Invited Talk: GraphSM/DBKDA-2014

The Sixth International Conference on Advances in Databases, Knowledge, and Data Applications

April 20 - 26, 2014 - Chamonix, France

About Reachability in Graphs

Andreas Schmidt

(1)
Department of Informatics and Business Information Systems
University of Applied Sciences Karlsruhe
Moltkestraße 30
76133 Karlsruhe
Germany

(2)
Institute for Applied Sciences
Karlsruhe Institute of Technologie
PO-box 3640
76021 Karlsruhe
Germany

Outlook

- Motivation
- Some Graph definitions
- Different Approaches
- Summary & Further Readings

Motivation

Reachability queries are a very basic type of a graph query

Why do we need reachability queries?

- Bioinformatics (biological networks, genome biology)
- Social Science, link analysis, citation analysis
- XML Queries/Database query optimizer
- Internet routing
- Source Code Analysis
- Geographic navigation systems
- Ontology queries (RDF/OWL)

Directed Graphs

- Graph G:

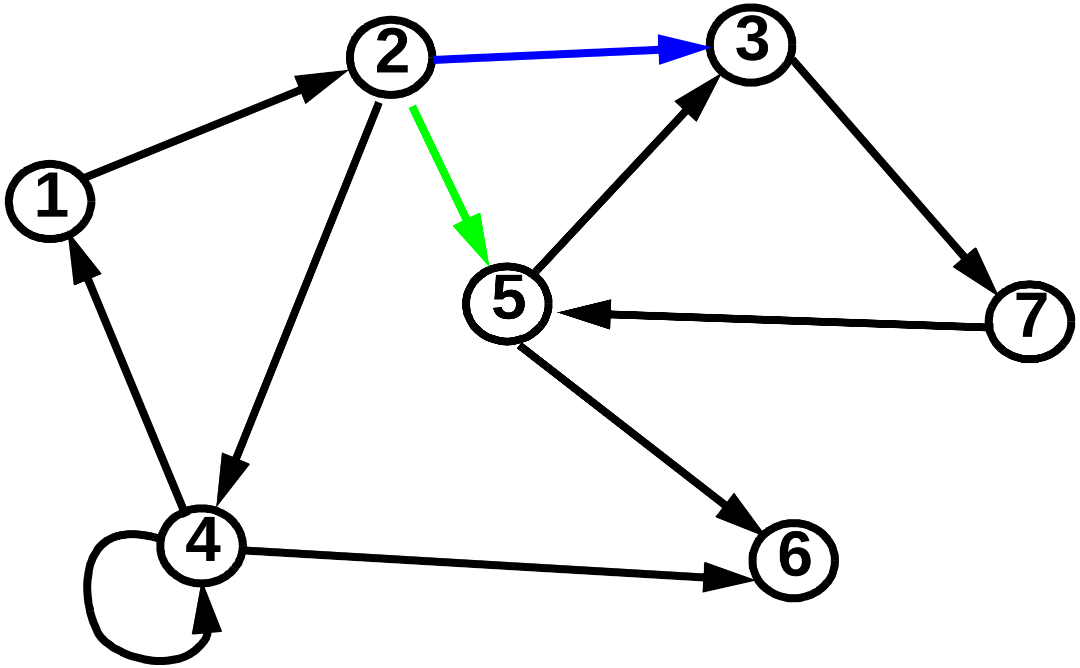
  $G = (V, E)$

  $V = \{v_1, v_2, ..., v_n\}$

  E: binary relation on V

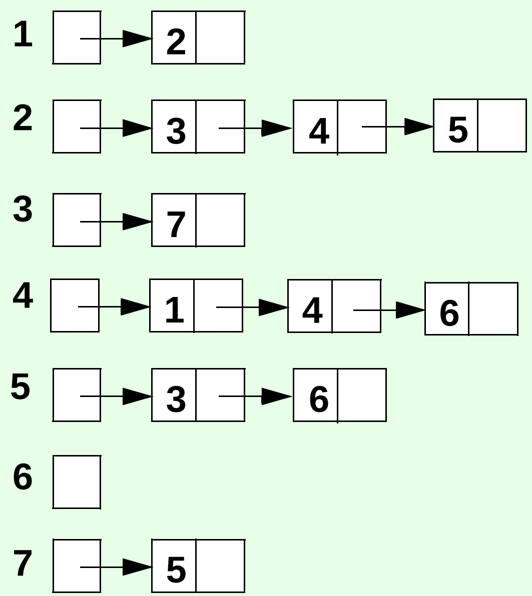
  $E=\{ (v_1,v_2), (v_2, v_3), (v_2, v_5), ... \}$

- Further concepts:
  - Path
  - Path length
  - Cyclic/Acyclic graph



Representation Forms for Directed Graphs

Adjacency list



Memory: $\mathcal{O}(|V| + |E|)$

Access: $\mathcal{O}(|V|)$ - nodes unsorted

$\mathcal{O}(\log_2|V|)$ - nodes sorted

Adjacency matrix

	1	2	3	4	5	6	7
1	0	1	0	0	0	0	0
2	0	0	1	1	1	0	0
3	0	0	0	0	0	0	1
4	1	0	0	1	0	1	0
5	0	0	1	0	0	1	0
6	0	0	0	0	0	0	0
7	0	0	0	0	1	0	0

Memory: $\mathcal{O}(|V^2|)$

Access: $\mathcal{O}(1)$

Reachability Query Types

- Query Types:
  - single pair
  - single source
  - multi source reachability

- Approaches:
  - Query on demand using breath- or depth-first search.
  - Precalculate the transitive closure, which contains all the reachability information
  - Something in between the two above solutions

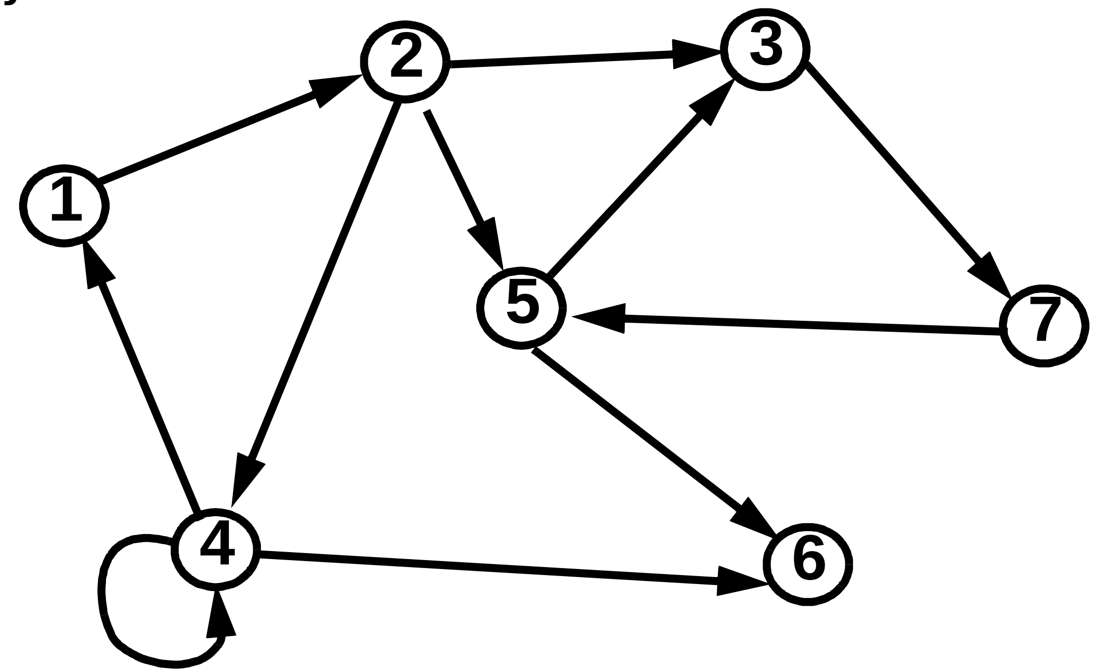
Summary Breath-/Depth-First Search

- Query Time: O(|V| + |E|)
- Additional memory consumption: none
- For large graphs to slow to answer queries efficiently

Transitive Closure

$G^+ = (V, E^+)$

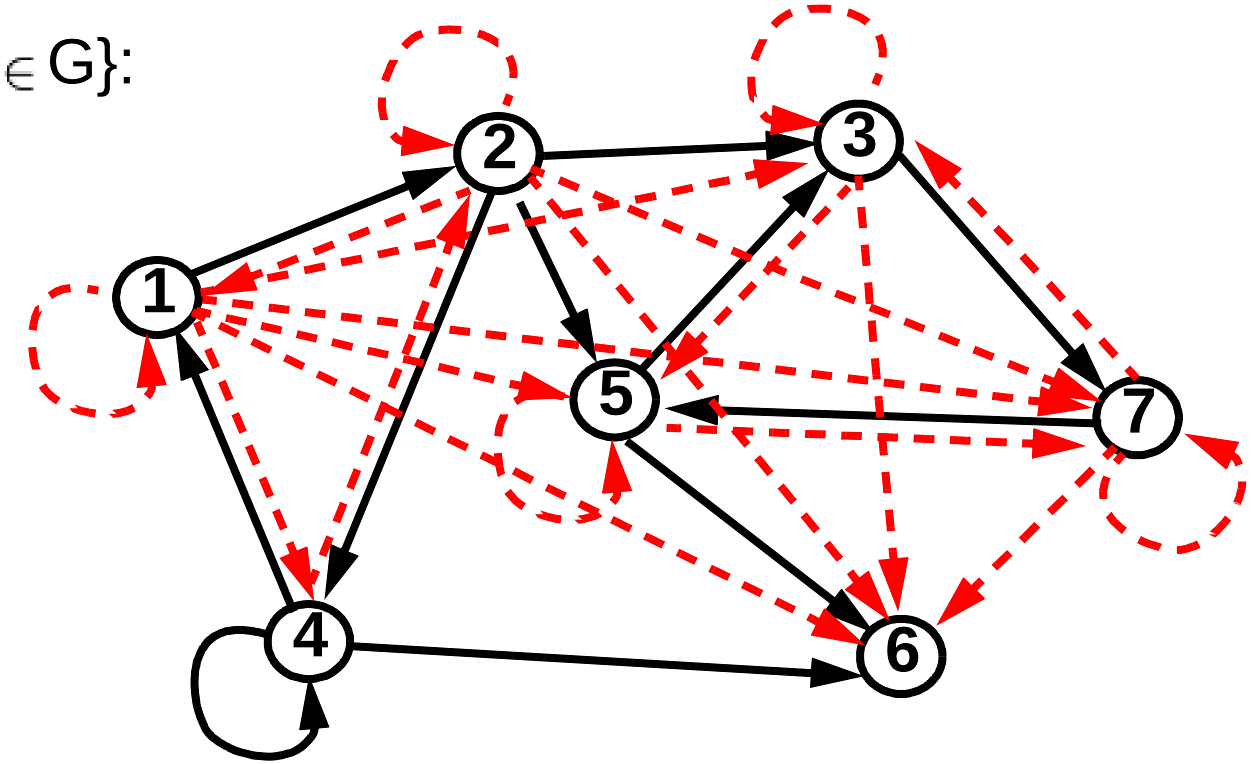
$E^+ = \{u, v \in V: u\text{->}v \in G\}$:



Transitive Closure

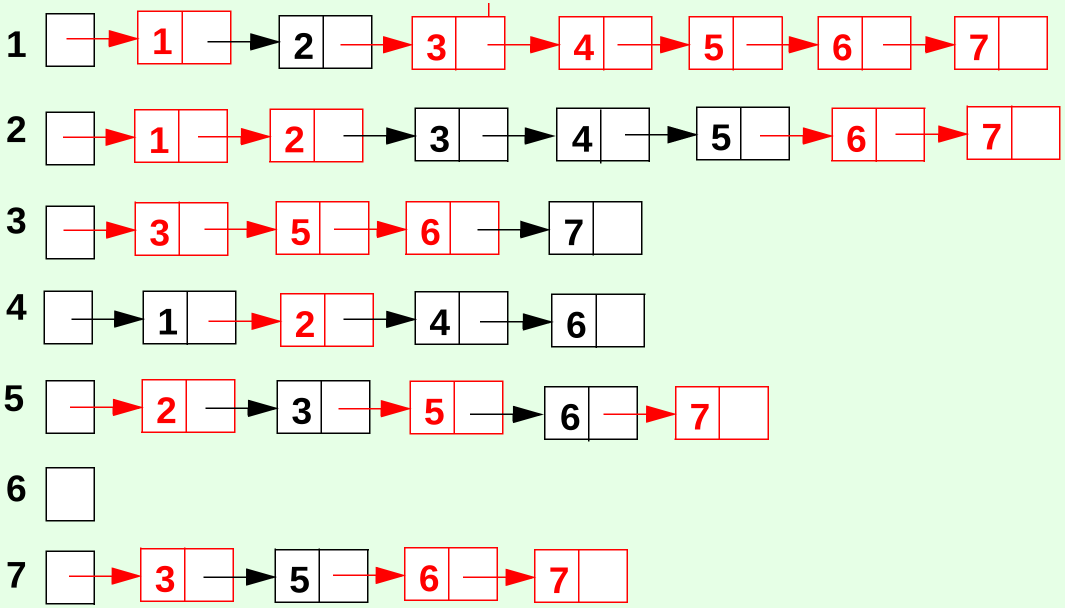
$G^+ = (V, E^+)$

$E^+ = \{u, v \in V: u\text{->}v \in G\}$:



Transitive Closure

Adjacency list



Memory: $\mathcal{O}(|V| + |E^+|)$

Access: $\mathcal{O}(|V|)$ - nodes unsorted

$\mathcal{O}(\log_2|V|)$ - nodes sorted

Adjacency matrix

	1	2	3	4	5	6	7
1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1
3	0	0	1	0	1	1	1
4	1	1	0	1	0	1	0
5	0	1	1	0	1	1	1
6	0	0	0	0	0	0	0
7	0	0	1	0	1	1	1

Memory: $\mathcal{O}(|V^2|)$

Access: $\mathcal{O}(1)$

Summary Transitive Closure

- Query Time
  - adjacency matrix: $O(1)$
  - unsorted adjacency list: $O(|V|)$
  - sorted adjacency list: $O(\log_2|V|)$
- memory consumption
  - adjacency matrix: $O(|V|^2)$
  - adjacency list: $O(|V| + |E^+|)$

- Additional Index construction time: $O(|V| * |E|)$

Time/Space Complexity of different approaches

	Query Time	Index Const. Time	Index Size
Transitive Closure	$O(1)$	$O(n * m)$	$O(n^2)$
Tree+SSPI	$O(m - n)$	$O(n + m)$	$O(n + m)$
GRIPP	$O(m - n)$	$O(n + m)$	$O(n + m)$
Dual-Labeling	$O(1)$	$O(n + m + t^3)$	$O(n + t^2)$
Tree Cover	$O(\log n)$	$O(nm)$	$O(n^2)$
Chain Cover	$O(\log k)$	$O(n^2 + knk^{1/2})$	$O(n * k)$
Path-Tree Cover	$O(\log^2 k')$	$O(m * k')$ or $O(n * m)$	$O(n * k')$
2-Hop Cover	$O(m^{1/2})$	$O(n^3 \|T * C\|)$	$O(n * m^{1/2})$
3-Hop Cover	$O(\log n + k)$	$O(kn^2 * \|Con(G)\|\|$	$O(n * k)$
BFS/DFS	$O(n + m)$	-	-

Source: Charu C. Aggarwal and Haixun Wang. 2010. Managing and Mining Graph Data (1st ed.). Springer Publishing Company, Incorporated.

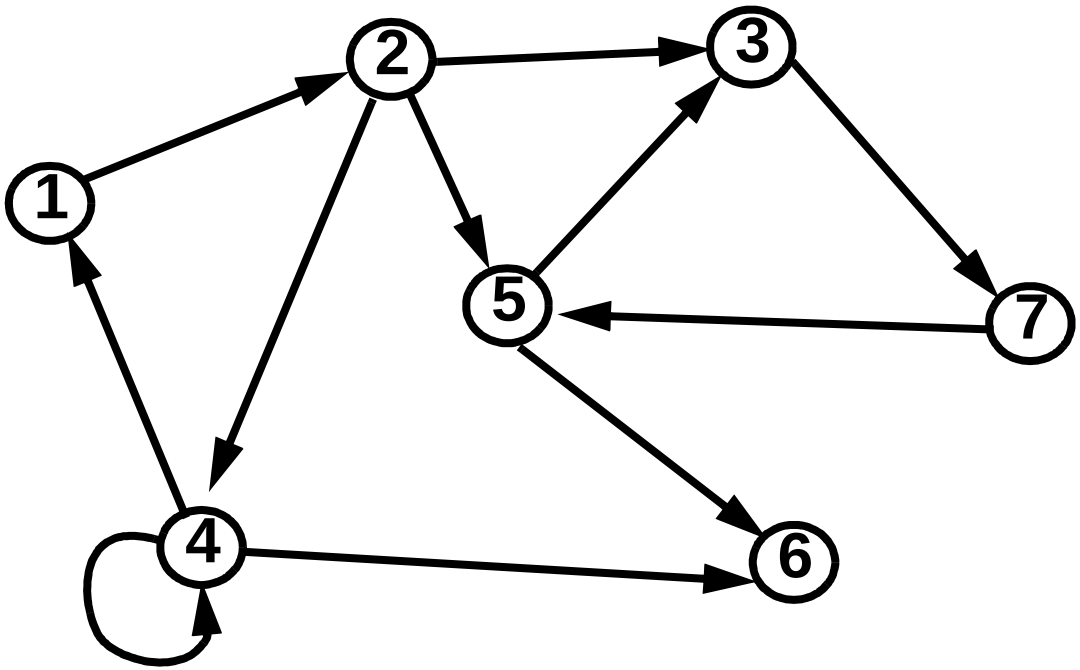
Some ideas ...

- Algorithms optimized for spares/dense graphs
- Transitive closure only over subgraphs
- Spanning tree (i.e. single interval tree coding schema - SIT) + additional data structure
- Represent adjacency matrix as compressed bitmaps

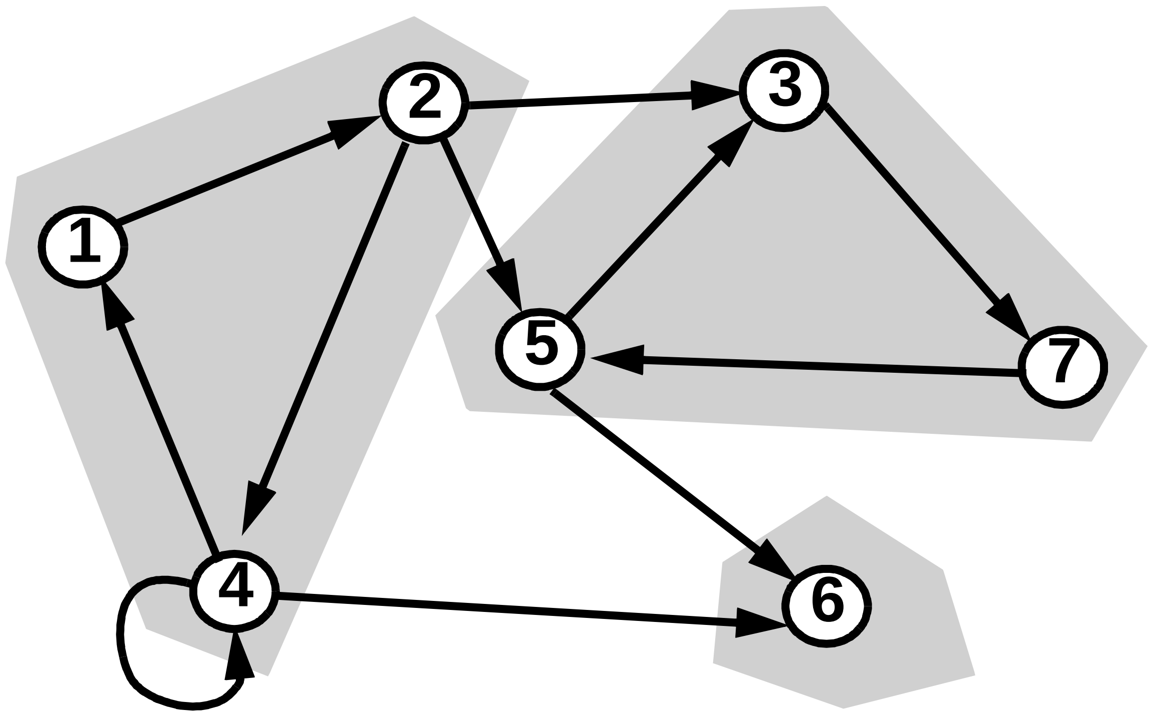
=> see literature at the end for details ...

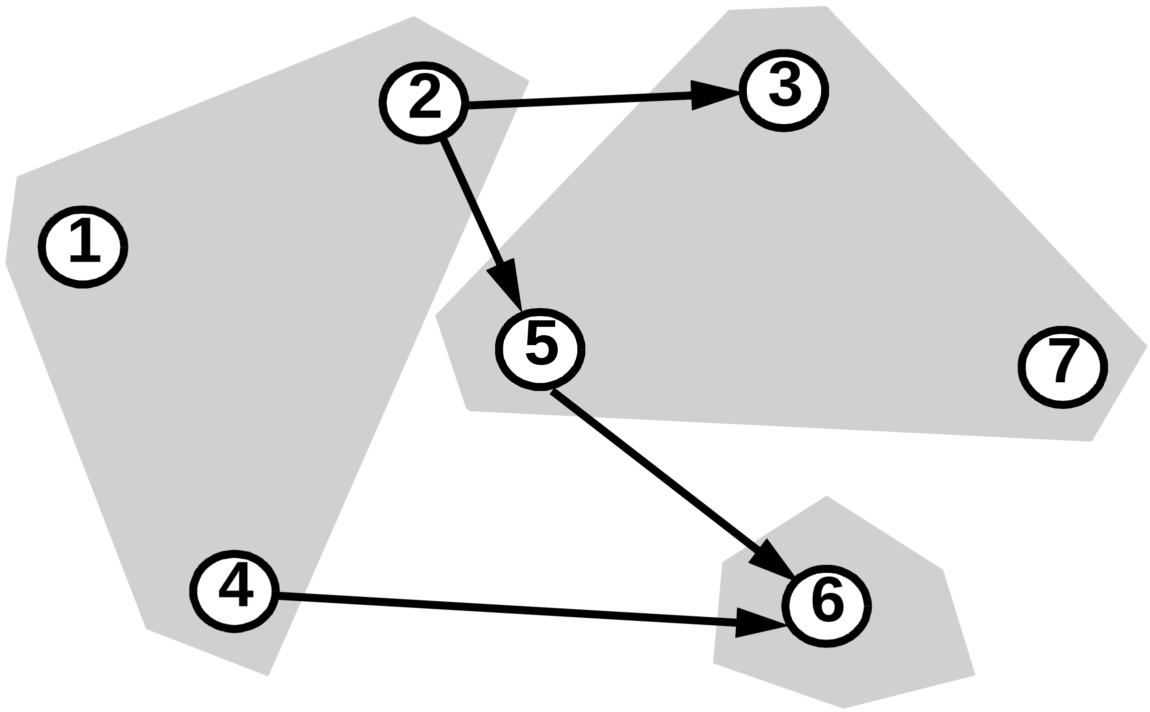
- Reduction of graph size
  (Strongly connected components)

Strongly Connected Components

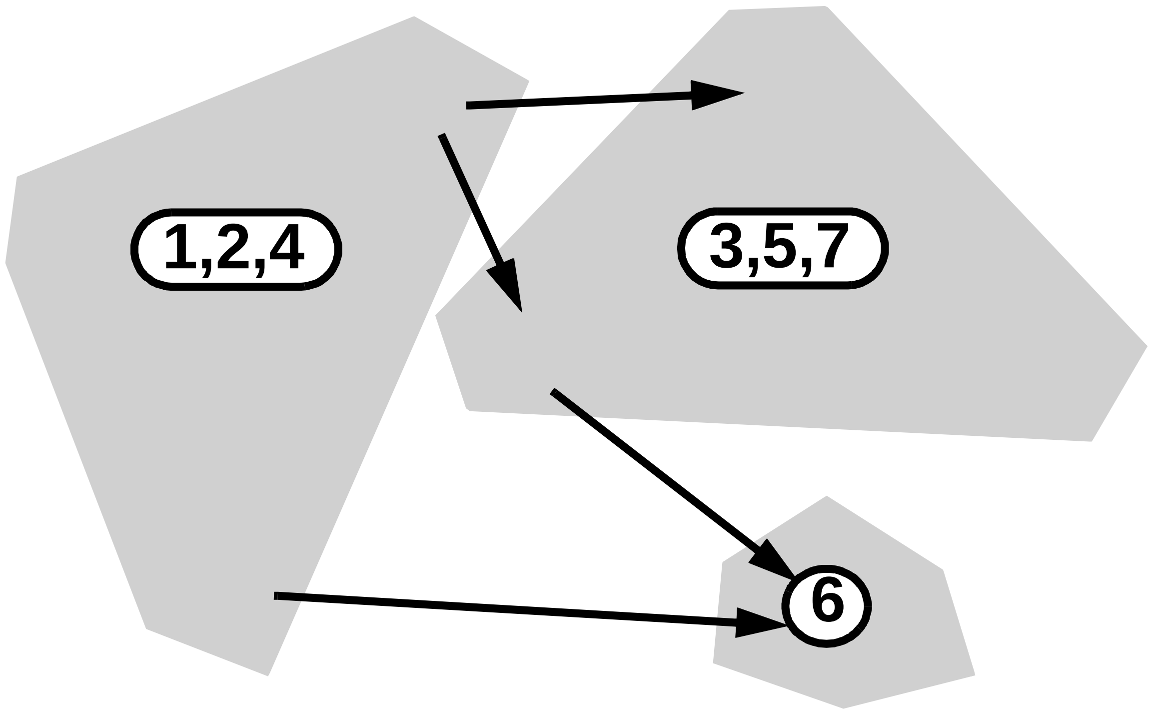


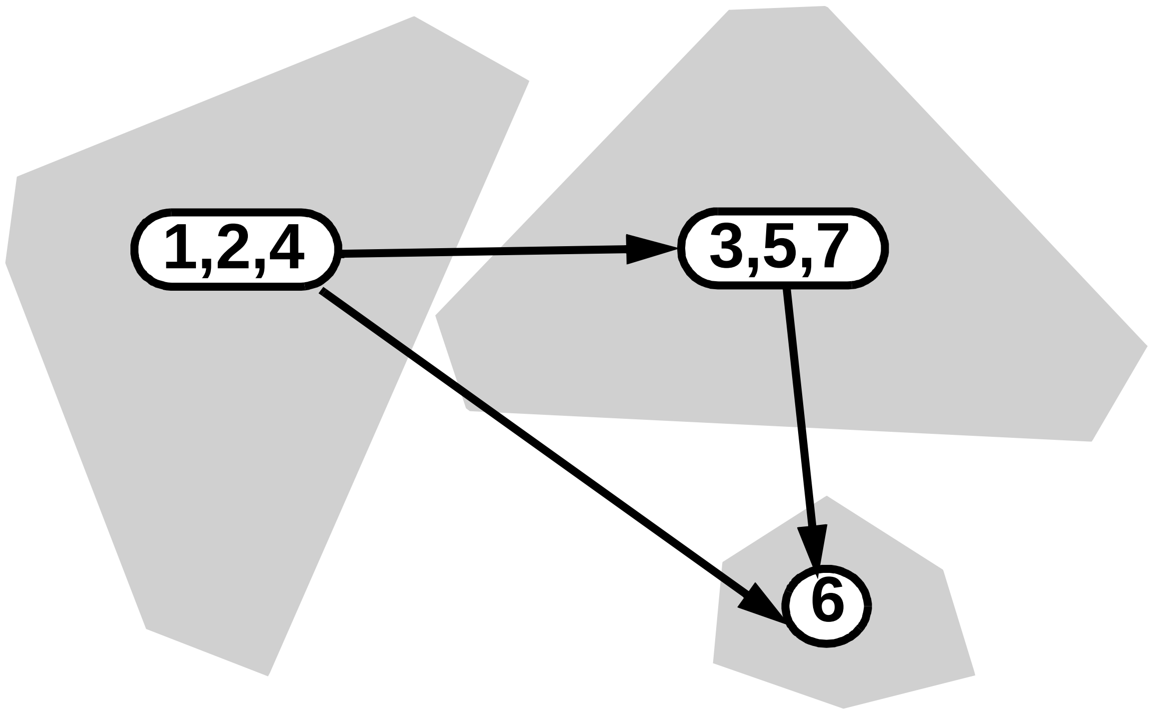
Strongly Connected Components





2
3
1
5
7
4
6

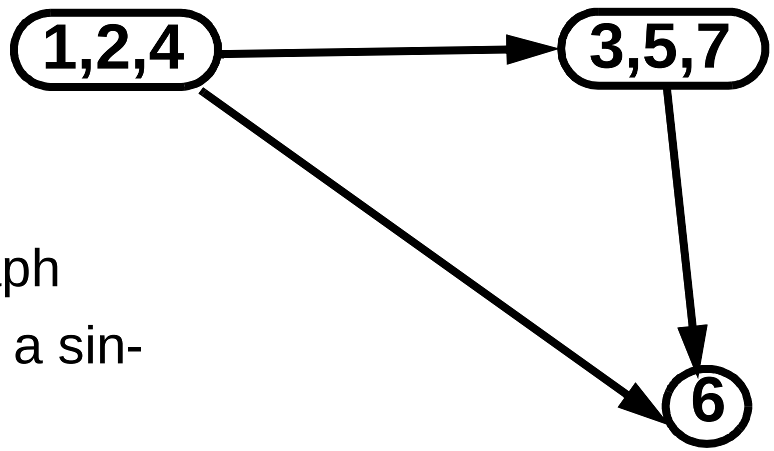

1,2,4
3,5,7
6


1,2,4
3,5,7
6

Tarjan's Algorithm

- Depth first search
- start at arbitrary node
- Time complexity:

  $\mathcal{O}(|V|+|E|)$



- Looks for cycles in graph
- Cycles are shrinked to a single nodes

Summary

- Reachability queries in graphs seem at first glance very simple queries
- But ...
  - in reality they have a wide range of use (query optimization, bioinformatics, social science, internet routing, geographic information systems, ...)
  - are not so simple to answer (quickly)
  - A wide range of algorithms have been developed to solve this problem for special cases (published at SIGMOD, ICDE, VLDB)
  - Always tradeoff between query time and memory consumption + index construction time

Literature

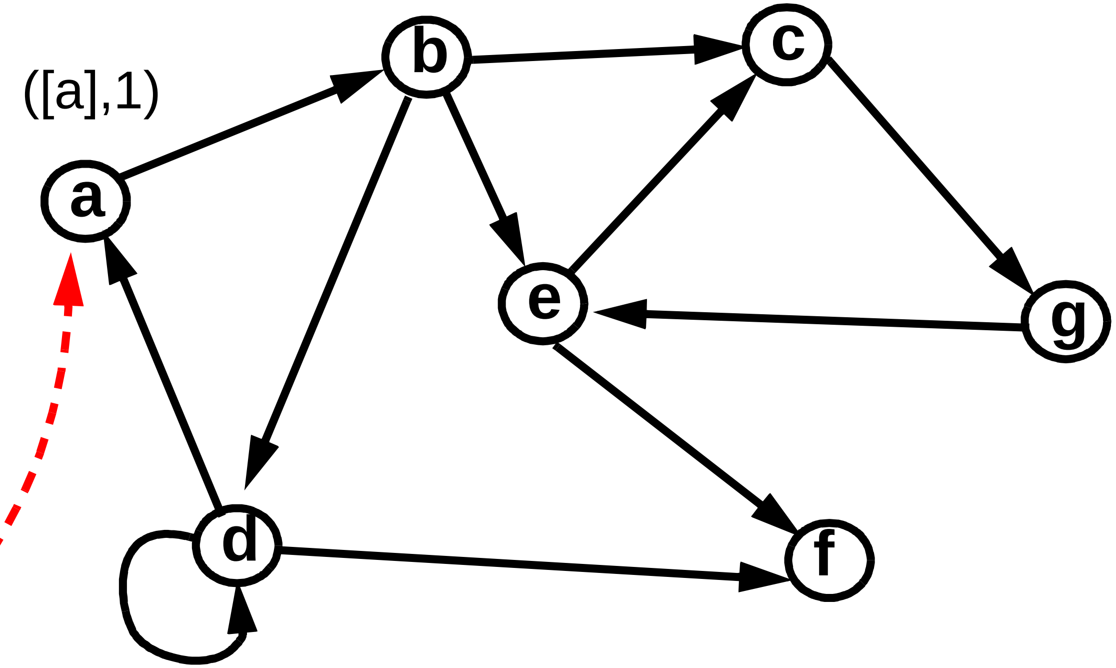
- [Tho04] Mikkel Thorup. 2004. Compact oracles for reachability and approximate distances in planar digraphs. J. ACM 51, 6 (November 2004), 993-1024.
- [Ski08] Steven S. Skiena, The Algorithm Design Manual, 2nd edition, 2008, Springer
- [AW10] Charu C. Aggarwal and Haixun Wang. 2010. Managing and Mining Graph Data (1st ed.). Springer Publishing Company, Incorporated.
- [SM11] Sebastiaan J. van Schaik, Oege de Moor: A memory efficient reachability data structure through bit vector compression. SIGMOD Conference 2011: 913-924
- [YC10] Jeffrey Xu Yu, Jiefeng Cheng; Graph Reachability Queries: A Survey; In: Managing and Mining Graph Data - Advances in Database Systems Volume 40, 2010, pp 181-215

Backup Slides

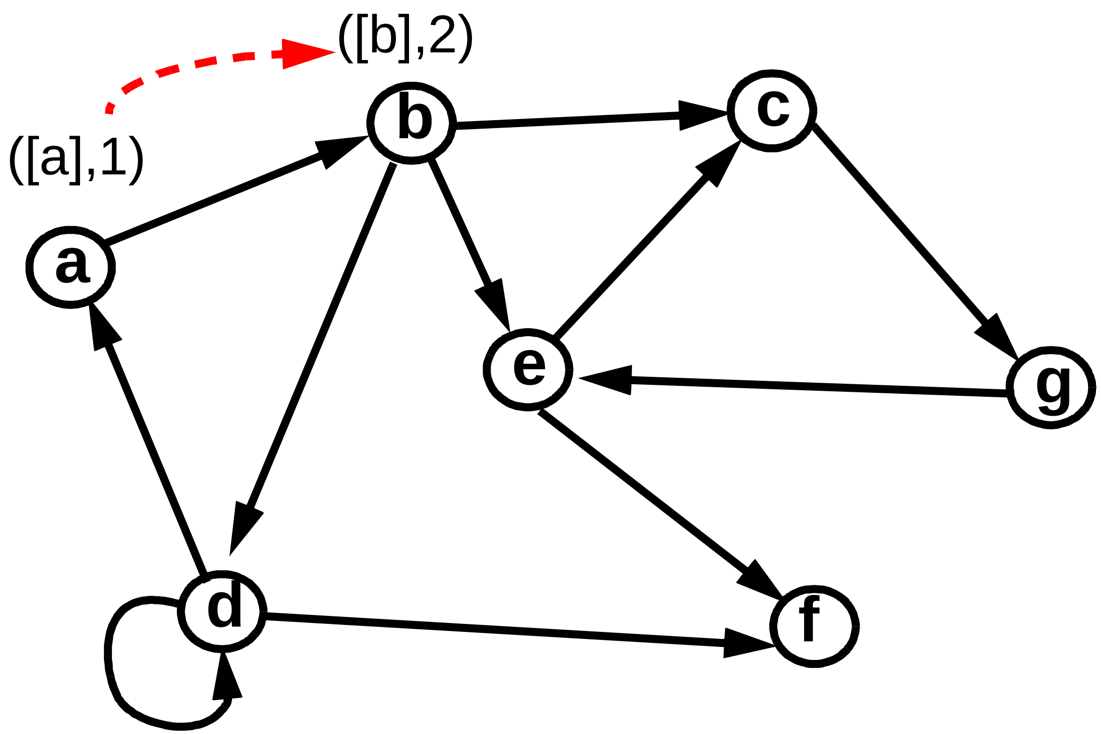
How to find strongly connected components?

- Tarjan's Algorithm:
  - Depth first search
  - start at arbitrary node
  - Time complexity:
    $\mathcal{O}(|V+E|)$
  - Looks for cycles in graph
  - Cycles are shrinked to a single node
  - Example:

Start



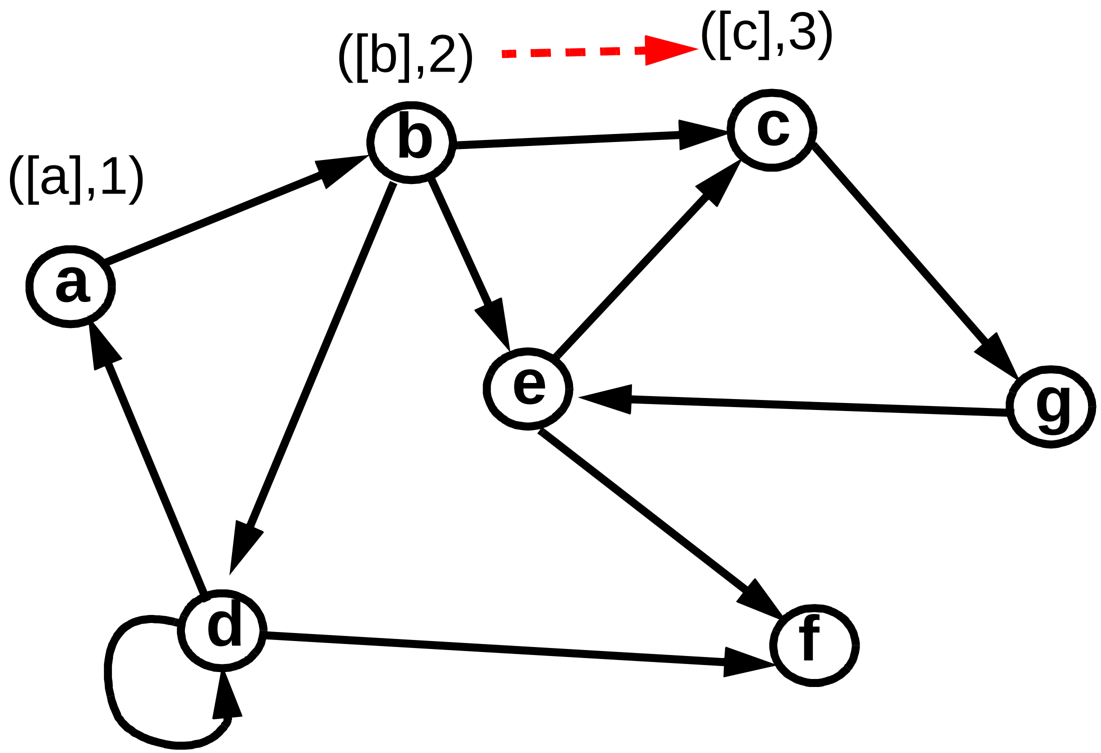
- Tarjan's Algorithm:
  - Depth first search
  - start at arbitrary noden-ode
  - $\mathcal{O}$(|V+E|) Time com-plexity:
  - Looks for cycles in graph
  - Cycles are shrinked to a single node
  - Example:



- Tarjan's Algorithm:
  - Depth first search
  - start at arbitrary node
  - Time complexity:

    $\mathcal{O}$(|V+E|)

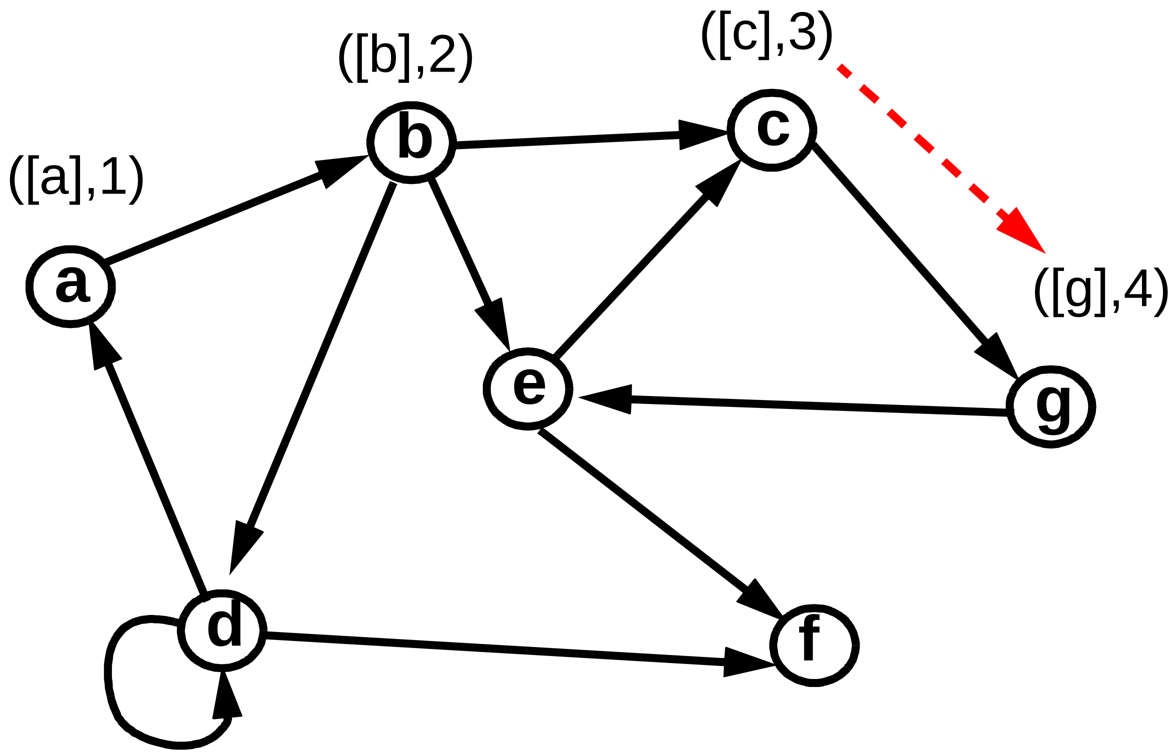
  - Looks for cycles in graph
  - Cycles are shrinked to a single node
  - Example:



- Tarjan's Algorithm:
  - Depth first search
  - start at arbitrary node
  - Time complexity:

    $\mathcal{O}(|V+E|)$

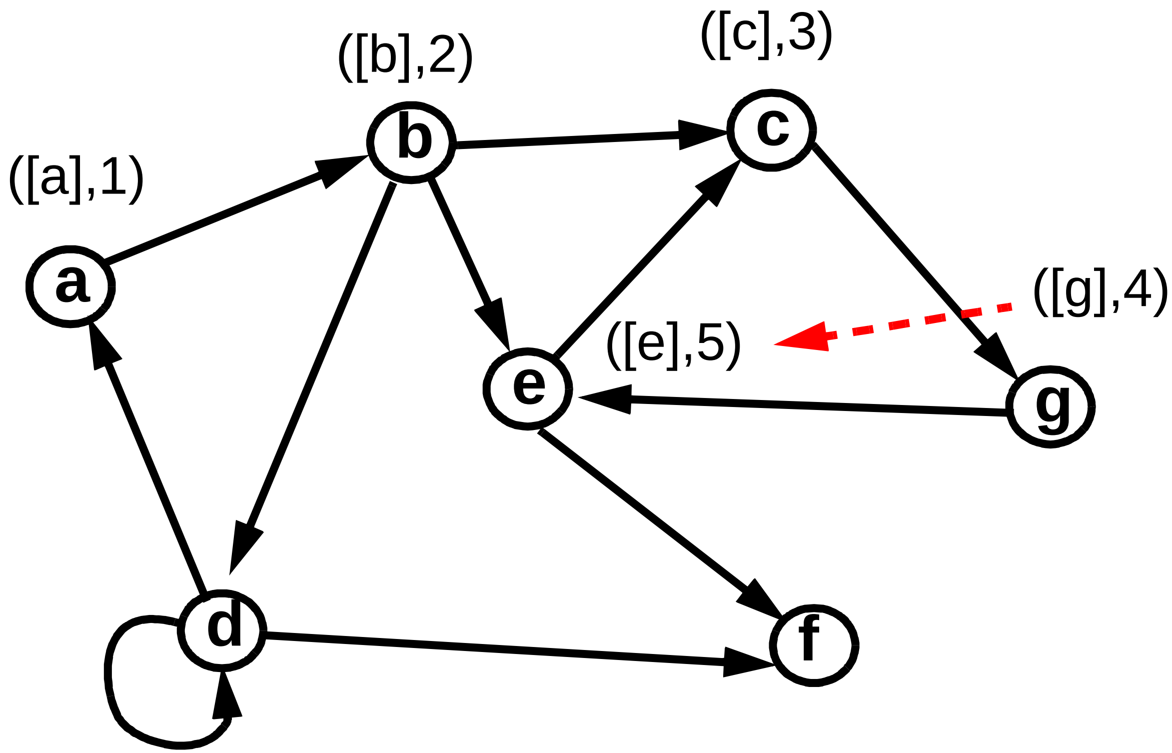
  - Looks for cycles in graph
  - Cycles are shrinked to a single node
  - Example:



- Tarjan's Algorithm:
  - Depth first search
  - start at arbitrary node
  - Time complexity:

    $\mathcal{O}(|V+E|)$

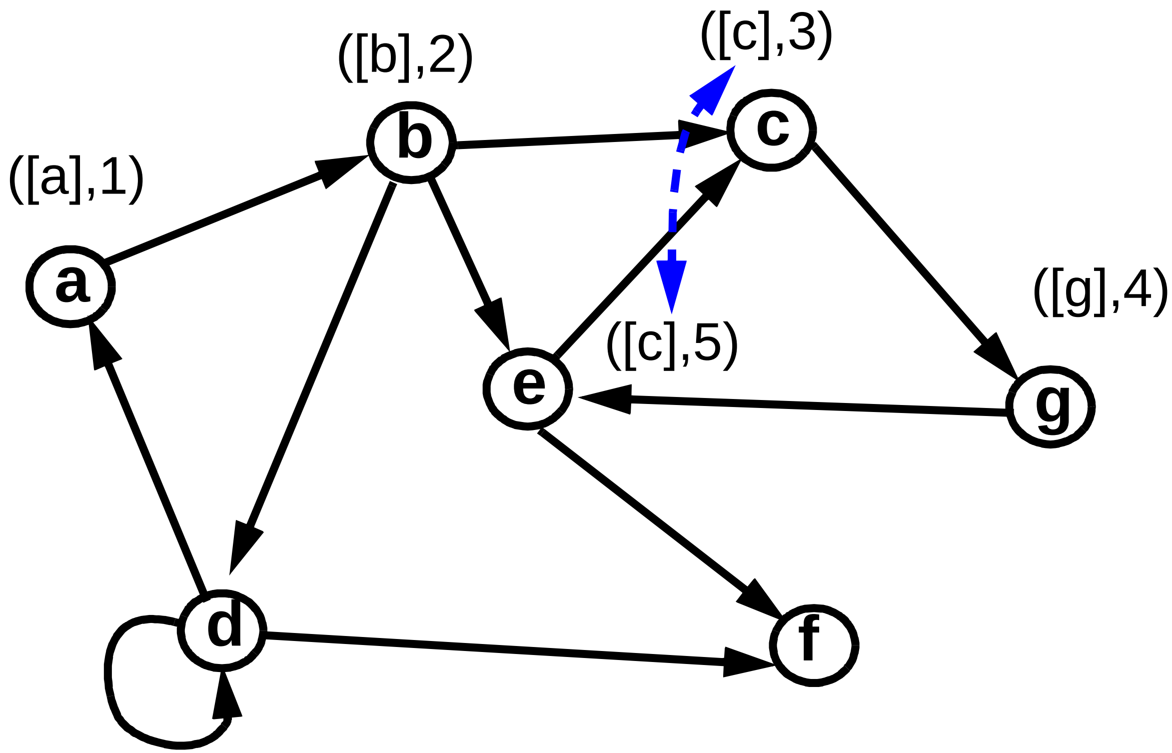
  - Looks for cycles in graph
  - Cycles are shrinked to a single node
  - Example:



- Tarjan's Algorithm:
  - Depth first search
  - start at arbitrary node
  - Time complexity:

    $\mathcal{O}$(|V+E|)

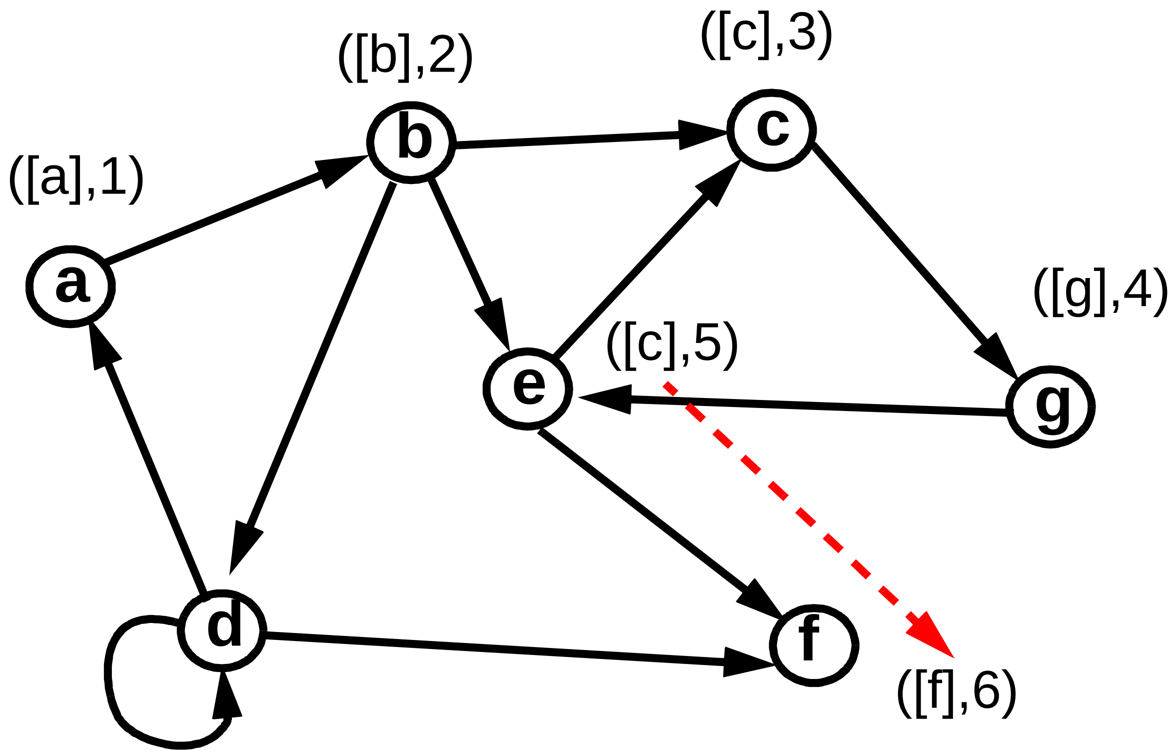
  - Looks for cycles in graph
  - Cycles are shrinked to a single node
  - Example:



- Tarjan’s Algorithm:
  - Depth first search
  - start at arbitrary node
  - Time complexity:

    $\mathcal{O}(|V+E|)$

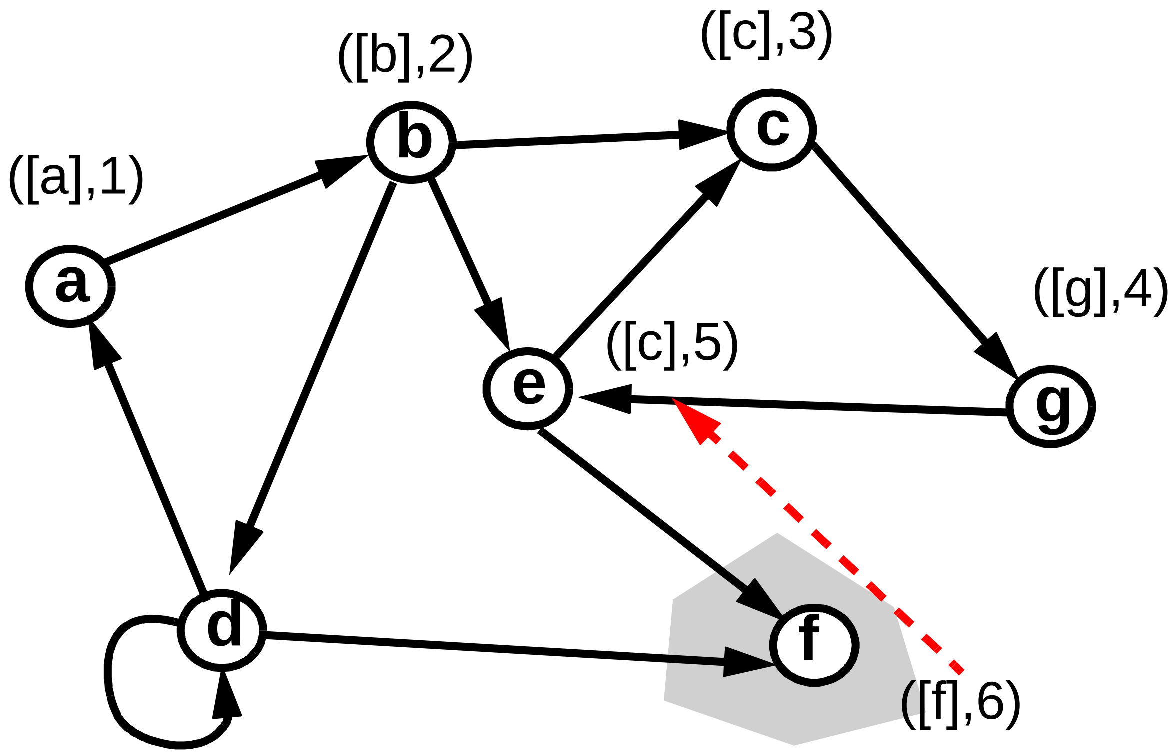
  - Looks for cycles in graph
  - Cycles are shrinked to a single node
  - Example:



- Tarjan's Algorithm:
  - Depth first search
  - start at arbitrary node
  - Time complexity:

    $\mathcal{O}(|V+E|)$

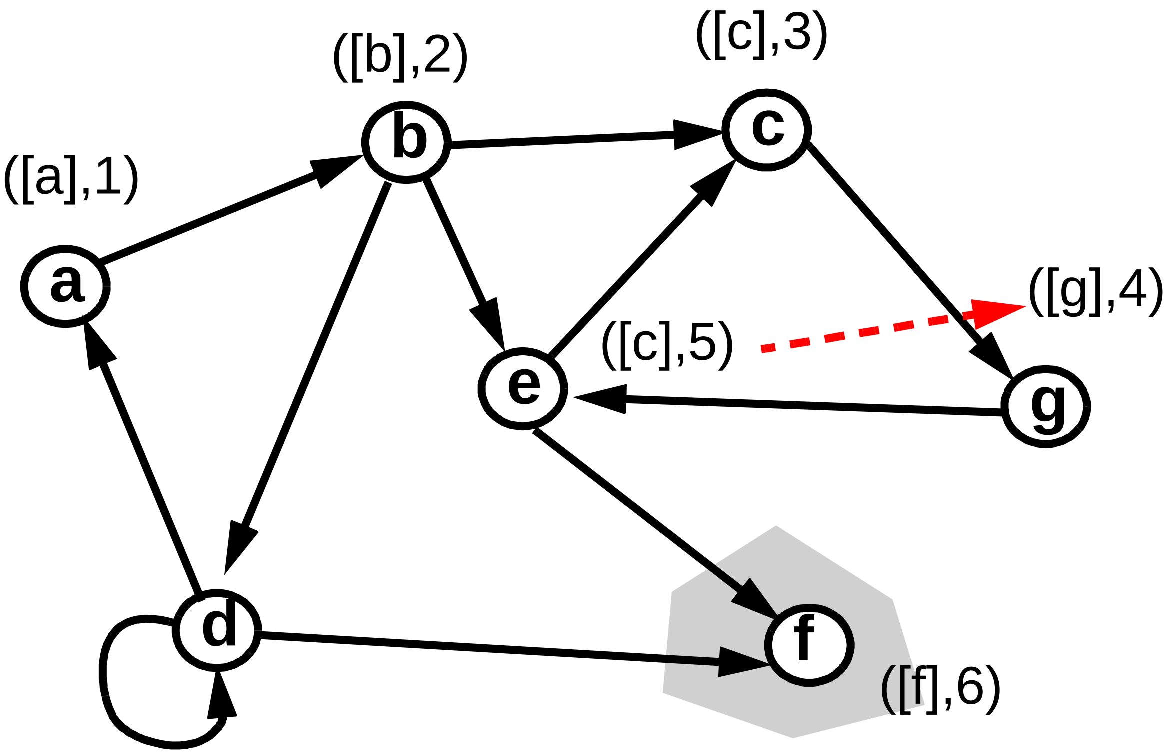
  - Looks for cycles in graph
  - Cycles are shrinked to a single node
  - Example:



- Tarjan's Algorithm:
  - Depth first search
  - start at arbitrary node
  - Time complexity:

    $\mathcal{O}(|V+E|)$

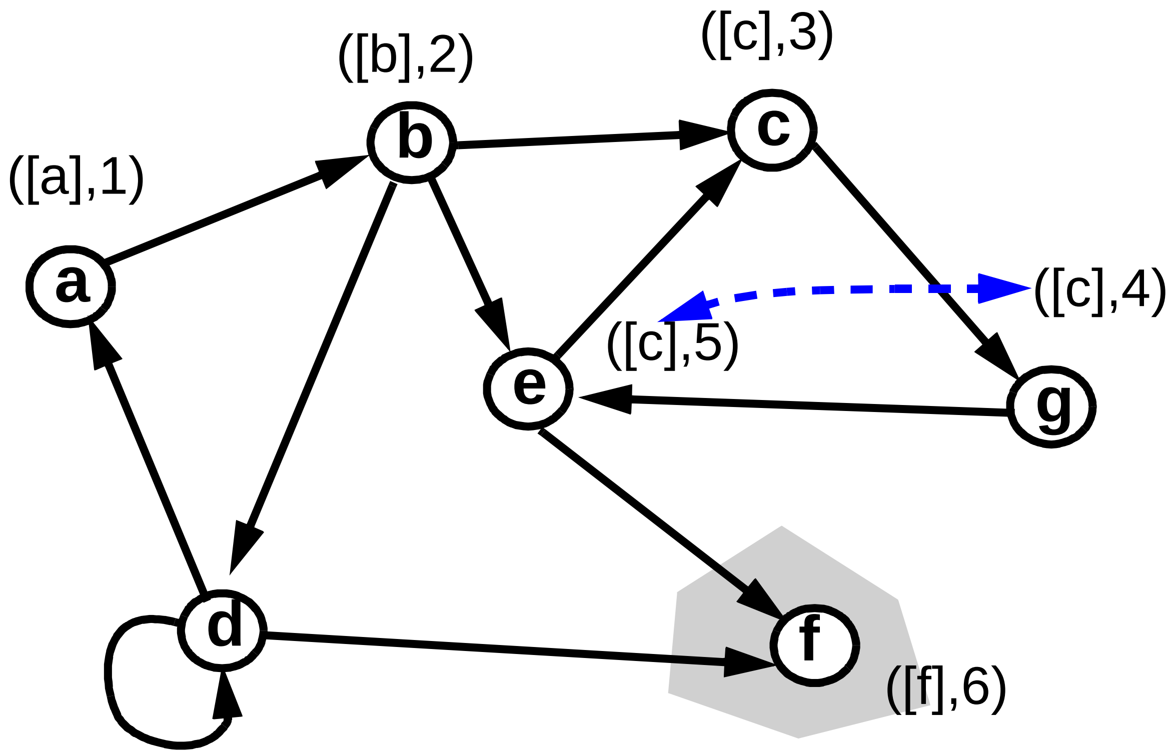
  - Looks for cycles in graph
  - Cycles are shrinked to a single node
  - Example:



- Tarjan's Algorithm:
    - Depth first search
    - start at arbitrary node
    - Time complexity:

      $\mathcal{O}(|V+E|)$

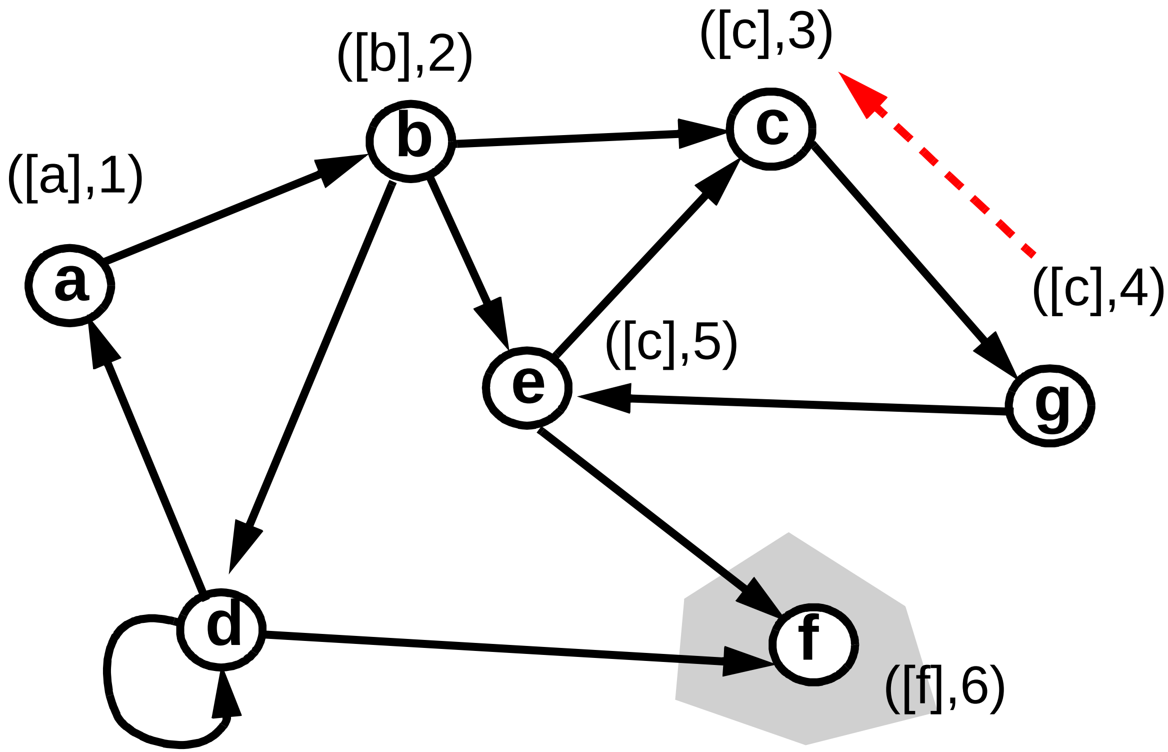
    - Looks for cycles in graph
    - Cycles are shrinked to a single node
    - Example:



- Tarjan's Algorithm:
  - Depth first search
  - start at arbitrary node
  - Time complexity:

    $\mathcal{O}(|V+E|)$

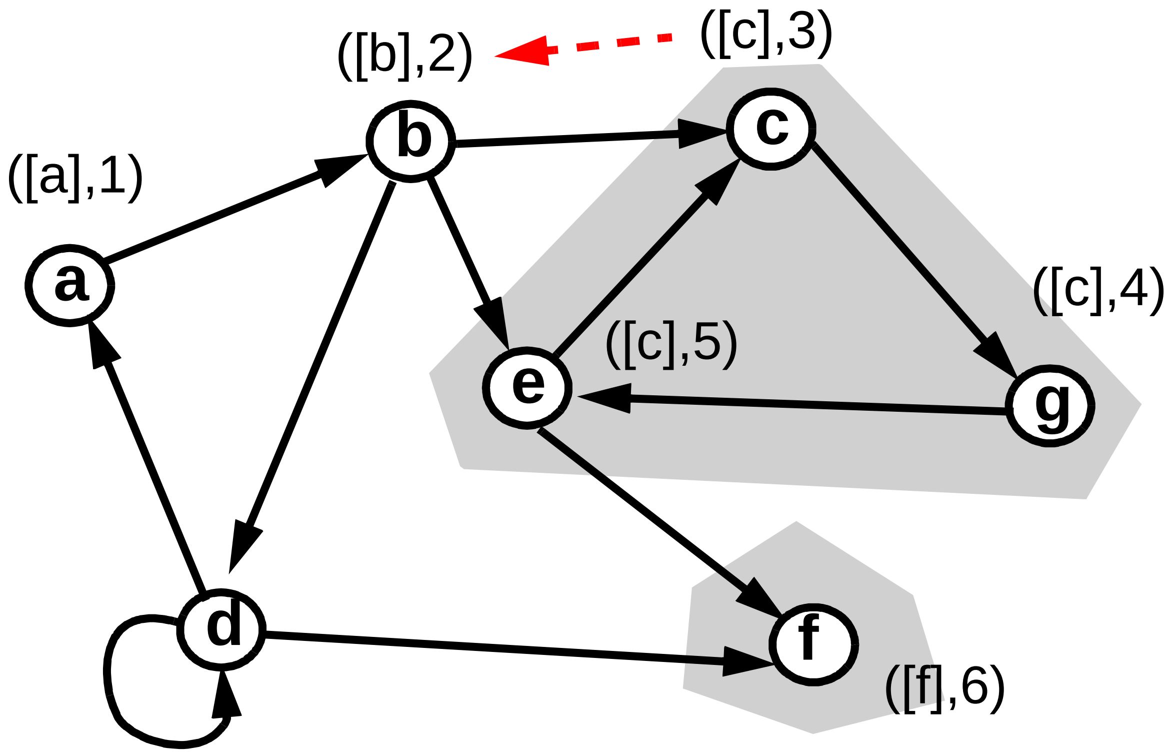
  - Looks for cycles in graph
  - Cycles are shrinked to a single node
  - Example:



- Tarjan’s Algorithm:
  - Depth first search
  - start at arbitrary node
  - Time complexity:

    $\mathcal{O}(|V+E|)$

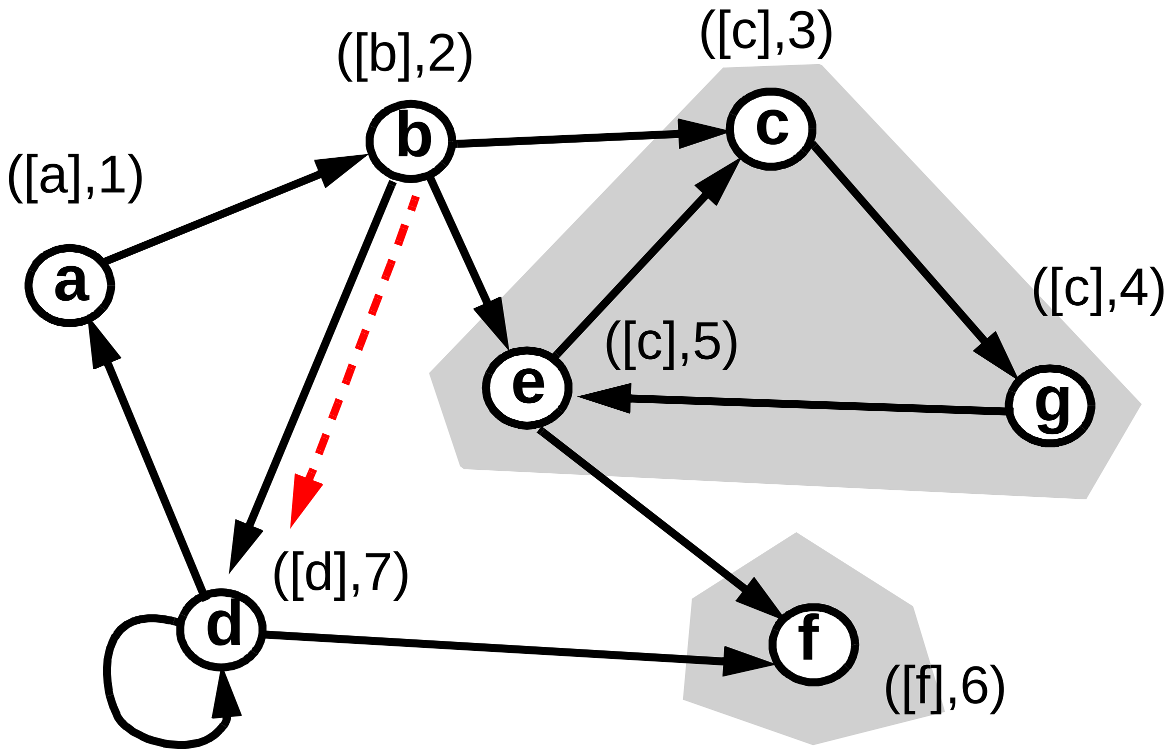
  - Looks for cycles in graph
  - Cycles are shrinked to a single node
  - Example:



- Tarjan's Algorithm:
  - Depth first search
  - start at arbitrary node
  - Time complexity:

    $\mathcal{O}(|V+E|)$

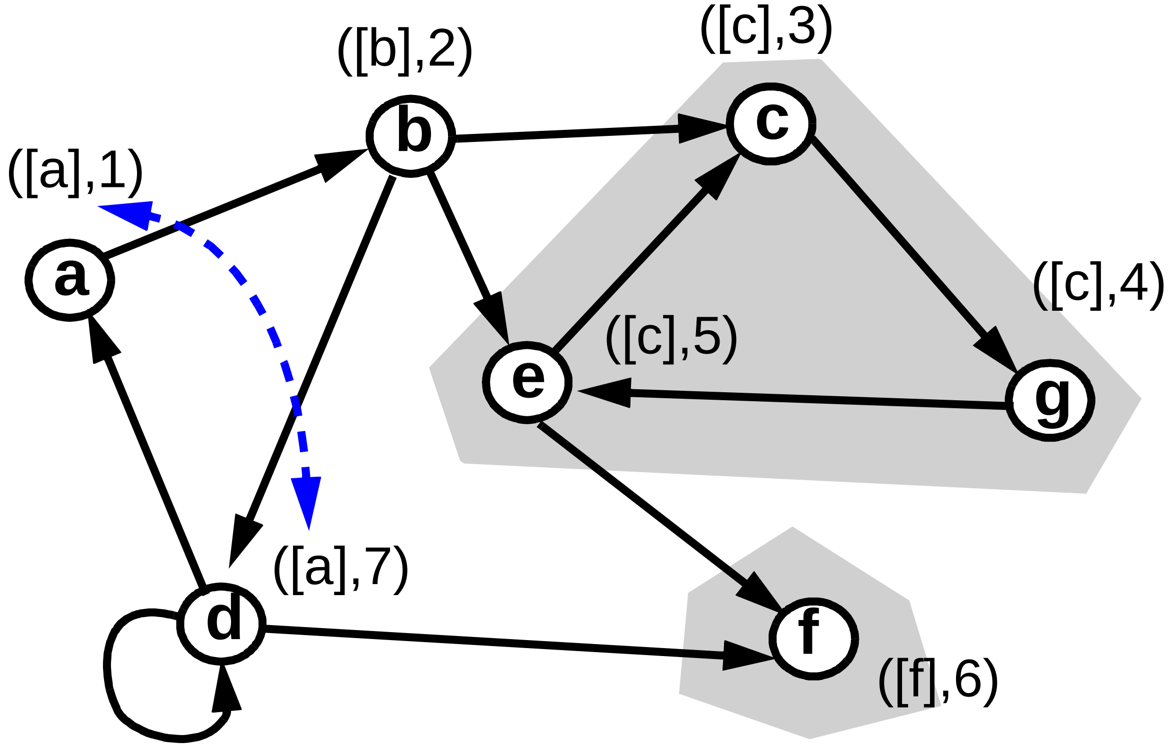
  - Looks for cycles in graph
  - Cycles are shrinked to a single node
  - Example:



- Tarjan's Algorithm:
  - Depth first search
  - start at arbitrary node
  - Time complexity:

    $\mathcal{O}(|V+E|)$

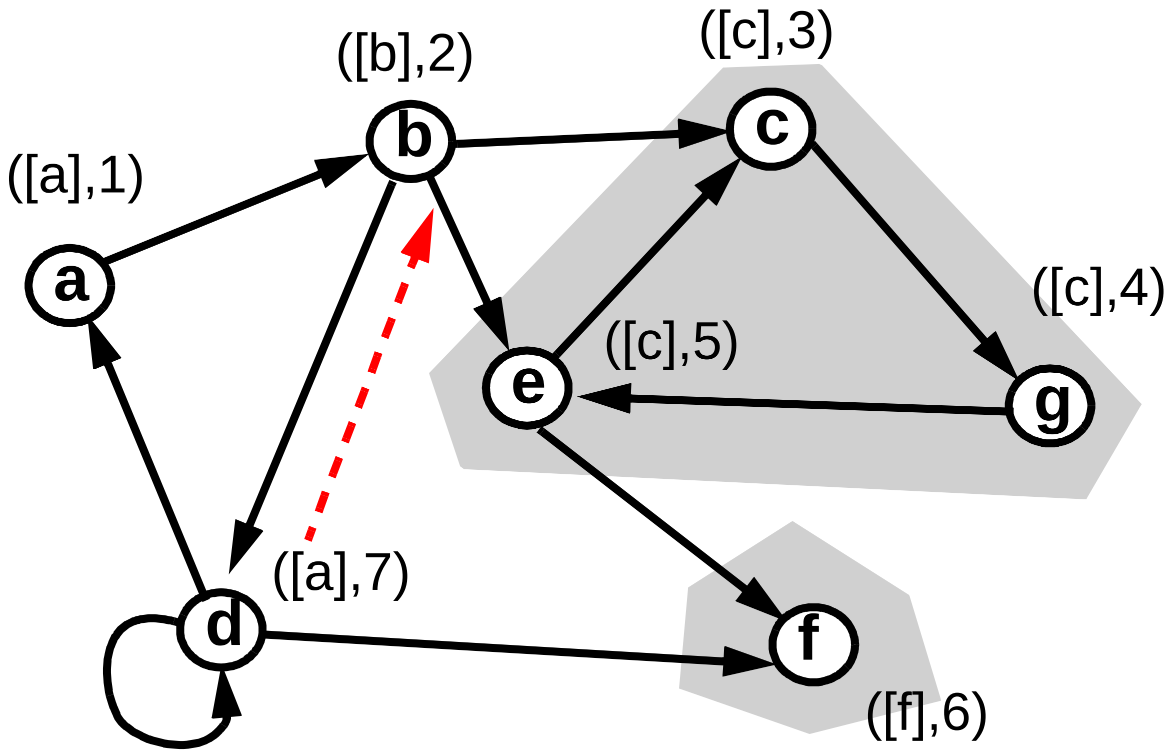
  - Looks for cycles in graph
  - Cycles are shrinked to a single node
  - Example:



- Tarjan's Algorithm:
  - Depth first search
  - start at arbitrary node
  - Time complexity:

    $\mathcal{O}(|V+E|)$

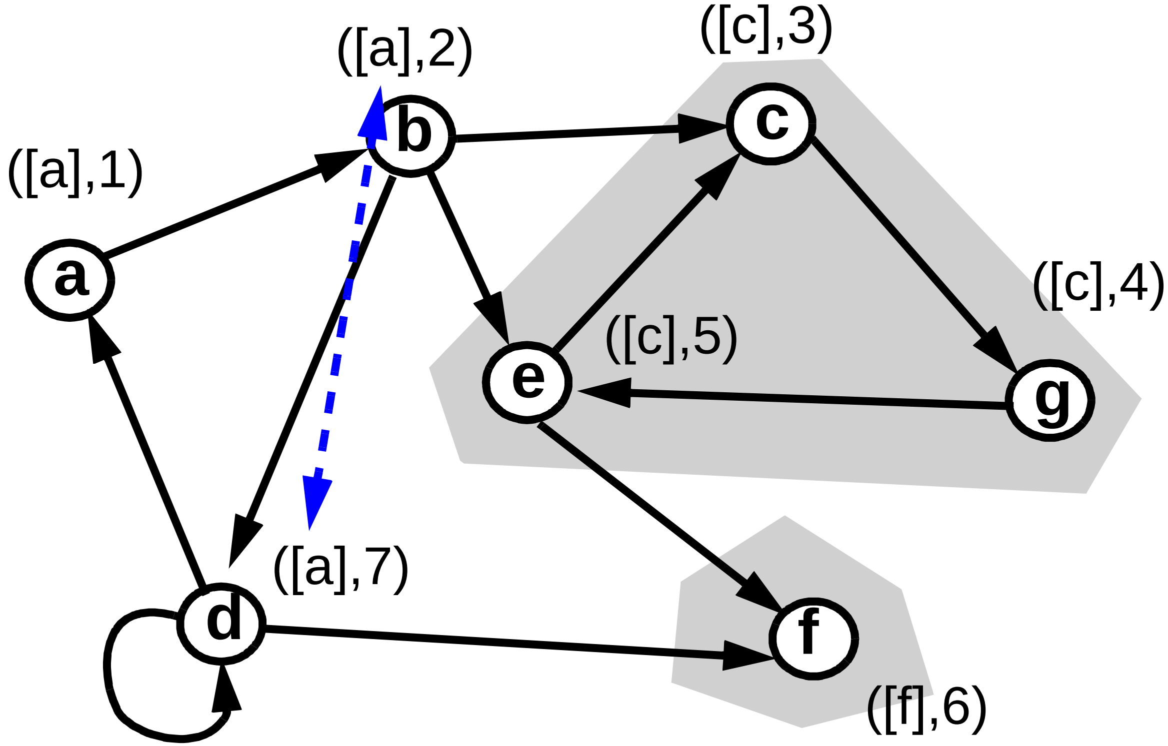
  - Looks for cycles in graph
  - Cycles are shrinked to a single node
  - Example:



- Tarjan's Algorithm:
  - Depth first search
  - start at arbitrary node
  - Time complexity:

    $\mathcal{O}(|V+E|)$

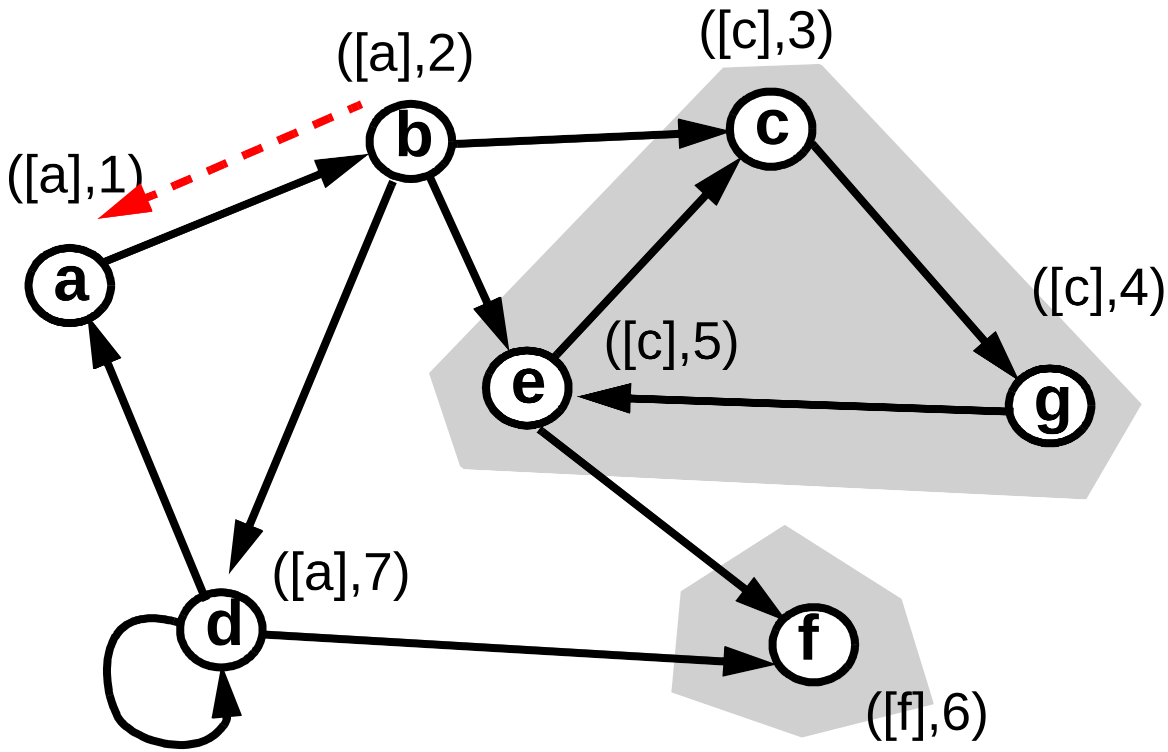
  - Looks for cycles in graph
  - Cycles are shrinked to a single node
  - Example:



- Tarjan's Algorithm:
  - Depth first search
  - start at arbitrary node
  - Time complexity:

    $\mathcal{O}(|V+E|)$

  - Looks for cycles in graph
  - Cycles are shrinked to a single node
  - Example:



- Tarjan's Algorithm:
  - Depth first search
  - start at arbitrary node
  - Time complexity:

    $\mathcal{O}(|V+E|)$

  - Looks for cycles in graph
  - Cycles are shrinked to a single node
  - Example:

End