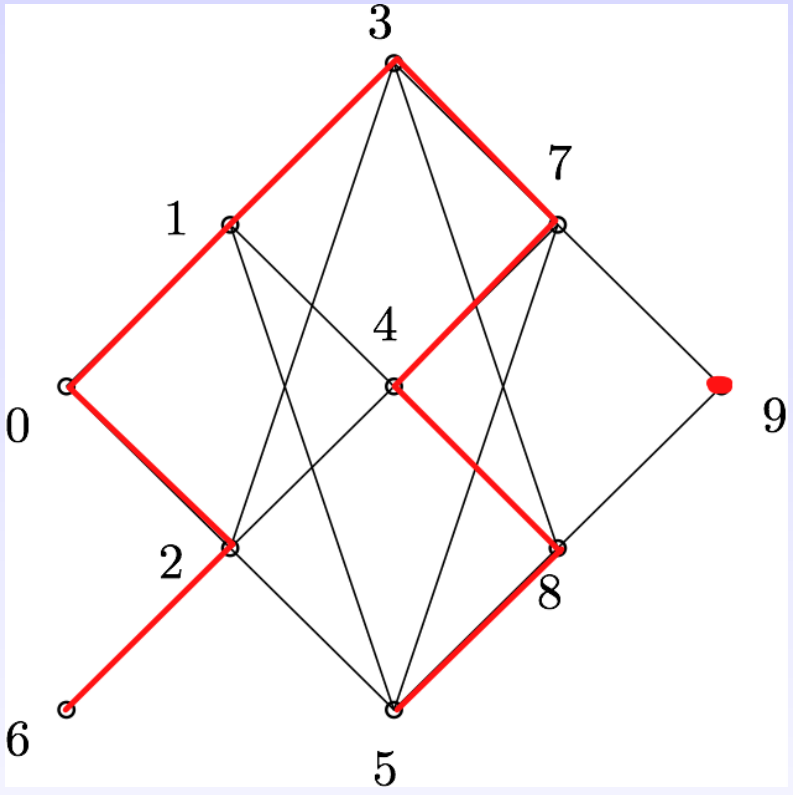
1. $\sigma = 2, 6, 0, 3, 4, 5, 1, 7, 8, 9$ a co-comparability ordering
2. $\tau = \mathit{LexDFS}^{+}(\sigma) = 9, 8, 5, 7, 4, 1, 3, 2, 0, 6$
3. $\mathit{RightMostNeighbour}(\tau) = 6, 2, 0, 1, 3, 7, 4, 8, 5, ||9$

Magic



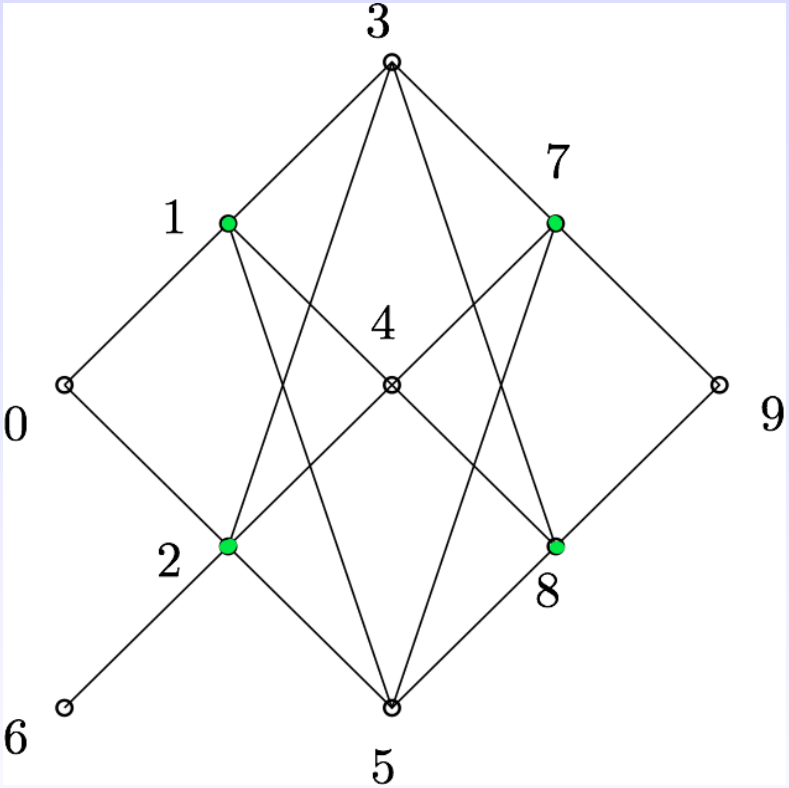
1. $RightMostNeighbour(\tau) = 6, 2, 0, 1, 3, 7, 4, 8, 5, ||9$

Magic



1. $RightMostNeighbour(\tau) = 6, 2, 0, 1, 3, 7, 4, 8, 5, ||9$
2. The cutset $S = \{1, 7, 2, 8\}$ disconnects G into 6 connected components.

Lower bound



As a byproduct we obtain new methods to produce linear extensions

Important Lemma

If σ is a co-comp ordering of G, then $LDFS^+(\sigma)$ is a co-comp ordering of G.

Diameter computations

" While theoretical work on models of computation and methods for analyzing algorithms has had enormous payoffs, we are not done. In many situations, simple algorithms do well. We don't understand why ! Developing means for predicting the performance of algorithms and heuristics on real data and on real computers is a grand challenge in algorithms." Challenges for Theory of Computing by :
CONDON, EDELSBRUNNER, EMERSON, FORTNOW, HABER, KARP, LEIVANT, LIPTON, LYNCH, PARBERRY, PAPADIMITRIOU, RABIN, ROSENBERG, ROYER, SAVAGE, SELMAN, SMITH, TARDOS, AND VITTER, Report for an NSF-sponsored workshop on research in theoretical computer science.

Up to now

At least 2 new searches with no applications yet
2 consecutive LBFS enough to recongize cographs
4-5 consecutive LBFS are enough to capture the structure interval graphs
(similar result for co-comparability graphs) ?

Graph decompositions and applications to networks

1. Failure to use treewith and δ-hyperbolicity for networks.

Graph decompositions and applications to networks

1. Failure to use treewith and δ-hyperbolicity for networks.
2. Negative results also for modularity measures.

Graph decompositions and applications to networks

1. Failure to use treewith and δ-hyperbolicity for networks.
2. Negative results also for modularity measures.
3. We keep this subject as a theoretical tool to analyze networks

Work in progress

1. Extend diameter results to efficient computation of centers (one or all)

Work in progress

1. Extend diameter results to efficient computation of centers (one or all)
2. Understand why the 4-sweep method works so well (analogy with Quicksort)

Work in progress

1. Extend diameter results to efficient computation of centers (one or all)
2. Understand why the 4-sweep method works so well (analogy with Quicksort)
3. Uses a series of sweep for preprocessing for hard combinatorial problems (already used in biological applications of interval graphs)

Work in progress

1. Extend diameter results to efficient computation of centers (one or all)
2. Understand why the 4-sweep method works so well (analogy with Quicksort)
3. Uses a series of sweep for preprocessing for hard combinatorial problems (already used in biological applications of interval graphs)
4. A theory of graph searches : searchoids ? Develop theoretical tools to analyze multisweep algorithms and prove fixed points properties.

Hints for future

1. Develop approximation algorithms even for polynomial problems but applied on huge data

Hints for future

1. Develop approximation algorithms even for polynomial problems but applied on huge data
2. Apply to max flow problems in competition with spectral graph theory à la PageRank

Hints for future

1. Develop approximation algorithms even for polynomial problems but applied on huge data
2. Apply to max flow problems in competition with spectral graph theory à la PageRank
3. Reconsider consecutive ones property and planarity

Hints for future

1. Develop approximation algorithms even for polynomial problems but applied on huge data
2. Apply to max flow problems in competition with spectral graph theory à la PageRank
3. Reconsider consecutive ones property and planarity
4. Community detection in networks (using LexDFS ?)

Biological networks

1. Clustering and graph compression

Biological networks

1. Clustering and graph compression
2. Propagation and recommendation algorithms

Biological networks

1. Clustering and graph compression
2. Propagation and recommendation algorithms
3. Phylogeny on networks (instead of trees)
   perhaps too hard for the next 4 years

Some informal justifications

1. Networking uses some graph theory and algorithms (example OLSR and many other routing protocols, but also P2P with their overlay networks)

Some informal justifications

1. Networking uses some graph theory and algorithms (example OLSR and many other routing protocols, but also P2P with their overlay networks)
2. Our group in distributed computing is very closed to graph algorithms

Some informal justifications

1. Networking uses some graph theory and algorithms (example OLSR and many other routing protocols, but also P2P with their overlay networks)
2. Our group in distributed computing is very closed to graph algorithms
3. Growing importance of social networks in our everyday life (ANR project Algopol, with Orange, EHESS and Linkfluence)

Some informal justifications

1. Networking uses some graph theory and algorithms (example OLSR and many other routing protocols, but also P2P with their overlay networks)
2. Our group in distributed computing is very closed to graph algorithms
3. Growing importance of social networks in our everyday life (ANR project Algopol, with Orange, EHESS and Linkfluence)
4. Clustering and recommendation algorithms to reason with heterogenous sets of data
   (join work with a start-up PaperMind, tremendous potential applications for graph algorithms)

Interdisciplinarity aspects

1. With Biologists (2-years post-doc on a common subject supported by Paris)
2. With sociologists from Orange (Uses of Networks) ANR project.

Thank you for your attention !