Graph Searching is playing with orders

Michel Habib
habib@irif.fr
http://www.irif.fr/~habib

7 novembre 2016

Schedule of this course

Introduction to graph search

Schedule of this course

Introduction to graph search

Application to Tarjan's strongly connected components algorithm

Schedule of this course

Introduction to graph search

Application to Tarjan's strongly connected components algorithm

End vertices

Schedule of this course

Introduction to graph search

Application to Tarjan's strongly connected components algorithm

End vertices

TBLS, a Tie-Breaking Label Search

Schedule of this course

Introduction to graph search

Application to Tarjan's strongly connected components algorithm

End vertices

TBLS, a Tie-Breaking Label Search

Two new searches LEXUP and LEXDOWN with no application

Introduction to graph search

Application to Tarjan's strongly connected components algorithm

End vertices

TBLS, a Tie-Breaking Label Search

Two new searches LEXUP and LEXDOWN with no application

Graph searches are very well known and used in many situations :

1. "Fil d'ariane" in the Greek mythology.

Graph searches are very well known and used in many situations :

1. "Fil d'ariane" in the Greek mythology.
2. Euler (1735) for solving the famous walk problem in Kœnisberg city

Graph searches are very well known and used in many situations :

1. "Fil d'ariane" in the Greek mythology.
2. Euler (1735) for solving the famous walk problem in Kœnisberg city
3. Euler's theorem proved by Hierholzer in 1873.

Graph searches are very well known and used in many situations :

1. "Fil d'ariane" in the Greek mythology.
2. Euler (1735) for solving the famous walk problem in Kœnisberg city
3. Euler's theorem proved by Hierholzer in 1873.
4. Tremaux (1882) and Tarry (1895) introduced DFS to solve maze problems

Graph searches are very well known and used in many situations :

1. "Fil d'ariane" in the Greek mythology.
2. Euler (1735) for solving the famous walk problem in Kœnisberg city
3. Euler's theorem proved by Hierholzer in 1873.
4. Tremaux (1882) and Tarry (1895) introduced DFS to solve maze problems
5. Fleury, proposed a nice algorithm to compute an Euler Tour, 1882

Graph searches are very well known and used in many situations :

1. "Fil d'ariane" in the Greek mythology.
2. Euler (1735) for solving the famous walk problem in Kœnisberg city
3. Euler's theorem proved by Hierholzer in 1873.
4. Tremaux (1882) and Tarry (1895) introduced DFS to solve maze problems
5. Fleury, proposed a nice algorithm to compute an Euler Tour, 1882
6. Computer scientists from 1950, in particular in the 70's, Tarjan for new applications of DFS....

Graph searches are very well known and used in many situations :

1. "Fil d'ariane" in the Greek mythology.
2. Euler (1735) for solving the famous walk problem in Kœnisberg city
3. Euler's theorem proved by Hierholzer in 1873.
4. Tremaux (1882) and Tarry (1895) introduced DFS to solve maze problems
5. Fleury, proposed a nice algorithm to compute an Euler Tour, 1882
6. Computer scientists from 1950, in particular in the 70's, Tarjan for new applications of DFS....
7. 4 points characterizations Corneil, Krueger (2008), and the definition of LDFS a new interesting basic search.

First course of Graph Properties, I know in Paris, at CNAM

Graphs or networks :
A. Sainte-Laguë, *Les réseaux ou graphes*, **Gauthier-Villars**, Paris,1926.

Some definitions

Graph Search

The graph is supposed to be connected so as the set of visited vertices. After choosing an initial vertex, a search of a connected graph visits each of the vertices and edges of the graph such that a new vertex is visited only if it is adjacent to some previously visited vertex.
At any point there may be several vertices that may possibly be visited next. To choose the next vertex we need a tie-break rule. The breadth-first search (BFS) and depth-first search (DFS) algorithms are the traditional strategies for determining the next vertex to visit.

They are many formalisms to represent graph search depending on the view point :

- Data structures involved

They are many formalisms to represent graph search depending on the view point :

- Data structures involved
- Implementation issues

They are many formalisms to represent graph search depending on the view point :

- Data structures involved
- Implementation issues
- Path algebras

They are many formalisms to represent graph search depending on the view point :

- Data structures involved
- Implementation issues
- Path algebras
- Bio-inspired graph searches (as used by Amos Korman with ants)

They are many formalisms to represent graph search depending on the view point :

- Data structures involved
- Implementation issues
- Path algebras
- Bio-inspired graph searches (as used by Amos Korman with ants)
- Reachability problems in complexity theory

They are many formalisms to represent graph search depending on the view point :

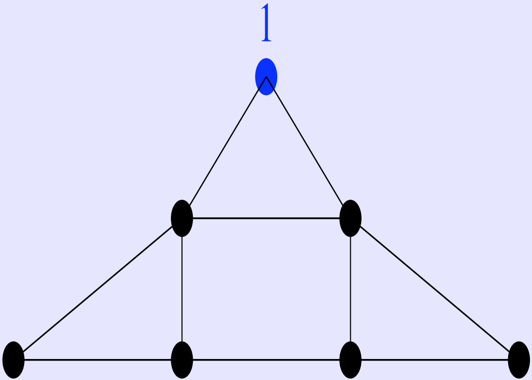
- Data structures involved
- Implementation issues
- Path algebras
- Bio-inspired graph searches (as used by Amos Korman with ants)
- Reachability problems in complexity theory
- Here we want to focus on the visiting ordering of the vertices and on the tie-break process that distinguishes graph searches

1. For us a graph search just produces a total ordering of the vertices.

1. For us a graph search just produces a total ordering of the vertices.
2. In the following an ordering of the vertices, always means a total ordering of the vertices.

1. For us a graph search just produces a total ordering of the vertices.
2. In the following an ordering of the vertices, always means a total ordering of the vertices.
3. Seminal paper with a systematic study of graph search : D.G. Corneil et R. M. Krueger, A unified view of graph searching, SIAM J. Discrete Math, 22, Num 4 (2008) 1259-1276

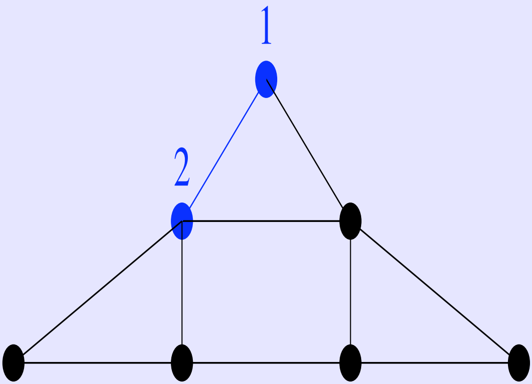
Generic Search



Invariant

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

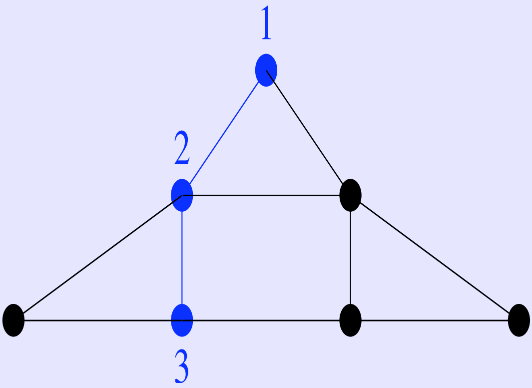
Generic Search



Invariant

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

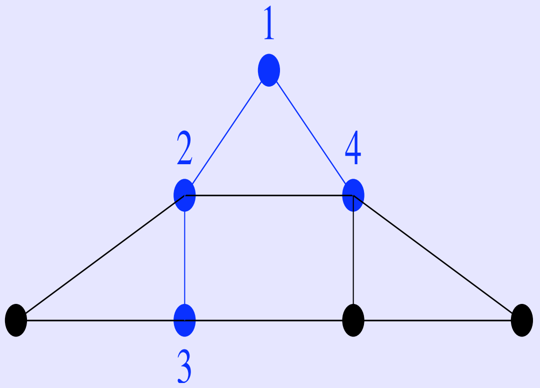
Generic Search



Invariant

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

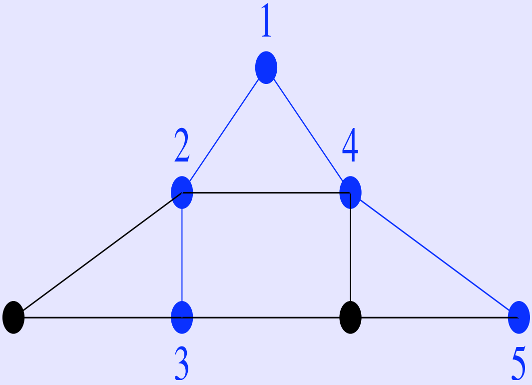
Generic Search



Invariant

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

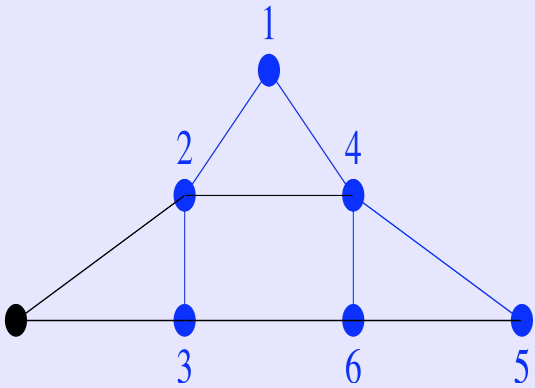
Generic Search



Invariant

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

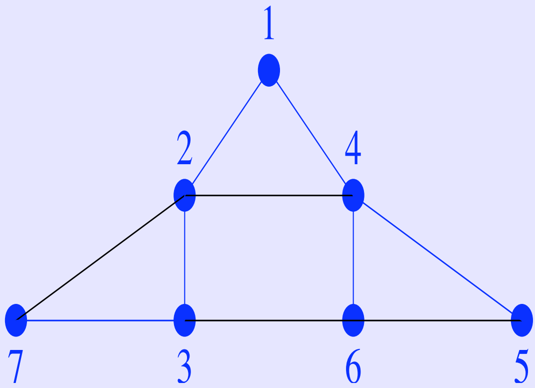
Generic Search



Invariant

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

Generic Search



Invariant

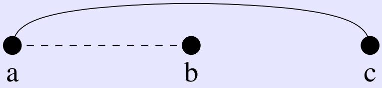
At each step, an edge between a visited vertex and a unvisited one is selected

Generic search

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

Generic question ?

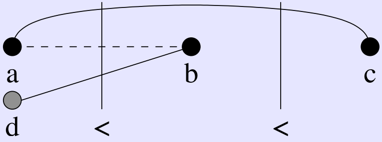
Let a, b et c be 3 vertices such that $ab \notin E$ et $ac \in E$.



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

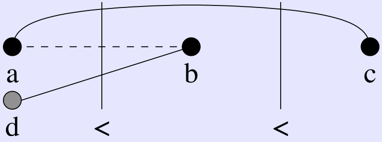
Property (Generic)

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



Property (Generic)

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



Theorem

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

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

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
- DFS (Stack), BFS(Queue), Dijkstra (Heap), ...

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
- DFS (Stack), BFS(Queue), Dijkstra (Heap), . . .
- Graph searches mainly differ by the management of the tie-break set

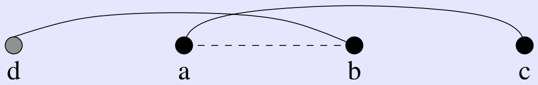
BFS

Data: a graph $G = (V, E)$ and a start vertex $s \in V$
Result: an ordering σ of V
Initialize *queue* to $\{s\}$
for $i \leftarrow 1$ **à** n **do**
    dequeue v from beginning of *queue*
    $\sigma(i) \leftarrow v$
    foreach *unnumbered vertex* $w \in N(v)$ **do**
        if *w is not already in queue* **then**
            enqueue w to the end of *queue*
        end
    end
end

Algorithm 1: Breadth First Search (BFS)

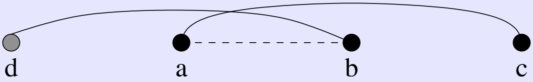
Property (BFS)

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



Property (BFS)

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



Theorem

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

Applications of BFS

1. Distance computations (unit length), diameter and centers, (see course # 2)

Applications of BFS

1. Distance computations (unit length), diameter and centers, (see course # 2)
2. BFS provides a useful layered structure of the graph

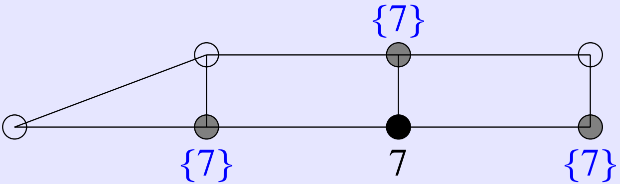
Applications of BFS

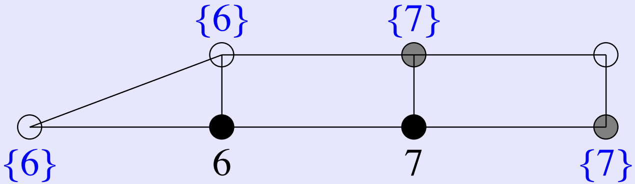
1. Distance computations (unit length), diameter and centers, (see course # 2)
2. BFS provides a useful layered structure of the graph
3. Using BFS to search an augmenting path provides a polynomial implementation of Ford-Fulkerson maximum flow algorithm.

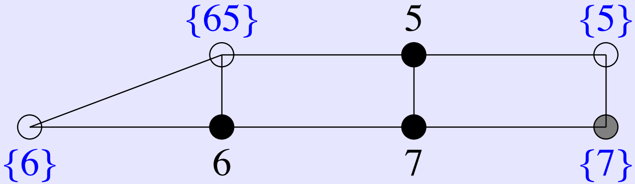
Lexicographic Breadth First Search (LBFS)

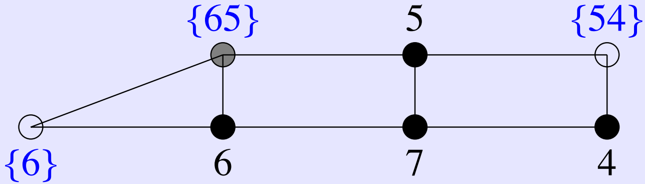
Data: a graph $G = (V, E)$ and a start vertex s
Result: an ordering σ of V
Assign the label $\emptyset$ to all vertices
$label(s) \leftarrow \{n\}$
for $i \leftarrow n$ **à** 1 **do**
  Pick an unnumbered vertex v with lexicographically largest label
  $\sigma(i) \leftarrow v$
  foreach *unnumbered vertex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

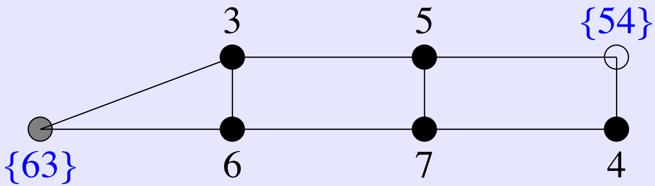
Algorithm 2: LBFS

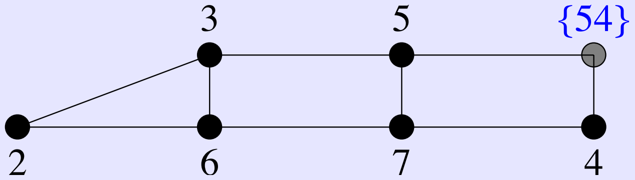

{7}
{7}
7
{7}

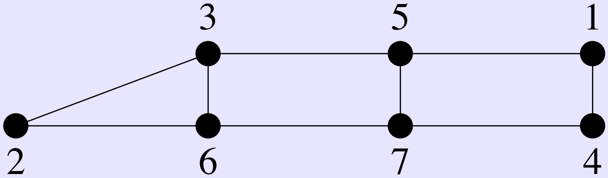

{6}
{7}
{6}
6
7
{7}


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


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


3
5
{54}
{63}
6
7
4

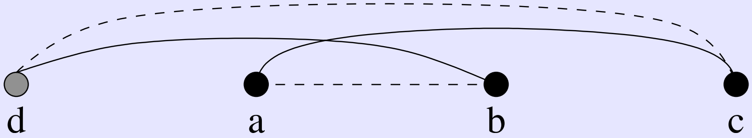

3
5
{54}
2
6
7
4


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 LBFS explores the vertices.

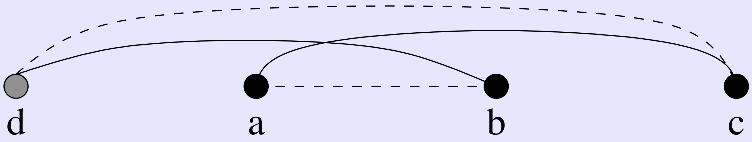
Property (LexB)

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



Property (LexB)

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



Theorem

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

Why LBFS behaves so nicely on well-structured graphs

A nice recursive property

On every tie-break set S, LBFS operates on $G(S)$ as a LBFS.

Why LBFS behaves so nicely on well-structured graphs

A nice recursive property

On every tie-break set S, LBFS operates on $G(S)$ as a LBFS.

proof

Consider $a, b, c \in S$ such that $a <_\sigma b <_\sigma c$ and $ac \in E$ and $ab \notin E$, then it must exist a vertex d such that $d <_\sigma a$ et $db \in E$ et $dc \notin E$. But then necessarily $d \in S$.

Why LBFS behaves so nicely on well-structured graphs

A nice recursive property

On every tie-break set S, LBFS operates on $G(S)$ as a LBFS.

proof

Consider $a, b, c \in S$ such that $a <_\sigma b <_\sigma c$ and $ac \in E$ and $ab \notin E$, then it must exist a vertex d such that $d <_\sigma a$ et $db \in E$ et $dc \notin E$. But then necessarily $d \in S$.

Remark

Analogous properties are false for other classical searches.

LexBFS versus LBFS !

Google Images query : LBFS (thanks to Fabien)

yields :

LexBFS versus LBFS !

Google Images query : LBFS (thanks to Fabien)

yields :

One of the First Answer

Applications of LBFS

1. Most famous one : chordal graph recognition via simplicial elimination schemes (easy application of the 4-points condition)

Applications of LBFS

1. Most famous one : chordal graph recognition via simplicial elimination schemes (easy application of the 4-points condition)
2. For many classes of graphs using LBFS ordering "backward" provides structural information on the graph.

Applications of LBFS

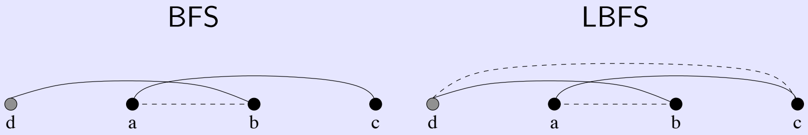
1. Most famous one : chordal graph recognition via simplicial elimination schemes (easy application of the 4-points condition)
2. For many classes of graphs using LBFS ordering "backward" provides structural information on the graph.
3. Last visited vertex (or clique) has some property (example simplicial for chordal graph)

Applications of LBFS

1. Most famous one : chordal graph recognition via simplicial elimination schemes (easy application of the 4-points condition)
2. For many classes of graphs using LBFS ordering "backward" provides structural information on the graph.
3. Last visited vertex (or clique) has some property (example simplicial for chordal graph)
4. Of course property LexB was known by authors such as Tarjan or Golumbic to study chordal graphs but they did not noticed that it was a characterization of LBFS.

LDFS

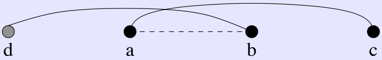
BFS vs LBFS



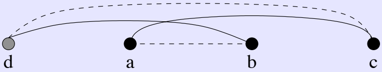
LDFS

BFS vs LBFS

BFS

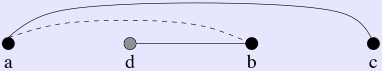


LBFS



DFS vs LDFS

DFS

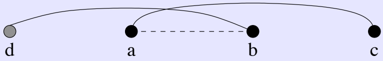


LDFS

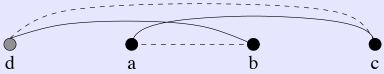
LDFS

BFS vs LBFS

BFS

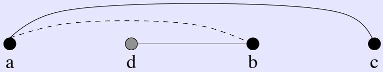


LBFS

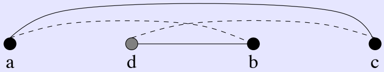


DFS vs LDFS

DFS

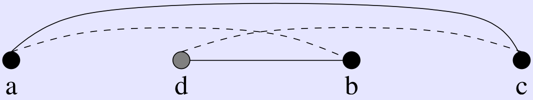


LDFS



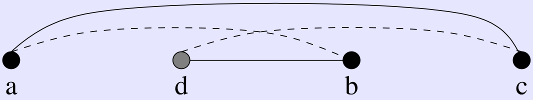
Property (LD)

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



Property (LD)

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



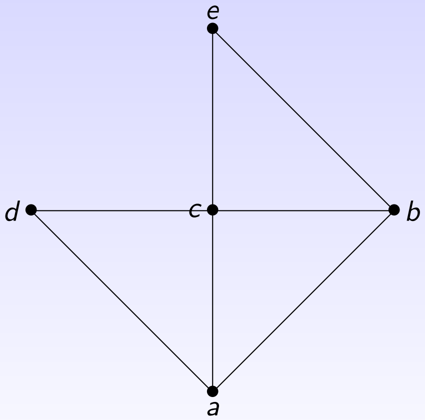
Theorem

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

Lexicographic Depth First Search (LDFS)

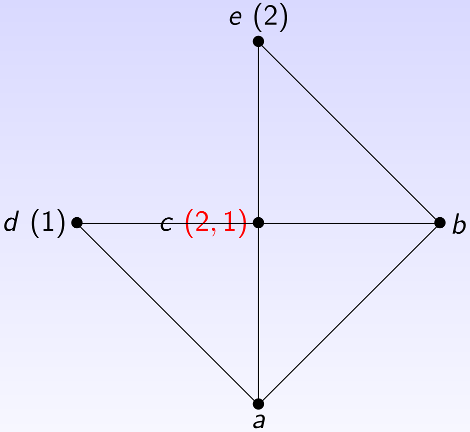
Data: a graph $G = (V, E)$ and a start vertex s
Result: an ordering σ of V
Assign the label $\emptyset$ to all vertices
$label(s) \leftarrow \{0\}$
for $i \leftarrow 1$ **à** n **do**
  Pick an unnumbered vertex v with lexicographically largest label
  $\sigma(i) \leftarrow v$
  foreach *unnumbered vertex* w *adjacent to* v **do**
    $label(w) \leftarrow \{i\}.label(w)$
  end
end

LDFS example



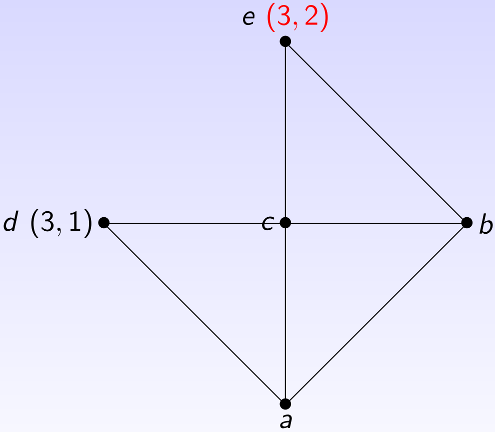
LDFS visiting a then b

LDFS example II



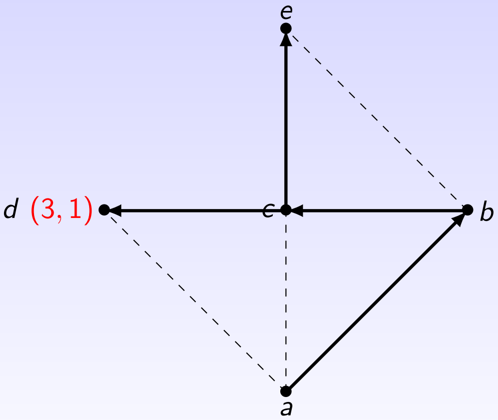
LDFS visiting a then b must visit c

LDFS example III



LDFS visiting a, b, c must visit e

LDFS example IV



LDFS visiting a, b, c, e and must finish in d

Applications of LDFS

- Hard to find application of this new tool !

Applications of LDFS

- Hard to find application of this new tool !
- Finding long paths, a very simple greedy algorithm :

Applications of LDFS

- Hard to find application of this new tool !
- Finding long paths, a very simple greedy algorithm :
- *LDFS-based certifying algorithm for the Minimum Path Cover problem on cocomparability graphs*
  D. Corneil, B. Dalton and M. Habib **SIAM J. of Computing** 42(3) : 792-807 (2013).

Applications of LDFS

- Hard to find application of this new tool !
- Finding long paths, a very simple greedy algorithm :
- *LDFS-based certifying algorithm for the Minimum Path Cover problem on cocomparability graphs*
  D. Corneil, B. Dalton and M. Habib **SIAM J. of Computing** 42(3) : 792-807 (2013).
- Nice graph based heuristics for community detection, joint work with J. Creusefond (PhD in Caen).

- So far we have considered visiting orderings of the vertices which characterize graph searches such as Generic Search, BFS, DFS, LBFS, LDFS.

- So far we have considered visiting orderings of the vertices which characterize graph searches such as Generic Search, BFS, DFS, LBFS, LDFS.
- These orderings allow to prove properties on graph searches (without considering the algorithm itself)

- So far we have considered visiting orderings of the vertices which characterize graph searches such as Generic Search, BFS, DFS, LBFS, LDFS.
- These orderings allow to prove properties on graph searches (without considering the algorithm itself)
- But also to certify (as for example for BFS and diameter computations)

- So far we have considered visiting orderings of the vertices which characterize graph searches such as Generic Search, BFS, DFS, LBFS, LDFS.
- These orderings allow to prove properties on graph searches (without considering the algorithm itself)
- But also to certify (as for example for BFS and diameter computations)
- Let us now go a little further with tie-breaking.

Application to Tarjan's strongly connected components algorithm

The following lemma captures the recursivity of DFS.

The Factor lemma J. Dusart 2014

Let σ be a DFS-ordering of a directed graph G. Let μ be a factor of σ, then μ is a legitimate DFS-ordering of the induced subgraph $G(\mu)$.

Application to Tarjan's strongly connected components algorithm

The following lemma captures the recursivity of DFS.

The Factor lemma J. Dusart 2014

Let σ be a DFS-ordering of a directed graph G. Let μ be a factor of σ, then μ is a legitimate DFS-ordering of the induced subgraph $G(\mu)$.

Proof

Let us consider a triple of vertices (a, b, c) in $G(\mu)$ such that : $ac \in A(G)$ and $ab \notin A(G)$. Using the DFS 4-points conditions it exists necessarily some vertex d between a and b such that $db \in A(G)$. Since d is between a and b in σ, and μ a factor, necessarily $d \in G(\mu)$.

DFS(G**)**;
Data: A directed graph G
Result: a DFS-ordering of the vertices σ and the lists of strongly connected components of G

$i \leftarrow 1$;
$Result \leftarrow \emptyset$;
foreach $x \in V(G)$ **do**
    $Closed(x) \leftarrow False$; $Stack(x) = False$
end
foreach $x \in V(G)$ **do**
    if $Closed(x) = False$ **then**
        $Explore(G, x)$
    end
end

Algorithm 3: The Tarjan's Strongly Connected Components

Explore(G, x);
$Push(x, Result)$; $Stack(x) = True$; $Closed(x) \leftarrow True$;
$\sigma(i) \leftarrow x$; $root(x) \leftarrow i$; $i \leftarrow i + 1$;
foreach $xy \in A(G)$ **do**
    if $Closed(y) = False$ **then**
        $Explore(G, y)$; $root(x) \leftarrow min\{root(x), root(y)\}$;
    end
    else
        if $Stack(y) = True$ **then**
            $root(x) \leftarrow min\{root(x), root(y)\}$
        end
    end
end
if $root(x) = \sigma^{-1}(x)$ **then**
    Pop Result until x included, print these vertices as a list and update their value in the array Stack to false ;
end

In the above algorithm : Result is a stack, Stack is a boolean array describing if a vertex belongs to Result. σ is the DFS-ordering yielded by this DFS search.

Definition 1

For an ordering σ of the vertices of G, a **flyer** is $xy \in A(G)$ such that there exists $z \in V(G)$ with $x <_\sigma z <_\sigma y$.

In the above algorithm : Result is a stack, Stack is a boolean array describing if a vertex belongs to Result. σ is the DFS-ordering yielded by this DFS search.

Definition 1

For an ordering σ of the vertices of G, a **flyer** is $xy \in A(G)$ such that there exists $z \in V(G)$ with $x <_\sigma z <_\sigma y$.

Definition 2

A vertex x is called a **root** if during the execution of the algorithm, when the work is finished at x (i.e. at the end of $Explore(G, x)$), $root(x) = \sigma^{-1}(x)$.

In the above algorithm : Result is a stack, Stack is a boolean array describing if a vertex belongs to Result. σ is the DFS-ordering yielded by this DFS search.

Definition 1

For an ordering σ of the vertices of G, a **flyer** is $xy \in A(G)$ such that there exists $z \in V(G)$ with $x <_\sigma z <_\sigma y$.

Definition 2

A vertex x is called a **root** if during the execution of the algorithm, when the work is finished at x (i.e. at the end of $Explore(G, x)$), $root(x) = \sigma^{-1}(x)$.

Theorem

Tarjan's algorithm applied on a directed graph G computes its strongly connected components.

The proof

It goes by induction on the size of G. If G is reduced to a vertex z, the stack *Result* contains z which is by default a strongly connected component.
The proof will rely on σ the DFS-ordering generated by the execution of Tarjan's algorithm on a graph G.
Let z be the last root vertex in σ. It always exists at least one such vertex, since the first vertex of the DFS is necessarily a root. Let us denote by $D(z)$ the set of descendants of z after z in σ.

- **Claim 1** $G(z \cup D(z))$ is strongly connected.
  It suffices to prove that for every vertex $y \in D(z)$ there exists a path form y to z.
  Since y is not a root it admits a successor t previously considered in σ. If $z <_\sigma t$ we apply the same reasoning on t. Else $t <_\sigma z$ implies that z cannot be root. So we construct a path from y that must necessarily end in z.

Claim 2 $G(z \cup D(z))$ is maximal because it cannot be extended in σ in both directions.

- **Claim 3** $z \cup D(z)$ is a factor of σ.
  Else let us consider the closest to z, vertex b such that : $z <_\sigma b <_\sigma t$, with $t \in D(z)$, $b \notin D(z)$. Let us consider the smallest flyer across b. Such an arch is an arc ac with $a, c \in D(z)$ and $a <_\sigma b <_\sigma c$. using the DFS 4 points condition it exists d in between a and b in σ, such that : $db \in A(G)$. d cannot belong to $D(z)$ else b would also belong to $D(z)$. But then b is not the closest to z, a contradiction.
  If z is the first element of σ then $z \cup D(z) = V(G)$ and G is strongly connected, and all vertices belong to the stack *Result*, and therefore Tarjan's algorithm finds the right solution in this case.
  Else we can apply the factor Lemma , since $z \cup D(z)$ is a factor of σ, by induction the algorithm works on $G(z \cup D(z))$.

Let σ' the restriction of σ to $V(G)$-$z \cup D(z)$. It suffices to prove that σ' is a legitimate DFS on this graph denoted by G'.
Consider a triple $(a, b, c) \in G'$ with $a <_\sigma b <_\sigma c$, $ac \in A(G)$ and $ab \notin A(G)$. Using the DFS 4 points condition on G, it exists d in between a and b in σ, such that : $db \in A(G)$. Let us consider the position of z in σ with respect to this triple. The only interesting case is :
$a <_\sigma z <_\sigma b$
if $d \in z \cup D(z)$ then $b \in z \cup D(z)$ which is not possible, therefore the 4 points condition is satisfied in G'.
So the proof terminates using induction on G'.

End-vertex problem for a search S :
Input : A graph $G = (V, E)$, and a vertex t.
Question : Is there σ an S-ordering of V such that $\sigma(n) = t$?

Theorem

Given a bipartite graph G and a vertex v of G, it is NP-complete to decide if there exists an execution of BFS on G such that v is the end-vertex.

Reduction direct from 3-SAT

For every $n \in \mathcal{N}$ we define a graph G_n, which has one special vertex r_n called the root. It is constructed recursively as follows :

- G_0 is the graph with one vertex r_0.

For every $n \in \mathcal{N}$ we define a graph G_n, which has one special vertex r_n called the root. It is constructed recursively as follows :

- G_0 is the graph with one vertex r_0.
- G_n is constructed from G_{n-1} by first adding three vertices : the new root r_n, and its two neighbours y_n and $\overline{y_n}$, that are also adjacent to r_{n-1}. Finally we attach a path of $2n-1$ new vertices to y_n (respectively $\overline{y_n}$) and label its end-vertex x_n (respectively $\overline{x_n}$).

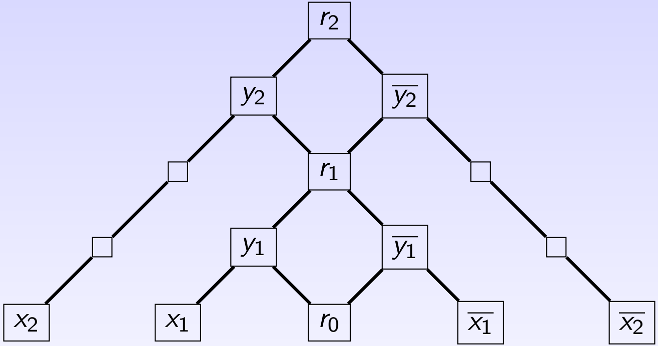


FIGURE: The graph G_2.

Proposition 1

G_n is a bipartite graph that has $(2n+1)(n+1)$ vertices that all are at distance at most $2n$ from r_n. There are $2n+1$ vertices at distance exactly $2n$ from r_n, and these are $x_1, \overline{x_1}, x_2, \overline{x_2}, \ldots, x_n, \overline{x_n}$ and r_0.

The following proposition is central to the reduction and concerns the order that we obtain on those $2n+1$ vertices when we do a BFS starting at the root.

Proposition 1

G_n is a bipartite graph that has $(2n+1)(n+1)$ vertices that all are at distance at most $2n$ from r_n. There are $2n+1$ vertices at distance exactly $2n$ from r_n, and these are $x_1, \overline{x_1}, x_2, \overline{x_2}, \ldots, x_n, \overline{x_n}$ and r_0.

The following proposition is central to the reduction and concerns the order that we obtain on those $2n+1$ vertices when we do a BFS starting at the root.

Proposition 2

Consider an order on the vertices of G_n given by an execution of a BFS starting at r_n. For each $1 \leq i \leq n$ at most one of x_i and $\overline{x_i}$ is before r_0. Moreover each of the 2^n choices of one among x_i and $\overline{x_i}$ for each i, can be obtained as the set of vertices that appear before r_0 for some BFS order of G_n.

End of the proof

1. We just add to the previous gadjet, the incidence bipartite clauses–variables and a pending edge $r_0 t$.

End of the proof

1. We just add to the previous gadjet, the incidence bipartite clauses–variables and a pending edge $r_0 t$.
2. There is a BFS ending at t iff the SAT instance is satisfiable.

End-vertex results	BFS	LBFS	DFS	LDFS	MNS
All Graphs	**NPC**	NPC	**NPC**	**NPC**	?(P)
Bipartite	**NPC**	?(NPC)	?(NPC)	?(NPC)	?(P)
Weakly Chordal	**NPC**	NPC	**NPC**	**NPC**	?(P)
Chordal	?(NPC)	?(NPC)	**NPC**	?	P
Split	**P**	**P**	**NPC**	**P**	P
Str. Chordal Split	**P**	**P**	**NPC**	**P**	P
Path Graphs	?	?(P)	**NPC**	?	P

A *graph search* is an iterative process that chooses at each step a vertex of the graph and numbers it (from 1 to n). Each vertex is chosen (also said *visited*) exactly once (even if the graph is disconnected). Let us now define a *General Tie-Breaking Label Search* (TBLS). It uses *labels* to decide the next vertex to be visited ; $label(v)$ is a subset of $\{1, ..., n\}$. A TBLS is defined on :

1. A graph $G = (V, E)$ on which the search is performed ;
2. A strict partial order $\prec$ over the label-set $P(\mathbb{N}^+)$;
3. An ordering τ of the vertices of V called the *tie-break permutation*.

TBLS$(G, \prec, \tau)$

foreach $v \in V$ **do** $label(v) \leftarrow \emptyset$;
for $i \leftarrow 1$ **to** n **do**
    $Eligible \leftarrow \{x \in V \mid x$ unnumbered and $\nexists y \in V$ such that $label(x) \prec label(y)\}$;
    Let v be the leftmost vertex of *Eligible* according to the ordering τ;
    $\sigma(v) \leftarrow i$;
    foreach unnumbered vertex w adjacent to v **do**
        $label(w) \leftarrow label(w) \cup \{i\}$;
    end
end

The ideas of this formalism :

- A graph search just produces a vertex ordering

The ideas of this formalism :

- A graph search just produces a vertex ordering
- Any ordering of the vertices can be used as a tie-break

The ideas of this formalism :

- A graph search just produces a vertex ordering
- Any ordering of the vertices can be used as a tie-break
- Then we can iterate graph searches and study what orderings can be obtained

With this formalism, in order to specify a particular search we just need to specify $\prec$, the partial order relation on the label sets for that search. The choice of permutation τ is useful in some situations described below ; otherwise, we consider the orderings output by an arbitrary choice of τ thanks to the following definition :

Definition

Let $\prec$ be some ordering over $P(\mathbb{N}^+)$. Then σ is a TBLS ordering for G and $\prec$ if there exists τ such that $\sigma = TBLS(G, \prec, \tau)$.

Usual conventions

$\mathbb{N}^+$ represents the set of integers strictly greater than 0 and $\mathbb{N}^+_p$ represents the set of integers strictly greater than 0 and less than p. $P(\mathbb{N}^+)$ denotes the power-set of $\mathbb{N}^+$ and $\mathfrak{S}_n$ denotes the set of all permutations of $\{1, ..., n\}$.

We always use the notation $<$ for the usual strict (i.e., irreflexive) order between integers, and $\prec$ for a partial strict order between elements from $P(\mathbb{N}^+)$ (or from another set when specified).

So far in the model we play with :

a number (label) associated to each vertex
a set a labels associated to each unnumbered vertex
and some partial order $\prec$ between sets of labels

A useful property

Theorem

A graph G, a search rule $\prec$, σ an ordering, then :
There exists τ such that $\sigma = TBLS(G, \prec, \tau)$ iff
$\sigma = TBLS(G, \prec, \sigma)$

Generic Search

We define $A \prec_{gen} B$ if and only if $A = \emptyset$ and $B \neq \emptyset$ and let σ be a permutation of V. The following conditions are equivalent :

1. σ is a generic search ordering of V (a TBLS using $\prec_{gen}$).
2. For every triple of vertices $a,\ b,\ c$ such that $a <_\sigma b <_\sigma c$, $a \in N(c) - N(b)$ there exists $d \in N(b)$ such that $d <_\sigma b$.

Ad-hoc min and max operators

For $A \in P(\mathbb{N}^+)$,

- let $umin(A)$ be :
  if $A = \emptyset$ then $umin(A) = \infty$ else $umin(A) = min\{i | i \in A\}$;

Ad-hoc min and max operators

For $A \in P(\mathbb{N}^+)$,

- let $umin(A)$ be :
  if $A = \emptyset$ then $umin(A) = \infty$ else $umin(A) = min\{i|i \in A\}$;
- and $umax(A)$ be :
  if $A = \emptyset$ then $umax(A) = 0$ else $umax(A) = max\{i|i \in A\}$.

DFS

We define $A \prec_{DFS} B$ if and only if $umax(A) < umax(B)$. Let σ be a permutation of V. The following conditions are equivalent :

1. σ is a DFS ordering (a TBLS using $\prec_{DFS}$).
2. for every triple of vertices a, b, c such that $a <_\sigma b <_\sigma c$, $a \in N(c) - N(b)$ there exists $d \in N(b)$ such that $a <_\sigma d <_\sigma b$.
3. for every triple of vertices a, b, c such that $a <_\sigma b <_\sigma c$, and a is the rightmost vertex of $N(b) \cup N(c)$ in σ, we have $a \in N(b)$.

BFS

We define $A \prec_{BFS} B$ if and only if $umin(A) > umin(B)$. Let σ be a permutation of V. The following conditions are equivalent :

1. σ is a BFS ordering (a TBLS using $\prec_{BFS}$).
2. for every triple $a, b, c \in V$ such that $a <_\sigma b <_\sigma c$, $a \in N(c) - N(b)$, there exists d such that $d \in N(b)$ and $d <_\sigma a$.
3. for every triple $a, b, c \in V$ such that $a <_\sigma b <_\sigma c$ and a is the leftmost vertex of $N(b) \cup N(c)$ in σ, we have $a \in N(b)$.

LDFS

We define $A \prec_{LDFS} B$ if and only if $umax(A - B) < umax(B - A)$.
Let σ be a permutation of V. The following conditions are equivalent :

1. σ is a LDFS ordering (a TBLS using $\prec_{LDFS}$)
2. for every triple $a, b, c \in V$ such that $a <_\sigma b <_\sigma c$, $a \in N(c) - N(b)$, there exists $a <_\sigma d <_\sigma b$, $d \in N(b) - N(c)$.
3. for every triple $a, b, c \in V$ such that $a <_\sigma b <_\sigma c$ and a is the rightmost vertex in $N(b) \triangle N(c)$ in σ, $a \in N(b) - N(c)$.

LBFS

We define $A \prec_{LBFS} B$ if and only if $umin(B - A) < umin(A - B)$. Let σ be a permutation of V. The following conditions are equivalent :

1. σ is a LBFS ordering (a TBLS using $\prec_{LBFS}$)
2. for every triple $a, b, c \in V$ such that $a <_\sigma b <_\sigma c$, $a \in N(c) - N(b)$, there exists $d <_\sigma a$, $d \in N(b) - N(c)$.
3. for every triple $a, b, c \in V$ such that $a <_\sigma b <_\sigma c$ and a is the leftmost vertex of $N(b) \triangle N(c)$ in σ, then a $\in N(b) - N(c)$.

LDFS

We define $A \prec_{LDFS} B$ if and only if $umax(A - B) < umax(B - A)$. Let σ be a permutation of V. The following conditions are equivalent :

1. σ is a LDFS ordering (a TBLS using $\prec_{LDFS}$)
2. for every triple $a, b, c \in V$ such that $a <_\sigma b <_\sigma c$, $a \in N(c) - N(b)$, there exists $a <_\sigma d <_\sigma b$, $d \in N(b) - N(c)$.
3. for every triple $a, b, c \in V$ such that $a <_\sigma b <_\sigma c$ and a is the rightmost vertex in $N(b) \triangle N(c)$ in σ, $a \in N(b) - N(c)$.

Consequences

- In our model only $\prec$ matters for a graph search.

Consequences

- In our model only $\prec$ matters for a graph search.
- Nice mathematical duality min-max between BFS and DFS (resp. LBFS and LDFS).

Consequences

- In our model only $\prec$ matters for a graph search.
- Nice mathematical duality min-max between BFS and DFS (resp. LBFS and LDFS).
- A stack (resp. a queue) is a data structure to manage a Maximum (resp. Minimum) current value.

Consequences

- In our model only $\prec$ matters for a graph search.
- Nice mathematical duality min-max between BFS and DFS (resp. LBFS and LDFS).
- A stack (resp. a queue) is a data structure to manage a Maximum (resp. Minimum) current value.
- But which data structures to implement $max(A \setminus B)$ or $min(A \setminus B)$?

Consequences

- In our model only $\prec$ matters for a graph search.
- Nice mathematical duality min-max between BFS and DFS (resp. LBFS and LDFS).
- A stack (resp. a queue) is a data structure to manage a Maximum (resp. Minimum) current value.
- But which data structures to implement $max(A \setminus B)$ or $min(A \setminus B)$?
- New characterizations of their orderings

Consequences

- In our model only $\prec$ matters for a graph search.
- Nice mathematical duality min-max between BFS and DFS (resp. LBFS and LDFS).
- A stack (resp. a queue) is a data structure to manage a Maximum (resp. Minimum) current value.
- But which data structures to implement $max(A \setminus B)$ or $min(A \setminus B)$?
- New characterizations of their orderings
- Prove results just using the orderings (without looking at all at the implementation).

Repeated LBFS$^+$ in this new model

Data: G an undirected graph, σ_0 a permutation on $V(G)$
Result: an ordering $\sigma_{|V(G)|}$
for $i = 1$ **to** $|V(G)|$ **do**
  $\sigma_i \leftarrow TBLS(G, \prec_{LBFS}, \sigma_{i-1}^d)$;
end
Output $\sigma_{|V(G)|}$;

A semi-lattice structure

Definition

For two TBLS searches S, S', we say that S' is an extension of S (denoted by $S \ll S'$) if and only if every S'-ordering σ also is an S-ordering. [a]

[a]. This definition is consistent with the usual extension ordering used in partial order theory ; in particular $S \ll S'$ means that the set of comparabilities in S is included in those of S'.

Definition

For two partial orders $\prec_P, \prec_Q$ on the same ground set X, we say that P extends Q if $\forall x, y \in X$, $x \prec_Q y$ implies $x \prec_P y$.

Theorem

Let S, S' be two TBLS. S' is an extension of S if and only if $\prec_{S'}$ is an extension of $\prec_S$.

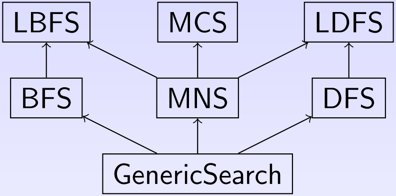


FIGURE: Summary of the heredity relationships. An arc from Search S to search S' means that S' extends S.

Clearly being an extension is a transitive relation. In fact $\ll$ structures the TBLS graph searches as $\wedge$-semilattice. The 0 search in this semi-lattice, denoted by the null search or S_{null}, corresponds to the empty ordering relation (no comparable pairs). At every step of S_{null} the Eligible set contains all unnumbered vertices. Therefore for every τ, $TBLS(G, \prec_{S_{null}}, \tau) = \tau$ and so any total ordering of the vertices can be produced by S_{null}.

The infimum between two searches S, S' can be defined as follows : For every pair of label sets A, B, we define : $A \prec_{S \wedge S'} B$ if and only if $A \prec_S B$ and $A \prec_{S'} B$.

Exercise

Show that Layered Search does not belong to the TBLS formalism.
How to include it ?

LEXUP

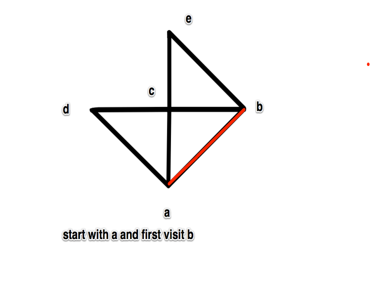
Data: a graph $G = (V, E)$ and a start vertex s;
Result: an ordering σ of V;
Assign the label $\emptyset$ to all vertices ;
$label(s) \leftarrow \{n\}$;
for $i \leftarrow 1$ **à** n **do**
    Pick an unnumbered vertex v with lexicographically largest label;
    $\sigma(i) \leftarrow v$;
    foreach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

Algorithm 4: LEXUP

LEXDOWN

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 unnumbered 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

Algorithm 5: LEXDOWN



$LBFS$: a, b, c, d, e
$LDFS$: a, b, c, e, d
$LEXUP$: a, b, e, c, d Hamilton path ! ! !
$LEXDOWN$: a, b, d, c, e
All different orderings in this simple example.
Potential application to AI discovery algorithms.

Perspectives

- Use this tie-break model to prove properties of graph searches

Perspectives

- Use this tie-break model to prove properties of graph searches
- Understanding the greedy aspects of LBFS and LDFS on cocomparability graphs.

Perspectives

- Use this tie-break model to prove properties of graph searches
- Understanding the greedy aspects of LBFS and LDFS on cocomparability graphs.
- More precisely : matroidal aspects for LDFS and anti-matroidal for LBFS.

Perspectives

- Use this tie-break model to prove properties of graph searches
- Understanding the greedy aspects of LBFS and LDFS on cocomparability graphs.
- More precisely : matroidal aspects for LDFS and anti-matroidal for LBFS.
- Study iterations of graph searches and their cycles.

Perspectives

- Use this tie-break model to prove properties of graph searches
- Understanding the greedy aspects of LBFS and LDFS on cocomparability graphs.
- More precisely : matroidal aspects for LDFS and anti-matroidal for LBFS.
- Study iterations of graph searches and their cycles.
- Find applications of 2 new lexicographic searches (Lexup, Lexdown) which came by symmetry on the algorithms.

- *Influence of the tie-break rule on the end-vertex problem*
  Pierre Charbit, Michel Habib, Antoine Mamcarz
  Discrete Mathematics & Theoretical Computer Science, Vol 16, No 2 (2014).

  http://www.dmtcs.org/dmtcs-ojs/index.php/dmtcs/article/view/2519
- TBLS : *A tie-break model for graph search*
  with Derek Corneil, Jérémie Dusart, Michel Habib, Antoine Mamcarz and Fabien de Montgolfier
  à paraître dans **Discrete Applied Mathematics**.
- Informations about LEXUP and LEXDOWN in J. Dusart's PhD, june 2014.