Strongly Connected Components

Andreas Klappenecker

Undirected Graphs

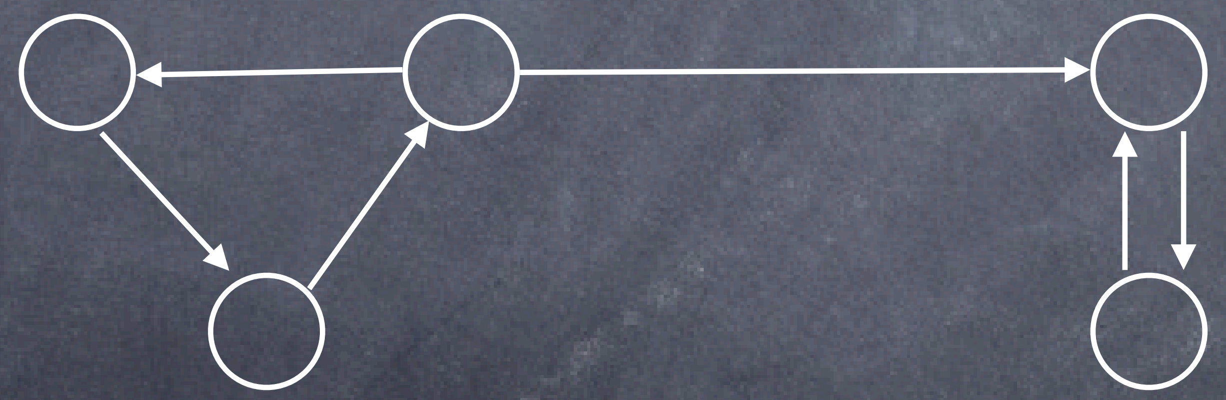
An undirected graph that is not connected decomposes into several connected components.

Finding the connected components is easily solved using DFS. Each restart finds a new component - done!

Directed Graphs

In a directed graph $G=(V,E)$, two nodes u and v are strongly connected if and only if there is a path from u to v and a path from v to u.

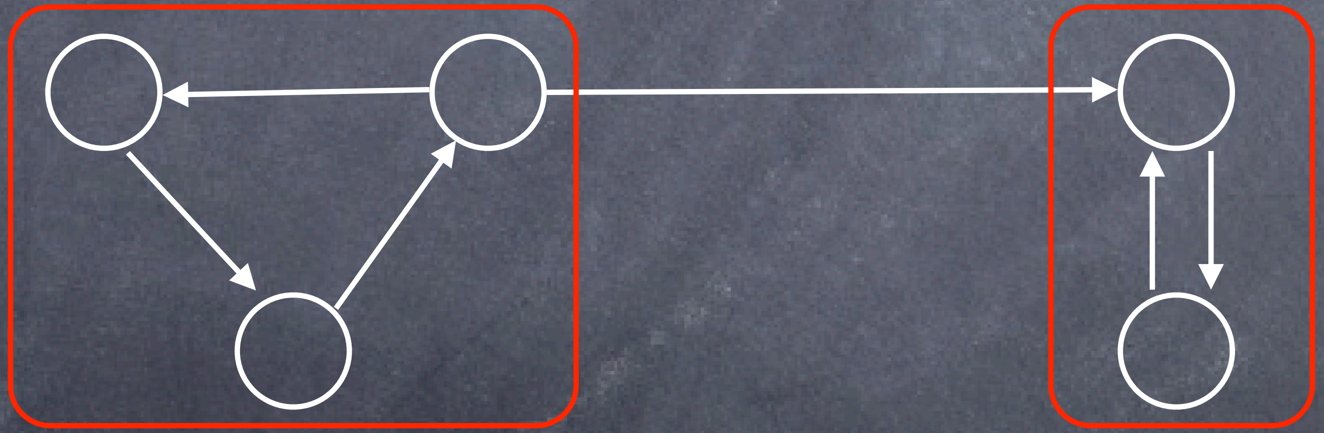
The strongly connected relation is an equivalence relation. Its equivalence classes are the strongly connected components.



Directed Graphs

In a directed graph G=(V,E), two nodes u and v are strongly connected if and only if there is a path from u to v and a path from v to u.

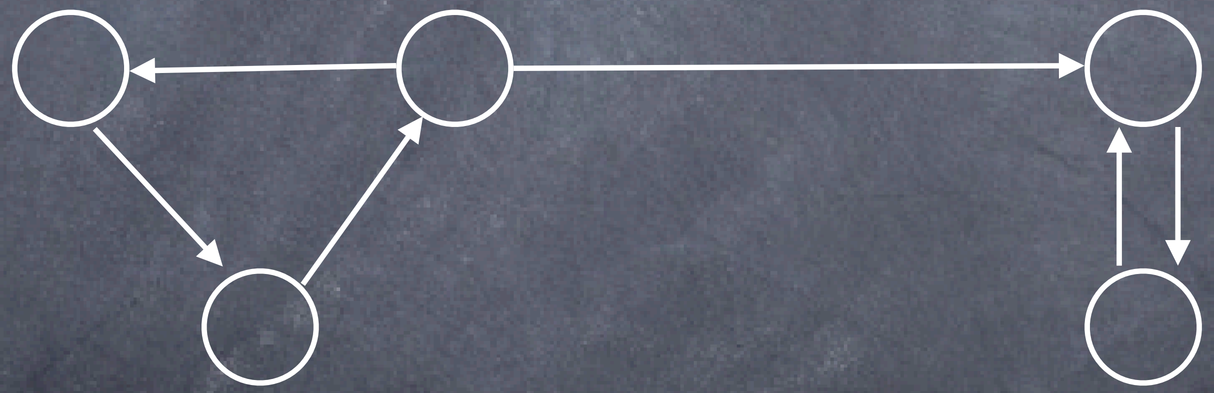
The strongly connected relation is an equivalence relation. Its equivalence classes are the strongly connected components.



Every node is in precisely one strongly connected component, since the equivalence classes partition the set of nodes.

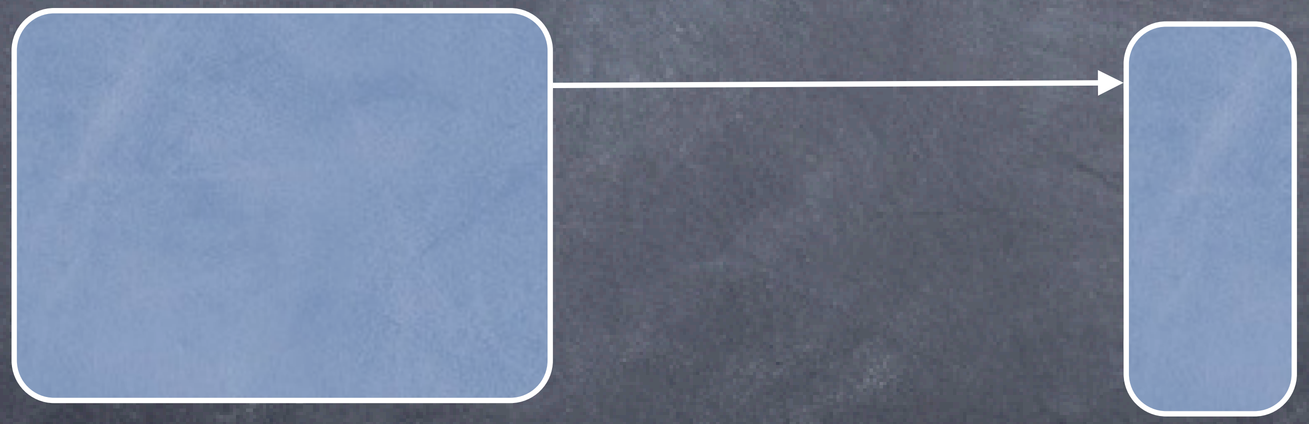
Component Graph

Take a directed graph G=(V,E) and let $\equiv$ be the strongly connected relation. Then we can define a graph $G^{SCC} = (V/\equiv, E_\equiv)$, where the nodes are the strongly connected components of G and there is an edge from component C to component D iff there is an edge in G from a vertex in C to a vertex in D.



Component Graph

Take a directed graph $G=(V,E)$ and let $\equiv$ be the strongly connected relation. Then we can define a graph $G^{SCC} = (V/\equiv, E_\equiv)$, where the nodes are the strongly connected components of G and there is an edge from component C to component D iff there is an edge in G from a vertex in C to a vertex in D.



Directed Graphs

Let G be a directed graph. Then G^{scc} is a directed acyclic graph.

[Indeed, the components in a cycle would have been merged into single equivalence class.]

Interesting decomposition of G: G^{scc} is a directed acyclic graph, and each node is a strongly connected component of G.

Terminology

In a directed acyclic graph, a node of in-degree 0 is called a source node and a node of out-degree 0 is called a sink node.

Each directed acyclic graph has at least one source node and at least one sink node.

Property 1

If depth-first search of a graph is started at node u, then it will get stuck and restarted precisely when all nodes that are reachable from u are visited.

In particular, if we start depth-first search at a node v in G that is in a component C that happens to be a sink in G^{SCC}, then it will get stuck precisely after visiting all the nodes of C.

Thus, we have a way of enumerating a strongly connected component given that it is a sink component.

Property 2

The node v in G with the highest final[v] timestamp in depth-first search belongs to a start component in G^{SCC}.

Property 2

The node v in G with the highest final[v] timestamp in depth-first search belongs to a start component in G^{scc}.

We wanted to find a sink component, but we merely found a way to find a node in a start component.

Reversed Graph Trick

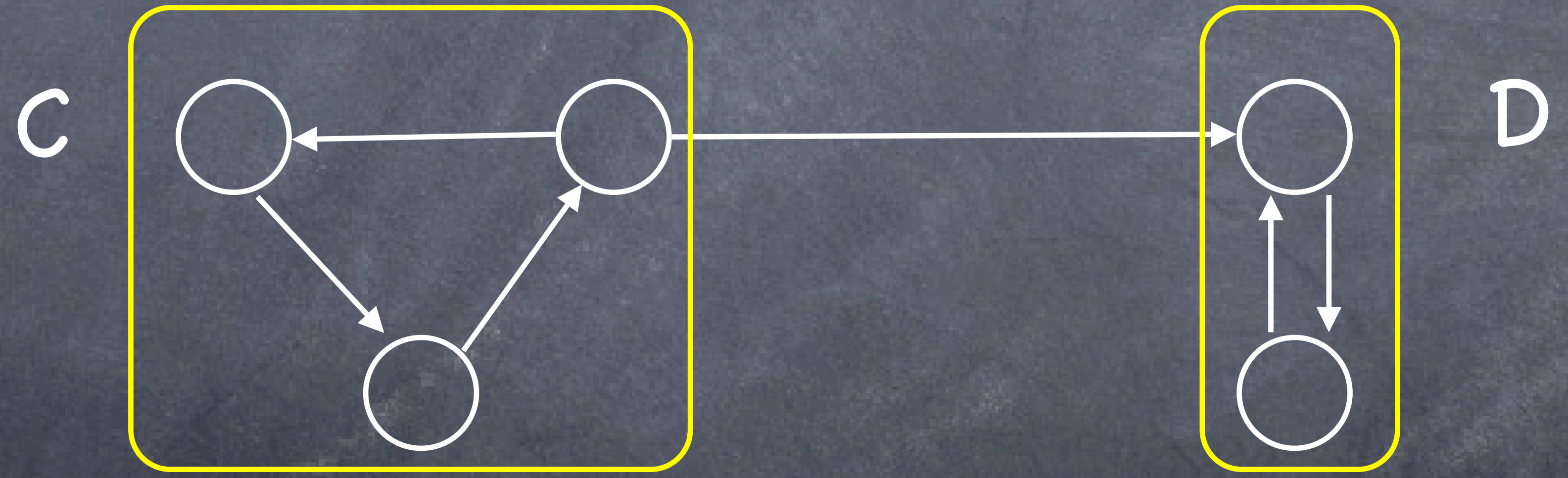
Given the graph $G=(V,E)$ consider its reversed graph $G_R=(V,E_R)$ with $E_R = \{ (u,v) \mid (v,u) \text{ in } E \}$, so all edges are reversed.

Then G_R has the same strongly connected components as G.

If we apply depth first search to G_R, then the node v with the largest finishing time belongs to a component that is a sink in G^{scc}.

Property 3

Let C and D be strongly connected components of a graph. Suppose that there is an edge from a node in C to a node in D. Then the vertex in C that is visited first by depth first search has larger final[v] than any vertex in D.



Corollary

Arranging the strongly connected components of a directed graph in decreasing order of the highest finish time in each component topologically sorts the strongest connected components of the graph.

[Well, this is just topological sorting applied to the directed acyclic graph G^{scc}.]

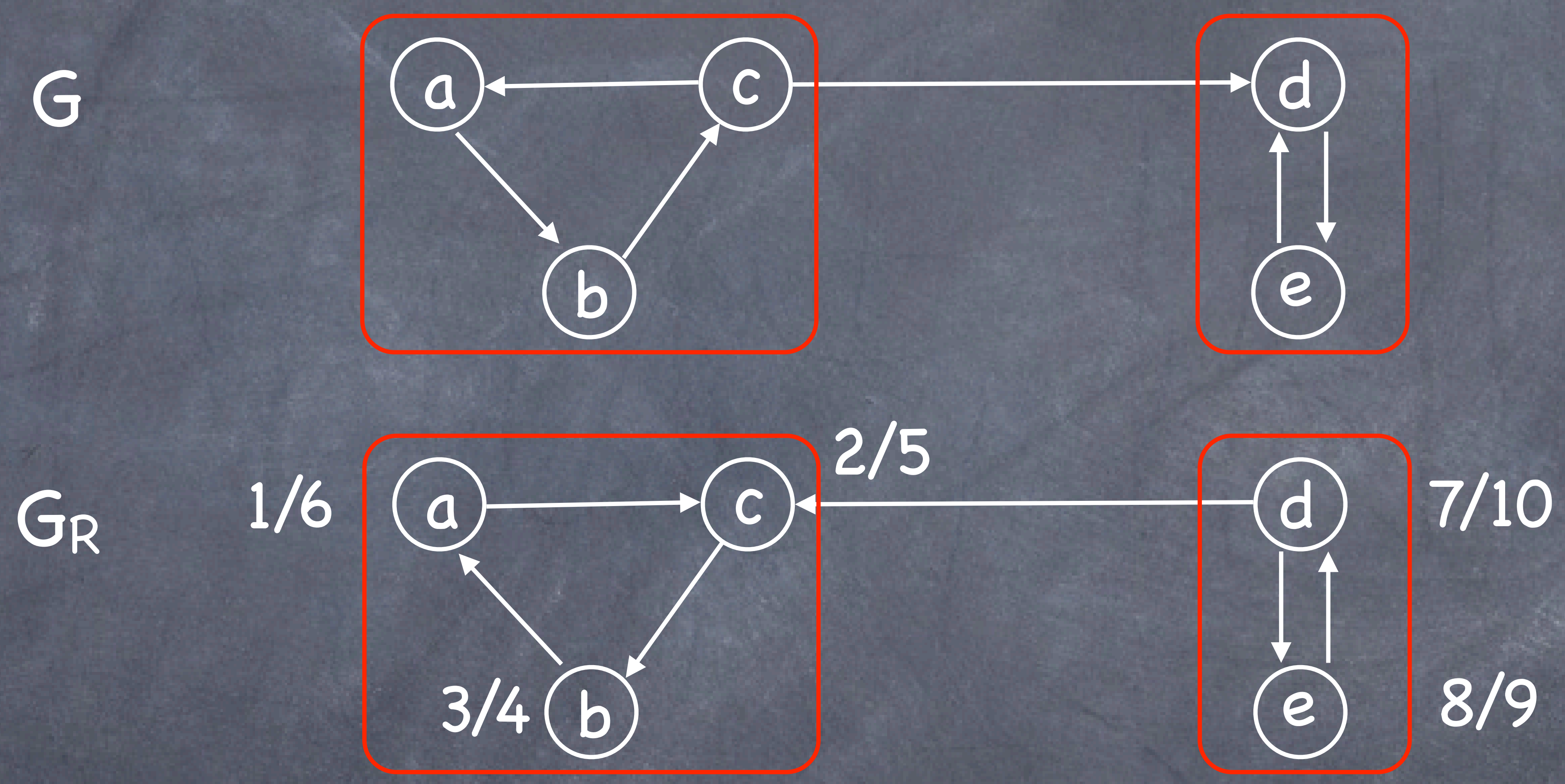
SCC Algorithm

1) Perform depth first search on G_R.

2) Perform depth first search on G in decreasing order of the final times computer in step 1).

Complexity: O(V+E)

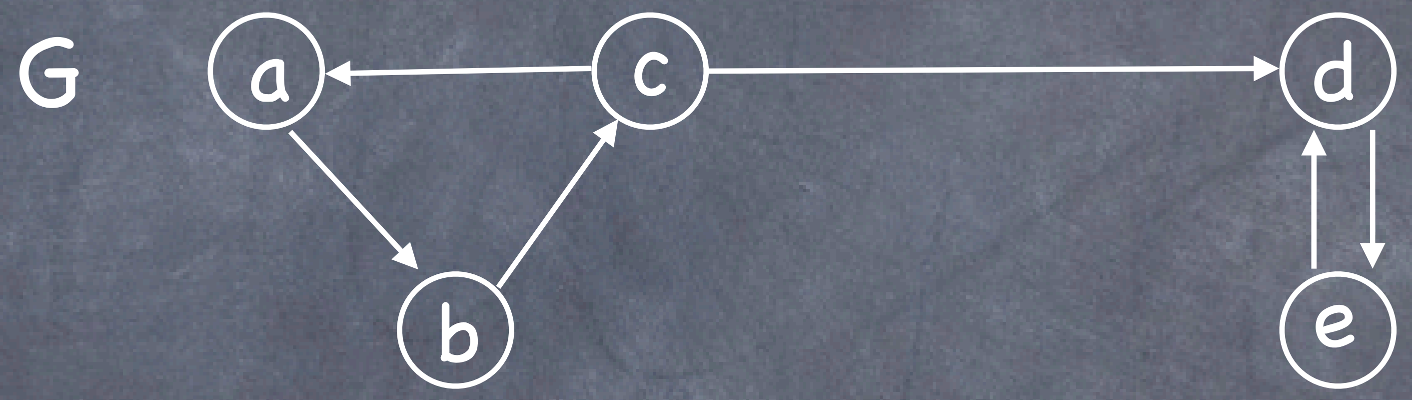
Example



Order by decreasing finishing time: d, e, a, c, b.

Example

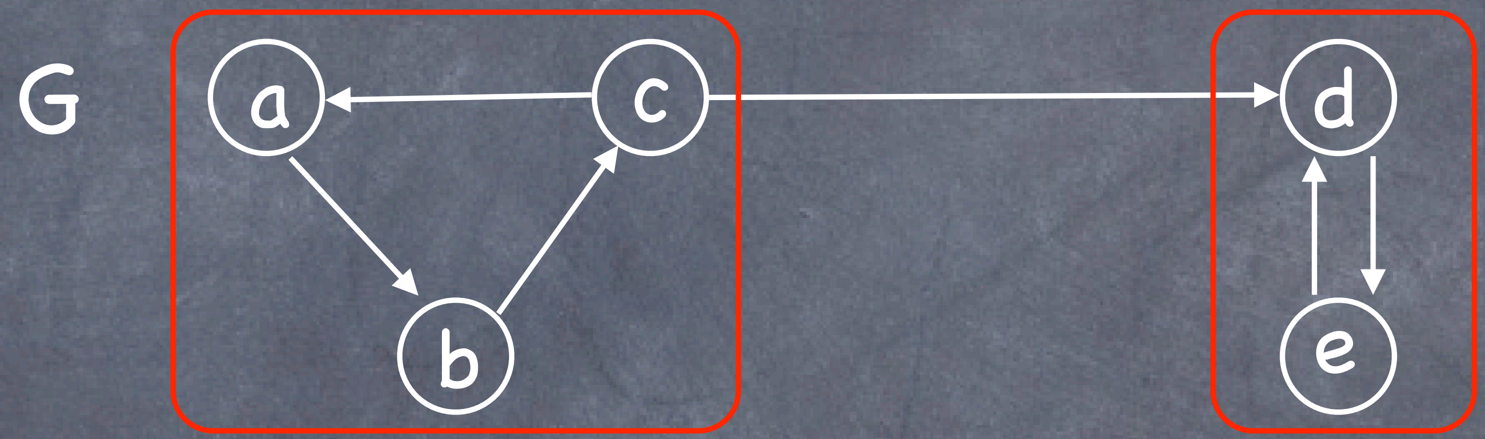
Order by decreasing finishing time: d, e, a, c, b.



Run DFS on G (use above order from G_R): {d, e}, {a, b, c}.

Example

Order by decreasing finishing time: d, e, a, c, b.



Run DFS on G (use above order from G_R): {d, e}, {a, b, c}.

References

I followed lecture notes by Umesh Vazirani in the preparation of these slides.