Parma University, Math. Deptartment.

Algorithms and Data Structures
Strongly connected components in directed graphs

Marco Pellegrini

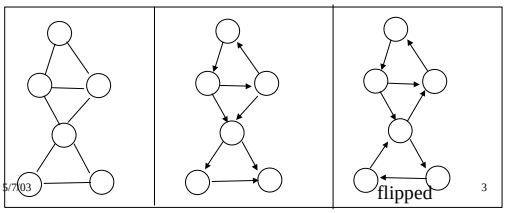
Spring 2002, Lecture 18

Summary

- This lecture:
  Strongly connected components of directed graphs.
  Kosaraju-Sharir's algorithm
  proof of correctness.
- Material: CLR 23.5

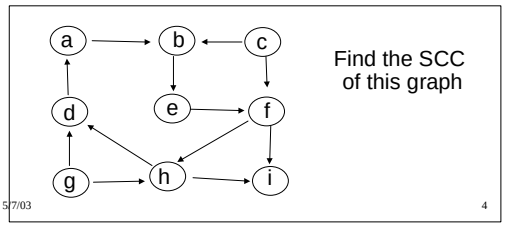
Strongly connected components.

- We saw that an un-directed graph can be split into connected components, sets of vertices and edges such that there is a path between any two nodes within the component.
- For directed-graphs things are more subtle because of the orientation of the edges.



Strongly connected components

- A strongly connected component of a directed graph G is a set of vertices V' in V such that there is a path between any two vertices u and v in V' that uses only vertices in V'.
- Moreover such set V' is maximal, meaning that we cannot add other vertices and satisfy the path condition.

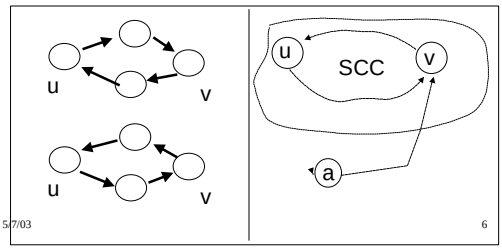


Applications:

- Planning of one-way streets in a city.
- Detecting groups of constraints that must be solved simultaneously. Suppose you have a node for each variable in a problem and an arrow when one variable depend on the value of the other. A SCC denote a set of variables and equations that we must solve together.

Properties of SCC.

- If u and v are in the same SCC of G(V,E) then u and v are in the same SCC in the graph transpose(G) obtained by reversing the direction of any edge.
- If u and are in the same SCC of G, than any path between u and v cannot get out of that SCC.

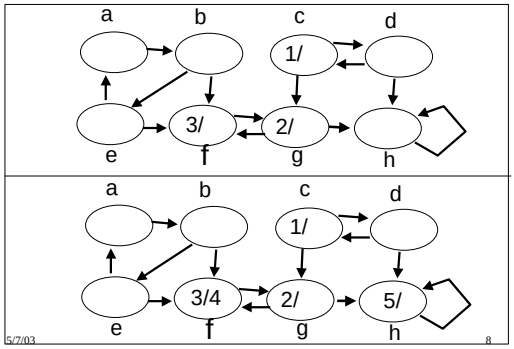


Kosaraju-Sharir's algorithm for SCC

- The idea is to use DFS on G to label the nodes.
- Then transpose (G), which does not change the SCC's, and run DFS again. Very simple, although it is not clear immediately why it should work.
- Strongly-con-comp(G)
  (1) call DFS(G), record for each u in V the finish time f[u].
  (2) compute transpose(G)
  (3) sort V by decreasing values of f[u]
  (4) call DFS(transpose(G))
  (5) output the DFS forest.
- Claim: The nodes in each tree of the DFS forest form a SCC.

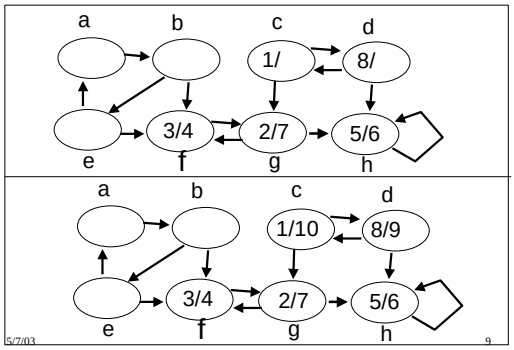
5/7/03 7

Example 1



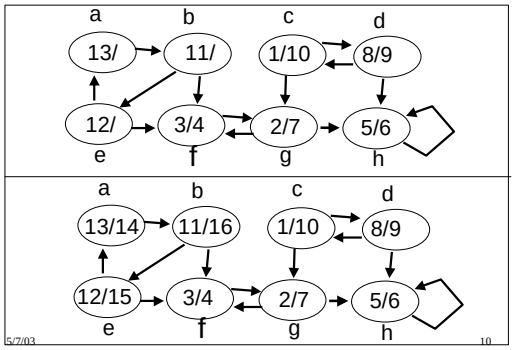
5/7/03 8

Example 2



5/7/03 9

Example 3



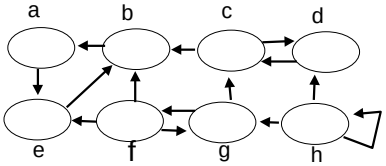
5/7/03 10

Example 4

a	b	c	d	e	f	g	h	
14	16	10	9	15	4	7	6	f[u]

b	e	a	c	d	g	h	f
16	15	14	10	9	7	6	4

Tr(G)



5/7/03 11

Exercise

- Complete the simulation of strongly-con-com(G) on the graph of the example. Show how DFS(Tr(G) marks the nodes when we use as order of the nodes the decreasing f[u] (finish time) order.
- For your reference I've replicated the code for DFS in the next two slides, but you should be able to do it without looking at the code.

* Does the DFS-forest represent the SCC's?

5/7/03 12

Code for DFS

- Input G=(V,E) in adj list representation.
- Output: arrays d[1..n], f[1..n], and p[1..n].
- Auxiliary data: color[1..n], time.
- DFS(G)
  FOR each vertex u in V(G) DO
  color[u] := white
  p[u] := NIL
  END FOR
  time := 0
  FOR each vertex u in V(G) DO
  IF color[u] = white
  THEN DFS-vist(u)
  END IF
  END FOR

DFS-visit

- DFS-visit(u)
  color[u] := grey
  time := time + 1
  d[u] := time
  FOR each v in adiacency-list(u) DO
  IF color[v] = white
  THEN p[v] := u
  DFS-vist(v)
  END IF
  END FOR
  color[u] := black
  time := time +1
  f[u] := time

Proof of correctness.

(Overview)

Theorem A.
In any DFSearch the nodes of a SCC are in the same tree of the resulting DFS-forest.

Theorem B
In G=(V,E) two vertices v and u are in the same SCC *if and only if* ϕ[u]= ϕ[v] in a DFS search of G.

Theorem C
The Kosaraju-Sharir algorithm correctly finds all SCC in G.

Proof of theorem A

Theorem A.
In any DFS the nodes of a SCC are in the same tree of the DFS-forest.

Proof.

This is easy to see. As DFS finds the first node of a SCC, then all other nodes in the same SCC are reachable from it and are still white so a single DFS-visit is sufficient to find them all.

The forefather of a node

Consider the numbering after the first call of DFS in the program Strongly-con-com.

For a node u, the forefather of u (denoted (ϕ(u)) is the node w reachable from u and with largest value of f[].

Esercise: Find ϕ[e], ϕ[f], ϕ[h] in the example.

Lemma: ϕ[u] has the same f[] value as u or a larger one.

Proof of Theorem C

Theorem C
The Kosaraju-Sharir algorithm correctly finds all SCC in G.

Proof. Let us see how Theorem B helps in explaining the algorithm.

Look at vertex r with the maximum value of f[] after the DFS of line 1. Since this is the maximum f[] globally it must be the FF of the SCC containing r.

So to discover that SCC we just have to find all vertices in G that can reach r.

But this is the same as taking Transpose(G) and look for all the nodes that r can reach.

So DFS of Transpose(G) starting with the node r of maximum value of f[] finds SCC(r).

Let q be the node not visited yet that has the maximum value of f[] among the nodes in V/SCC(r).

We can repeat the same reasoning on q and conclude that DFS-visit starting on q on Transpose(G) finds the SCC(q) containing q.

Inductively we can easily show that second DFS of the code finds all SCC of G.

This proof is made more formal in Theorem 23.17. CLR.

Proof of Theorem B (part 1)

Theorem B
In G=(V,E) two vertices v and u are in the same SCC *if and only if* ϕ[u]= ϕ[v] in a DFS search of G.

Proof. (only if part) if u reaches v in G then f[ϕ[u]] ≥f[ϕ[v]].

This is easy since ϕ[u] can only do better than ϕ[v] or be ϕ[v].

But in a SCC all nodes can reach any other node so for any u and v in a SCC f[ϕ [u]]=f[ϕ[v]] which implies ϕ[u]= ϕ[v] since all finishing times are different.

Proof of Theorem B (if part)

Suppose that u and v are such that ϕ[u]= ϕ[v]. To show that u and v are in the same SCC it is sufficient to show that u and ϕ[u] are in the same SCC (and similarly that v and ϕ[v] are in the same SCC).

By the definition of FF we know there is a path from u to ϕ[u].

The difficult part is to show that there is a path also from ϕ[u] to u.

We do so by showing that after the first DFS of the code, ϕ[u] is an ancestor of u in a DFS-tree.

To prove this part we exploit the time-stamping and the coloring of DFS.

Easy case: u= ϕ[u] we are done. Otherwise u≠ ϕ[u].

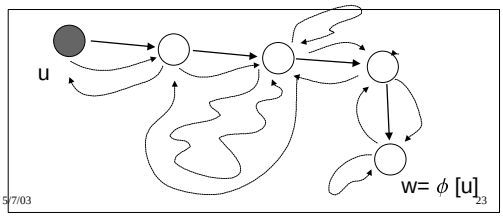
What is the color of ϕ[u] at the time d[u] when DFS sees u for the first time?

If ϕ[u] is grey than we are done since all grey nodes form a path pointing towards the last visited node u.

If ϕ[u] is black, then its finishing time comes earlier that of u, contradicting the definition of ϕ[u]. So ϕ[u] cannot be black.

Can ϕ[u] be white? No

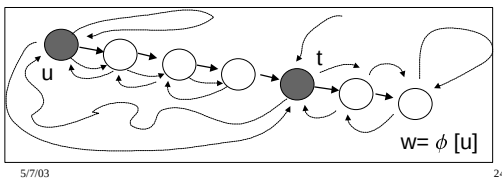
Case 1: The path from u to ϕ[u] is made only of white nodes.
In his case we visit u before ϕ[u] and ϕ[u] becomes a descendant of u.
In this case the parenthesis theorem of DFS states that f[ϕ[u]] < f[u], contradicting the definition of a forefather.



Can ϕ[u] be white? No

Case 2. ϕ[u] is white and there are non-white nodes on the path from u to ϕ[u]. Let t be the last one on the path from u to ϕ(u). (t must be grey!).

Since t is grey and there is a white path from t to ϕ[u], ϕ[u] is a descendant of t, but if it is so by the parenthesis theorem f[t] > f[ϕ[u]] and there is a path from u and t. This contradicts our choice of ϕ[u]. End of the proof.



Conclusions and HW

- In directed graph, strongly connected components are important sub-structures.
- We can use DFS (twice) to compute the SCC.
- HW5.1: Analyze the running time of Strongly-conn-comp. If it is not O(|V| +|E|) modify the algorithm so to make it run in that time.