LexBFS and its applications

Guillaume Aubian

March 12, 2020

Overview

What's in a graph ?

Graph

We consider non-oriented, simple and **connected** graphs

INCLUDE GRAPHS EXAMPLES AND COUNTEREXAMPLES

Generic Search

```
for i in [1, ..., n]:
    if i == 1:
        u = any vertex
    else:
        u = any unvisited marked vertex
    visit(u)
    for v in neighbours(u):
        mark(v)
```

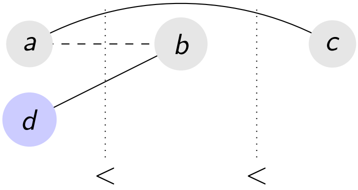
Another Characterization

Let's number vertices in the order they are visited.

Theorem

An order σ corresponds to a Generic Search if and only if

$$\forall a <_\sigma b <_\sigma c, ac \in E \text{ and } ab \notin E, \exists d <_\sigma b \text{ st } db \in E$$



DFS

INCLUDE GRAPH EXAMPLE

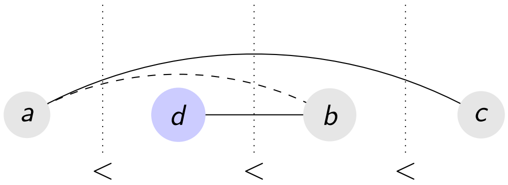
```
for i in [1, ..., n]:
    if i == 1:
        u = any vertex
    else:
        u = any unvisited vertex w/ max label
    visit(u)
    for v in neighbours(u):
        label[v] = i
```

Another Characterization

Theorem

An order σ corresponds to a DFS if and only if

$$\forall a <_\sigma b <_\sigma c, ac \in E \text{ and } ab \notin E, \exists a <_\sigma d <_\sigma b \text{ st } db \in E$$



BFS

```
for i in [n, ..., 1]:
    if i == n:
        u = any vertex
    else:
        u = any unvisited vertex w/ max label
    visit(u)
    for v in neighbours(u):
        if v has no label:
            label[v] = i
```

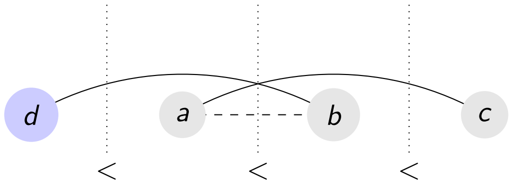
INCLUDE GRAPH EXAMPLE

Another Characterization

Theorem

An order σ corresponds to a BFS if and only if

$$\forall a <_\sigma b <_\sigma c, ac \in E \text{ and } ab \notin E, \exists d <_\sigma a \text{ st } db \in E$$



Let's rewrite BFS

```
for i in [n, ..., 1]:
    if i == n:
        u = any vertex
    else:
        u = any unvisited vertex
            w/ max first element of label
    visit(u)
    for v in neighbours(u):
        label[v].append(i)
```

Here is LexBFS

```
for i in [n, ..., 1]:
    if i == n:
        u = any vertex
    else:
        u = any unvisited vertex
            w/ max lexicographical label
    visit(u)
    for v in neighbours(u):
        label[v].append(i)
```

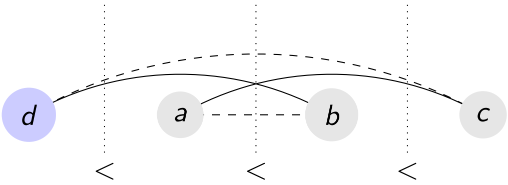
INCLUDE GRAPH EXAMPLE

Another Characterization

Theorem

An order σ corresponds to a LexBFS if and only if

$\forall a <_\sigma b <_\sigma c, ac \in E \text{ and } ab \notin E, \exists d <_\sigma a \text{ st } db \in E \text{ and } dc \notin E$



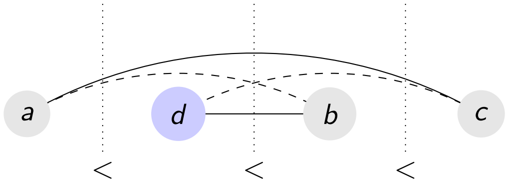
Another Characterization

Iterating on [1, ..., n] and prepending, we obtain LexDFS

Theorem

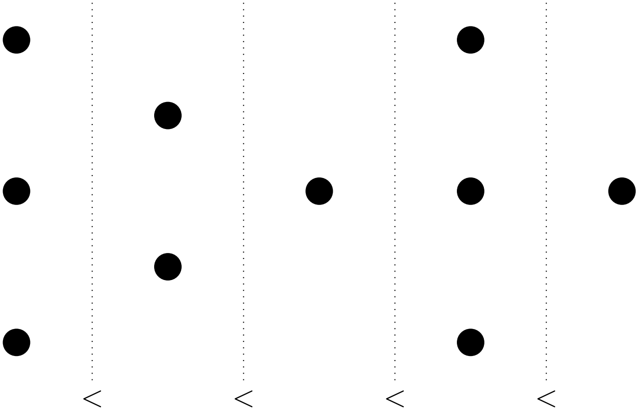
An order σ corresponds to a LexDFS if and only if

$$\forall a <_\sigma b <_\sigma c, ac \in E, ab \notin E, \exists a <_\sigma d <_\sigma b \; st \; db \in E, dc \notin E$$



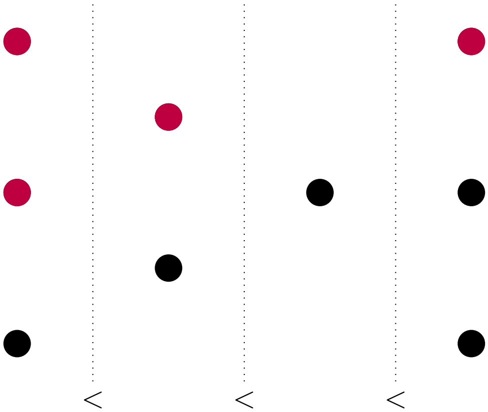
How to implement LexBFS: Pivoting

Let's order vertices by label :



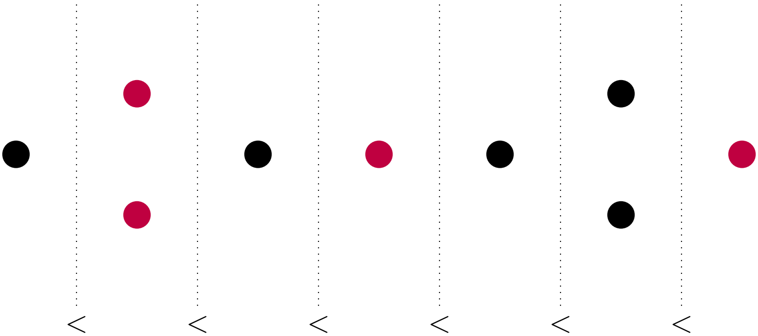
How to implement LexBFS: Pivoting

We remove one of the maximum vertex :



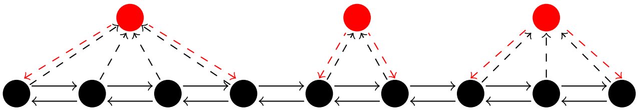
How to implement LexBFS: Pivoting

Now, we split each bucket



Data structure for pivoting

- Vertices in a doubly linked list
- Pointer from each vertex to its class
- Pointers from each class to its first and last vertex



Some remarks

Complexity

- Each step is $O(deg_v)$
- Thus, LexBFS is linear
- co-LexBFS is linear

LexDFS

As of 2019, no known linear-time for LexDFS

Chordal Graph

Chordal Graph

A graph is chordal if all cycles of four or more vertices have an edge that is not part of the cycle but connects two of its vertices.

Perfect Elimination Ordering (=PEO)

σ is a PEO if for each vertex v, v and its neighbours occuring after v in σ form a clique.

Chordal Graph

A graph is chordal if it admits a perfect elimination ordering.

Finding a Perfect Elimination Ordering

Main Theorem

If G is chordal, any LexBFS ordering taken backward is a PEO.

Proof:

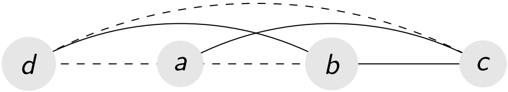
c : non simplicial vertex

Thus $\exists a <_\sigma b \in N(c)$ with $ab \notin E$

σ is a LexBFS thus $\exists d <_\sigma a$ with $db \in E, dc \notin E$

G is chordal, so $ad \notin E$

Ending the proof



- **Initialization:** (d, a, b) such that $da \notin E, ab \notin E, db \in E$
- **General Case:** $\exists d' <_\sigma d$ with $d'a \in E, d'b \notin E$ so $d'd \notin E$.
- Using the triple (d', d, a), infinite chain.