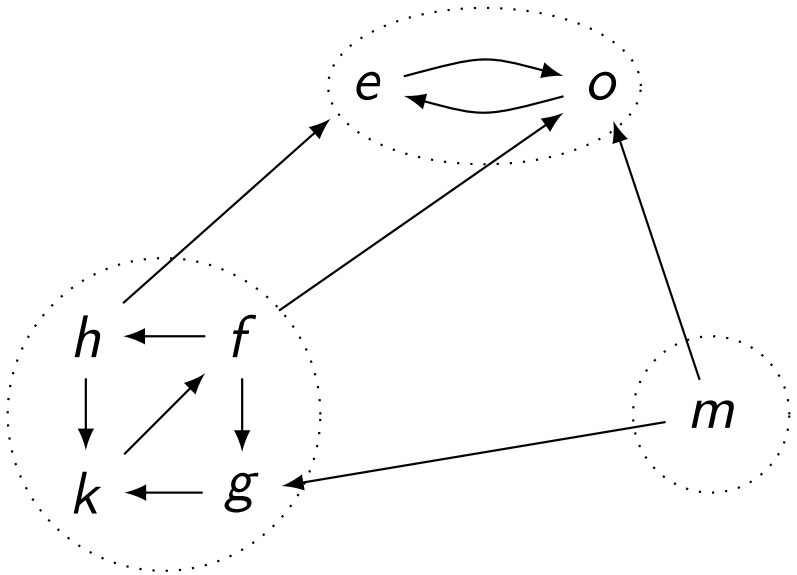
CSCB63 – Design and Analysis of Data Structures

Anya Tafliovich[1]

[1] based on notes by Anna Bretscher and Albert Lai

strongly connected component (SCC)

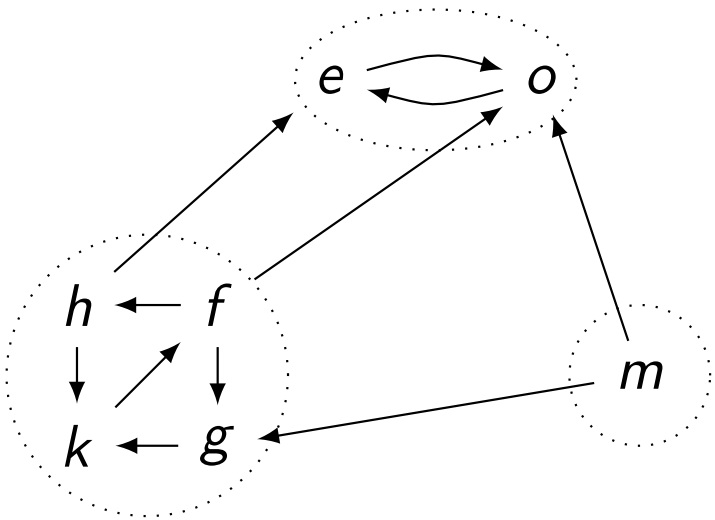
Strongly connected component (SCC): maximal subset of vertices reachable from each other.



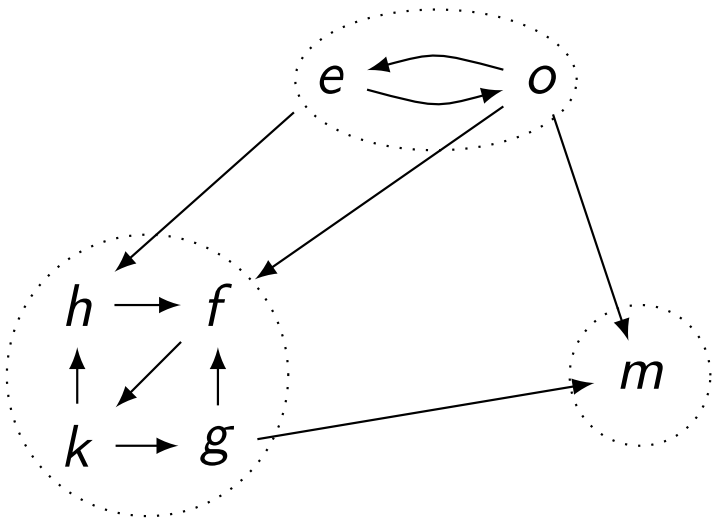
transposed graph

Transpose of G (G^{T}) means a graph with the same vertices as G and the edges are the reverse of G's.
Why is it called "transpose"?

G:



G^{T}:



G^{T} has the same strongly connected components as G's.
How much time to compute adjacency lists of G^{T}:

computing SCCs: idea

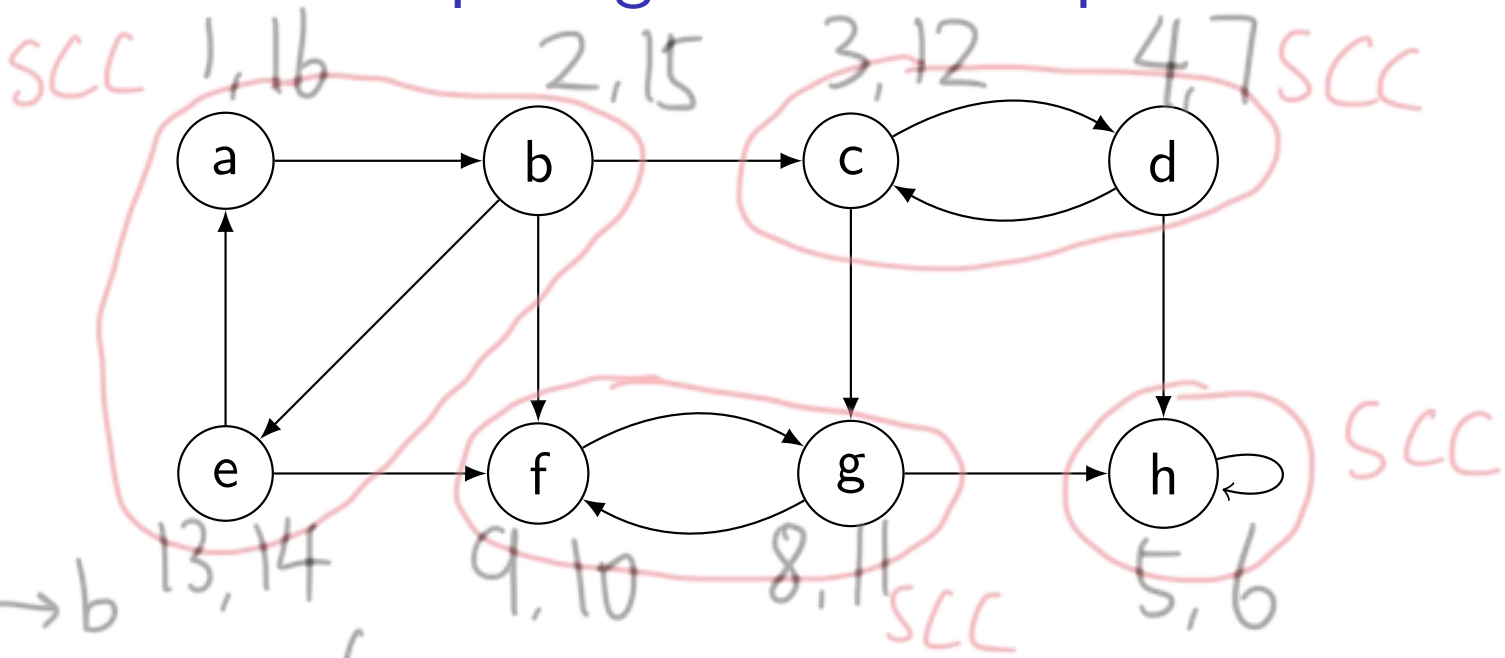
1. DFS on G
   - visit all vertices
   - store all finish times
   - accumulate vertices in reverse finish-time order
2. Compute adjacency lists of G^{T}
3. DFS on G^{T}
   - use the above order to pick start/restart vertices
4. Each tree found has the vertices of one strongly connected component.

Total time: $O(|V|+|E|)$

[traverse all vertices & edges]

start time / finish time

computing SCCs: example



G:

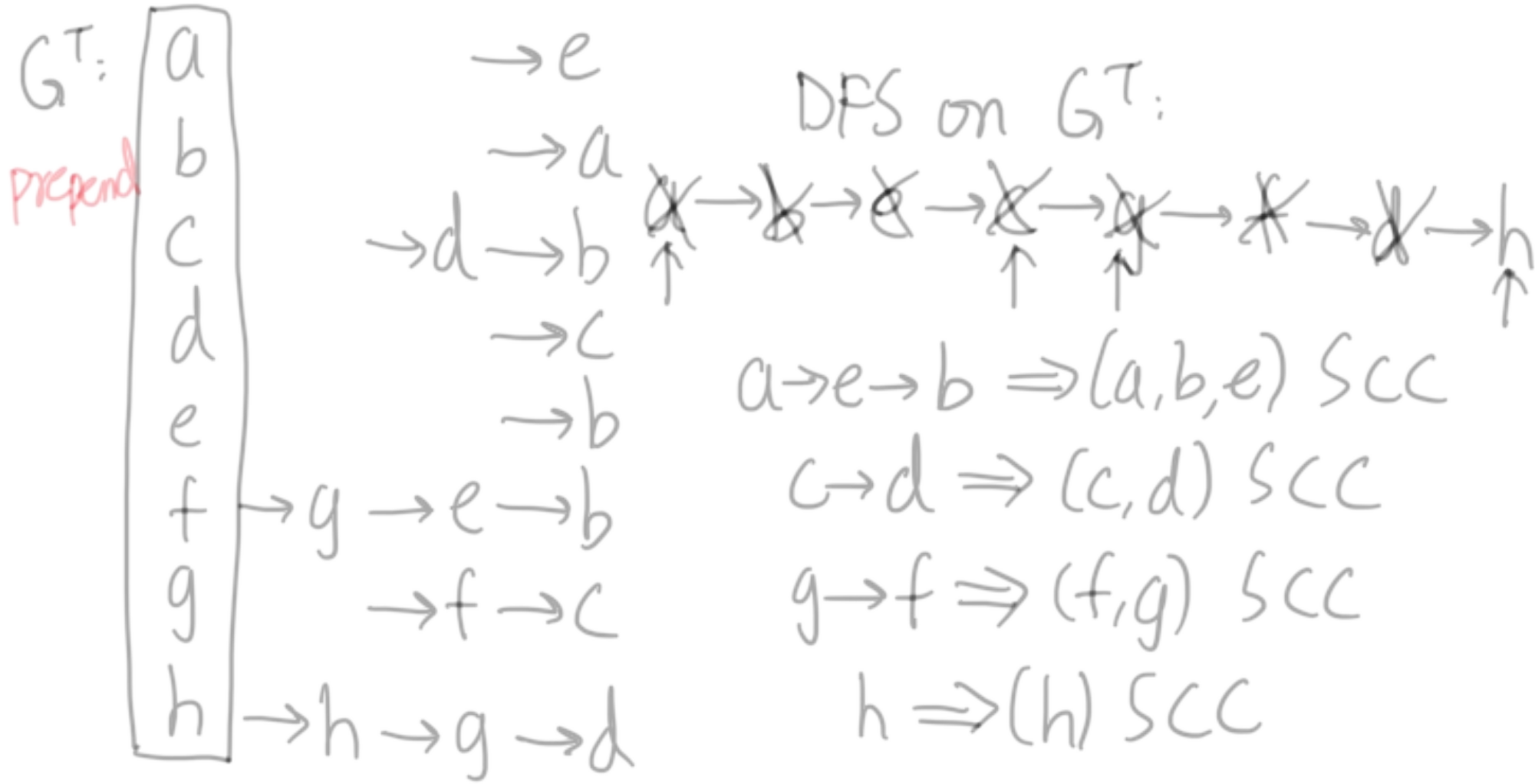
a	→ b
b	→ c → e → f
c	→ d → g
d	→ c → h
e	→ f → a
f	→ g
g	→ f → h
h	→ h

DFS G:

finish time linked list (prepend):

a → b → e → c → g → f → d → h

O(1) complexity


G^T:
prepend
a → e
b → a
c → d → b
d → c
e → b
f → g → e → b
g → f → c
h → h → g → d
DFS on G^T:
a → b → e → c → f → g → d → h
a → e → b ⇒ (a, b, e) SCC
c → d ⇒ (c, d) SCC
g → f ⇒ (f, g) SCC
h ⇒ (h) SCC

computing SCCs: DFS(G)

```
0. mark all vertices white
1. time := 0
2. R := []
3. for each vertex v:
4.    if v is white:
5.       DFS-visit(v)

6. DFS-visit(u):
7.    mark u gray
8.    for each v in adjacency list of u:
9.       if v is white:
10.         DFS-visit(v)
11.   mark u black
12.   finish-time(u) := ++time
13.   insert u at the front of R
```

computing SCCs: DFS(G^{T})

```
0. mark all vertices white
1. for each vertex v in R's order:
2.   if v is white:
3.     SCC := []
4.     DFS-visit2(v)
5.     output/record SCC

6. DFS-visit2(u):
7.   add u to SCC
8.   mark u gray
9.   for each v in u's adjacency list in G^T:
10.    if v is white:
11.      DFS-visit2(v)
12.  mark u black
```

computing SCC: proof

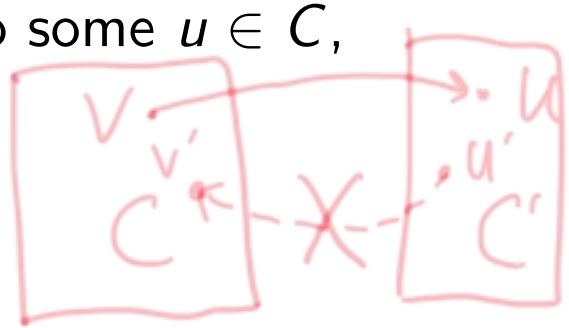
Prove: each depth-first tree found in DFS(G^T) is a SCC.

Let C and C' be distinct SCC's of G.
Define $max\_finish(C) = \max\{finish\_time(u) \mid u \in C\}$.

Proof steps:

- If some vertex $u \in C$ has an edge in G to some $v \in C'$, then $max\_finish(C) > max\_finish(C')$.
  - C discovered earlier: finish C' before V
  - C' discovered earlier: no edge from C' to C

- In G^{T}, if some vertex $v \in C'$ has an edge to some $u \in C$, then $max\_finish(C) > max\_finish(C')$.



computing SCC: proof

Proof steps (continued):

- In G^{T}, if some vertex $v \in C'$ has an edge to some $u \in C$, then $max\_finish(C) > max\_finish(C')$.
- If $max\_finish(C) > max\_finish(C')$, then in G^{T} no edge from C to C'.
- DFS(G^T):

Complete proof: textbook / exercise: by induction on the number of depth-first trees found.