Problem

- Given directed graph $G = (V, E)$

Problem

- Given directed graph $G = (V, E)$
- Operations:
  - answer query (v, w): 'if $v \rightsquigarrow w$?'

Problem

- Given directed graph $G = (V, E)$
- Operations:
  - answer query (v, w): 'if $v \rightsquigarrow w$?'
  - add arc $v \longrightarrow w$
- Assumptions
  - For each query (v, w) there is $v \rightsquigarrow w$ or $w \rightsquigarrow v$
  - None addition creates the cycle

Problem

- Given directed graph $G = (V, E)$
- Operations:
  - answer query (v, w): 'if $v \rightsquigarrow w$?'
  - add arc $v \longrightarrow w$
- Assumptions
  - For each query (v, w) there is $v \rightsquigarrow w$ or $w \rightsquigarrow v$
  - None addition creates the cycle

Brute force search

For each query (v, w) traverse graph from vertex v until w is reached

Problem

- Given directed graph $G = (V, E)$
- Operations:
  - answer query (v, w): 'if $v \rightsquigarrow w$?'
- Assumptions
  - For each query (v, w) there is $v \rightsquigarrow w$ or $w \rightsquigarrow v$
  - None addition creates the cycle

Brute force search

For each query (v, w) traverse graph from vertex v until w is reached

No addition operation

Compute the topological order of V. Then $v \rightsquigarrow w$ iff $v < w$.

Problem

- Given directed graph $G = (V, E)$
- Operations:
  - answer query (v, w): 'if $v \rightsquigarrow w$?'
  - add arc $v \longrightarrow w$
- Assumptions
  - For each query (v, w) there is $v \rightsquigarrow w$ or $w \rightsquigarrow v$
  - None addition creates the cycle

Brute force search

For each query (v, w) traverse graph from vertex v until w is reached

Solution — Incremental topological order

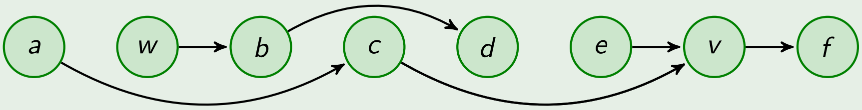
Maintain topological order due to addition of new arcs.

Addition of arc $v \longrightarrow w$

If $v < w$ then do nothing.

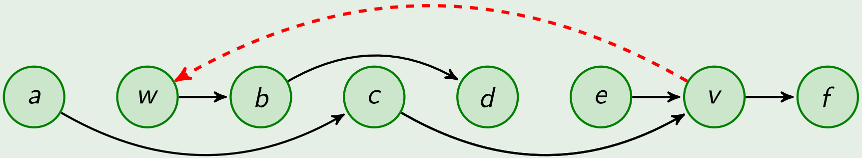
Addition of arc $v \longrightarrow w$

The case $w < v$



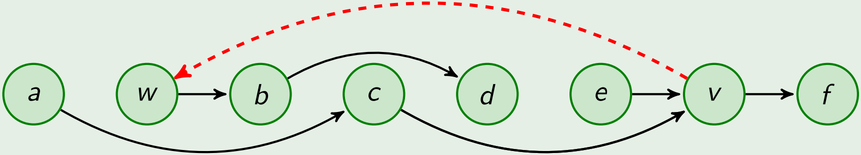
Addition of arc $v \longrightarrow w$

The case $w < v$



Addition of arc $v \longrightarrow w$

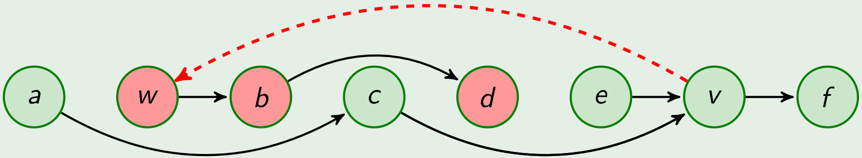
The case $w < v$



Definitions:

Addition of arc $v \longrightarrow w$

The case $w < v$

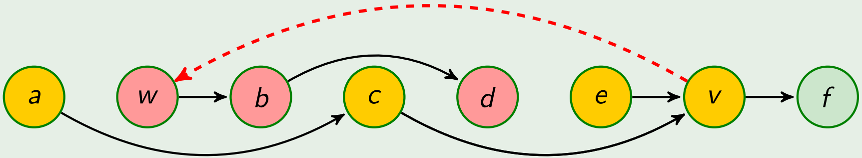


Definitions:

- **Forward vertices** $F = \{w\} \cup \{u\colon w \rightsquigarrow u\}$

Addition of arc $v \to w$

The case $w < v$

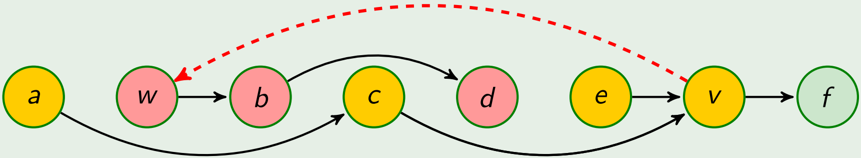


Definitions:

- **Forward vertices** $F = \{w\} \cup \{u \colon w \rightsquigarrow u\}$
- **Backward vertices** $B = \{v\} \cup \{u \colon u \rightsquigarrow v\}$

Addition of arc $v \longrightarrow w$

The case $w < v$



Definitions:

- **Forward vertices** $F = \{w\} \cup \{u\colon w \rightsquigarrow u\}$
- **Backward vertices** $B = \{v\} \cup \{u\colon u \rightsquigarrow v\}$

General Idea

Do bidirectional search forward from w and backward from v until finding either a cycle or a set of vertices whose reordering will restore topological order.

Definition

- u is scanned if it is forward (resp. backward) and we trawersed all its outgoing (resp. incoming) arcs.

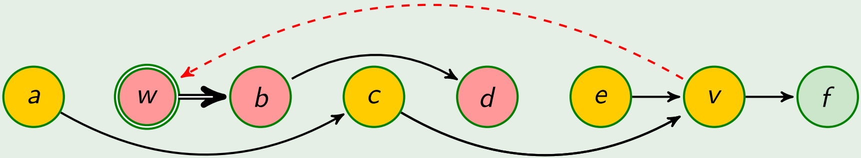
Definition

- u is scanned if it is forward (resp. backward) and we trawersed all its outgoing (resp. incoming) arcs.

Algorithm A

1. Traverse arcs forward from forward vertices and backward from backward vertices until there is a vertex s such that all forward vertices less than s and all backward vertices greater than s are scanned.

Example



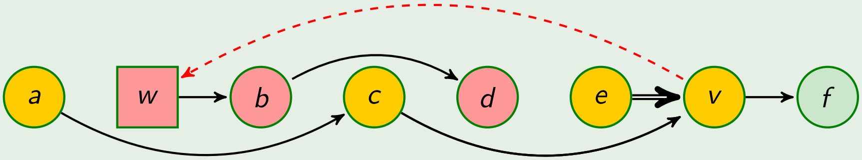
Definition

- u is scanned if it is forward (resp. backward) and we trawersed all its outgoing (resp. incoming) arcs.

Algorithm A

1. Traverse arcs forward from forward vertices and backward from backward vertices until there is a vertex s such that all forward vertices less than s and all backward vertices greater than s are scanned.

Example



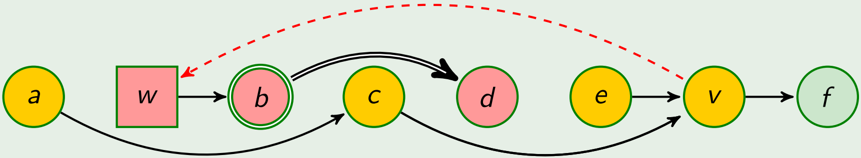
Definition

- u is scanned if it is forward (resp. backward) and we trawersed all its outgoing (resp. incoming) arcs.

Algorithm A

1. Traverse arcs forward from forward vertices and backward from backward vertices until there is a vertex s such that all forward vertices less than s and all backward vertices greater than s are scanned.

Example



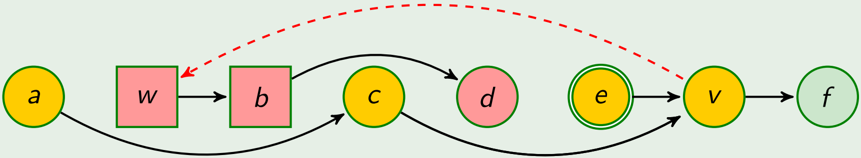
Definition

- u is scanned if it is forward (resp. backward) and we trawersed all its outgoing (resp. incoming) arcs.

Algorithm A

1. Traverse arcs forward from forward vertices and backward from backward vertices until there is a vertex s such that all forward vertices less than s and all backward vertices greater than s are scanned.

Example



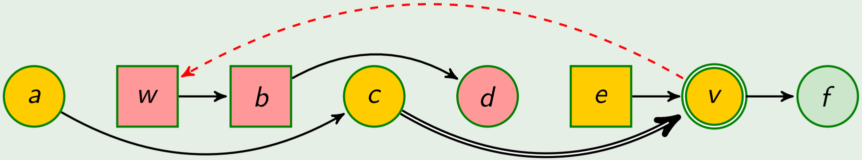
Definition

- u is scanned if it is forward (resp. backward) and we trawersed all its outgoing (resp. incoming) arcs.

Algorithm A

1. Traverse arcs forward from forward vertices and backward from backward vertices until there is a vertex s such that all forward vertices less than s and all backward vertices greater than s are scanned.

Example



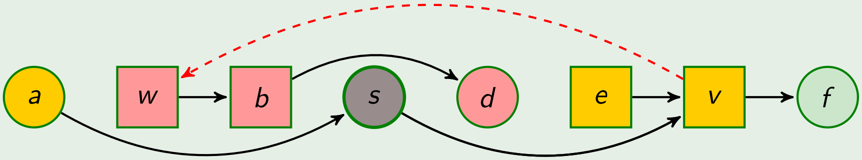
Definition

- u is scanned if it is forward (resp. backward) and we trawersed all its outgoing (resp. incoming) arcs.

Algorithm A

1. Traverse arcs forward from forward vertices and backward from backward vertices until there is a vertex s such that all forward vertices less than s and all backward vertices greater than s are scanned.

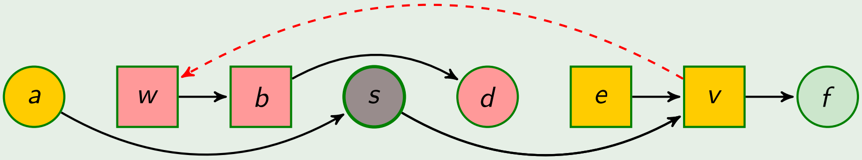
Example



Algorithm A

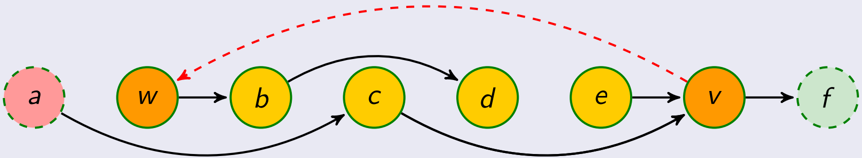
1. Traverse arcs forward from forward vertices and backward from backward vertices until there is a vertex s such that all forward vertices less than s and all backward vertices greater than s are scanned.
2. Let $X = \{x \in F\colon x < s\}$ and $Y = \{y \in B\colon s < y\}$. Find topological orders of O_X and O_Y of the subgraphs induced by X and Y, respectively.
3. Assume s is not forward (the case of s not backward is symmetric). Delete the vertices in $X \cup Y$ from the current vertex order and reinsert them just after s, in order O_Y followed by O_X.

Example



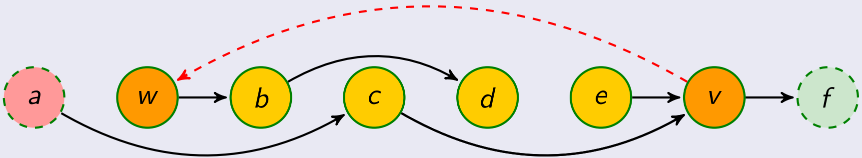
Affected Region

Restrict the search only to the affected region, i.e., the set of vertices between w and v.



Affected Region

Restrict the search only to the affected region, i.e., the set of vertices between w and v.



$\Downarrow$

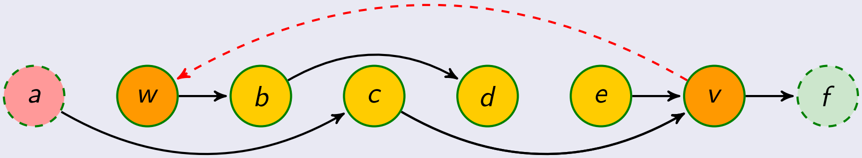
Complexity

$\mathcal{O}(n)$ — amortized time per arc addition

$\mathcal{O}(1)$ — for each query $a < b$

Affected Region

Restrict the search only to the affected region, i.e., the set of vertices between w and v.

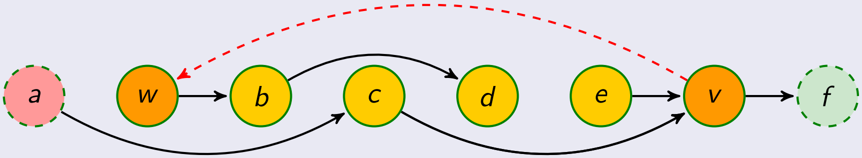


Compatible arcs

We call an arc $u \to x$ traversed forward and an arc $y \to z$ traversed backward compatible if $u < z$.

Affected Region

Restrict the search only to the affected region, i.e., the set of vertices between w and v.



Compatible arcs

We call an arc $u \to x$ traversed forward and an arc $y \to z$ traversed backward compatible if $u < z$.

Lemma

If the searches are compatible, the amortized number of arcs traversed during searches is $\mathcal{O}(m^{1/2})$ per arc addition.

Compatible search

Traverse arcs $u \to x$ forward in non-decreasing order on u and arcs $y \to z$ backward in non-increasing order on z.

Compatible search

Traverse arcs $u \to x$ forward in non-decreasing order on u and arcs $y \to z$ backward in non-increasing order on z.

Some implementation details

- We can implement an ordered search using two heaps (priority queues) to store unscanned forward and unscanned backward vertices

Compatible search

Traverse arcs $u \to x$ forward in non-decreasing order on u and arcs $y \to z$ backward in non-increasing order on z.

Some implementation details

- We can implement an ordered search using two heaps (priority queues) to store unscanned forward and unscanned backward vertices $\Rightarrow$ amortized time bound of $\mathcal{O}(m^{1/2} \log n)$ per arc addition.

Compatible search

Traverse arcs $u \to x$ forward in non-decreasing order on u and arcs $y \to z$ backward in non-increasing order on z.

Some implementation details

- We can implement an ordered search using two heaps (priority queues) to store unscanned forward and unscanned backward vertices $\Rightarrow$ amortized time bound of $\mathcal{O}(m^{1/2} \log n)$ per arc addition.
- We maintain the vertex order using a data structure such that testing the predicate $x < y$ for two vertices x and y takes $\mathcal{O}(1)$ time, as does deleting a vertex from the order and reinserting it just before or just after another vertex.