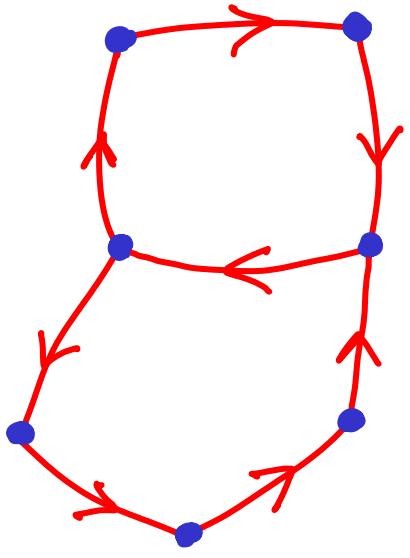
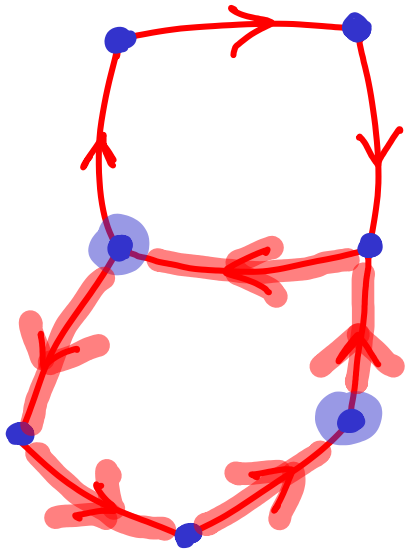
STRONGLY CONNECTED GRAPHS



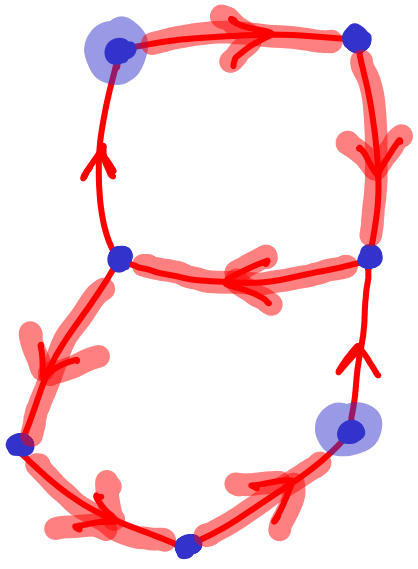
(directed) graph s.t. every vertex can be reached from every vertex.



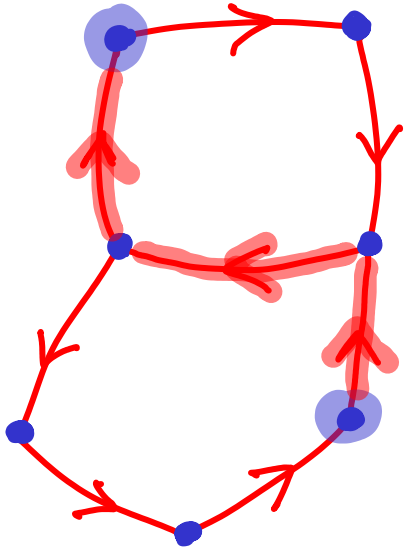
(directed) graph s.t. every vertex can be reached from every vertex.

e.g., 2 vertices can reach each other on a subgraph that is a simple cycle

→ using edges once

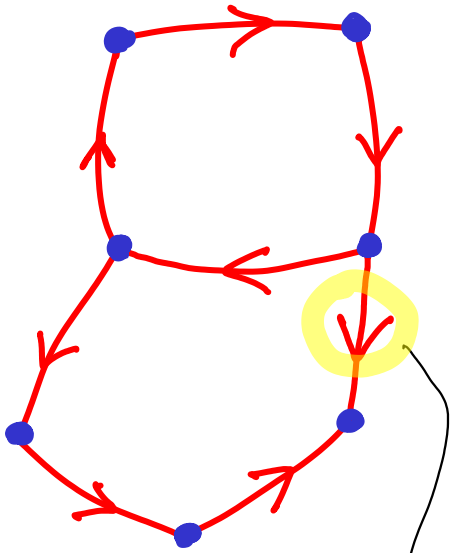


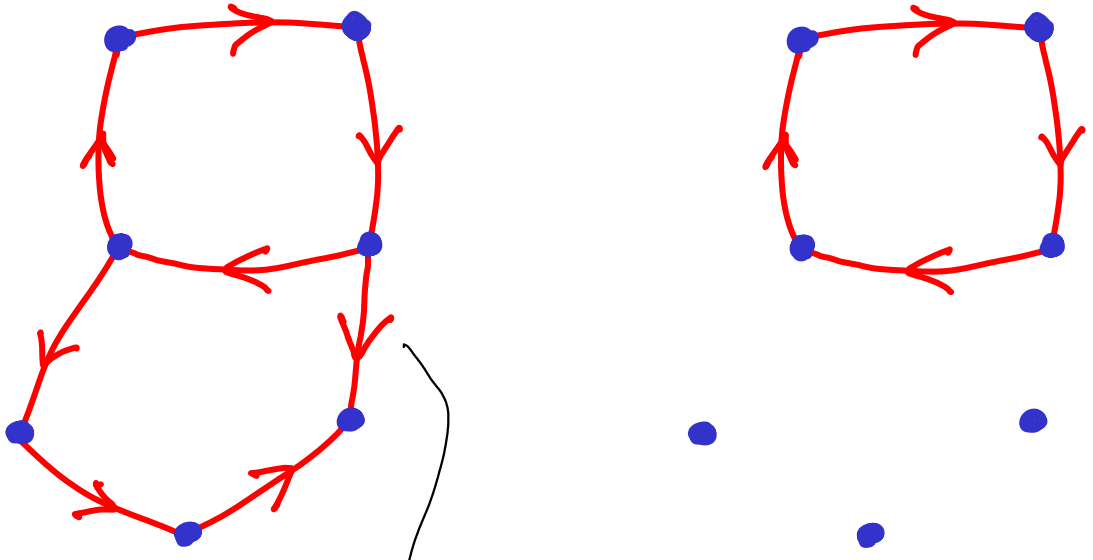
(directed) graph s.t. every vertex can be reached from every vertex.



(directed) graph s.t. every vertex can be reached from every vertex.

{ Not necessarily true that every 2 vertices reach each other with a simple cycle.

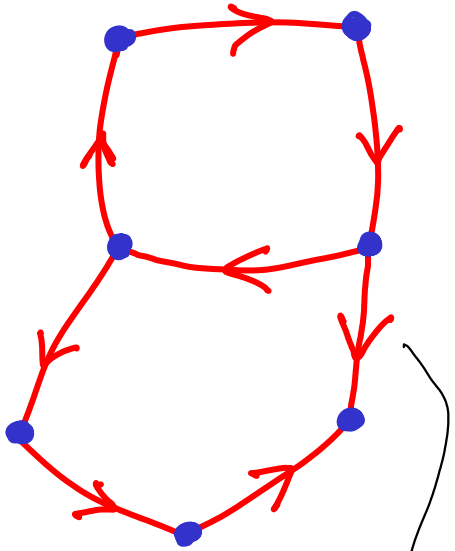

no longer
strongly
connected



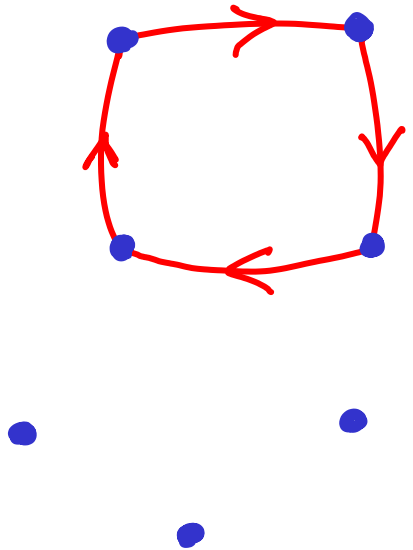
no longer strongly connected

chain reaction:
several vertices
no longer
mutually reachable

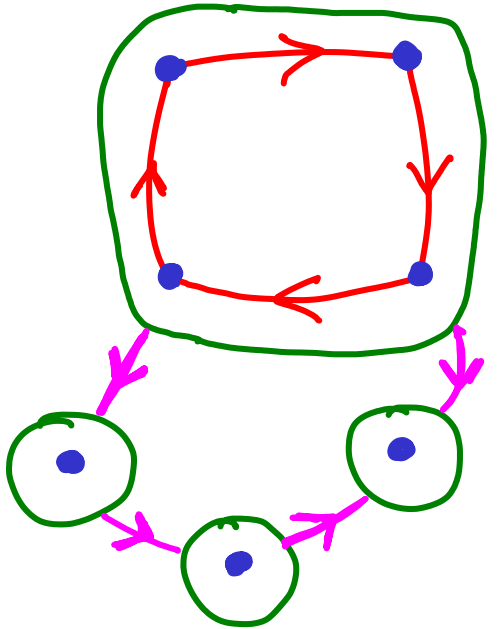
STRONGLY CONNECTED COMPONENTS



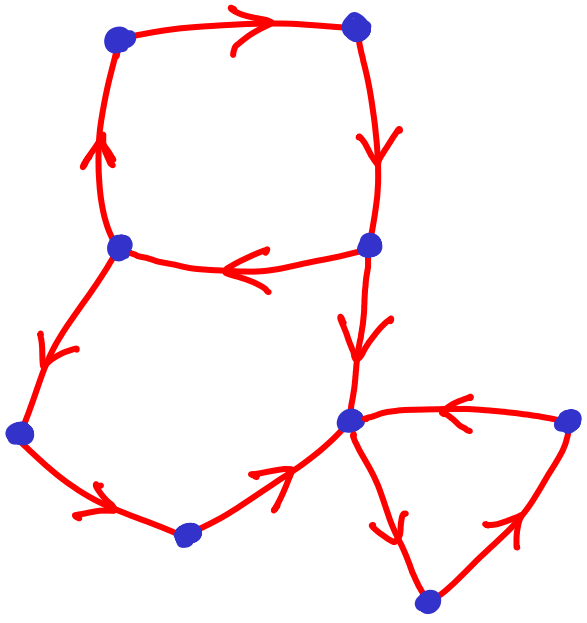
no longer strongly connected

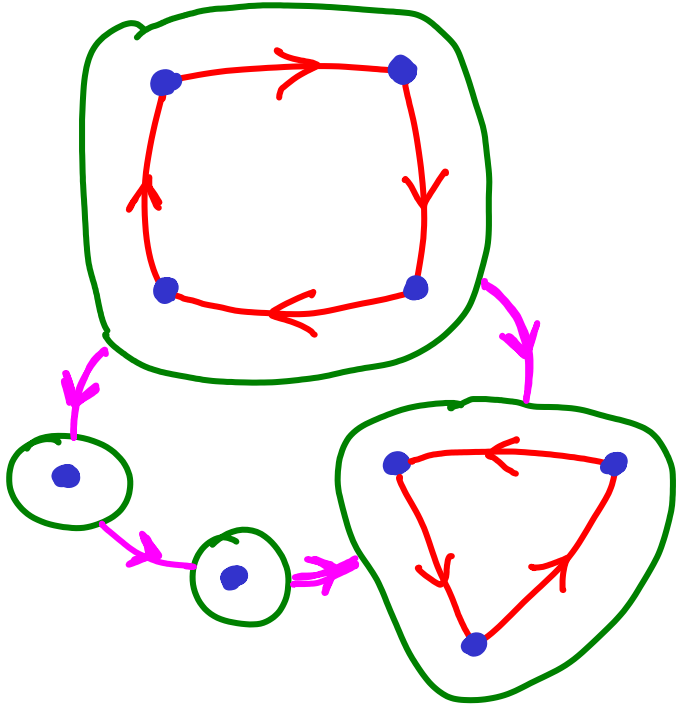


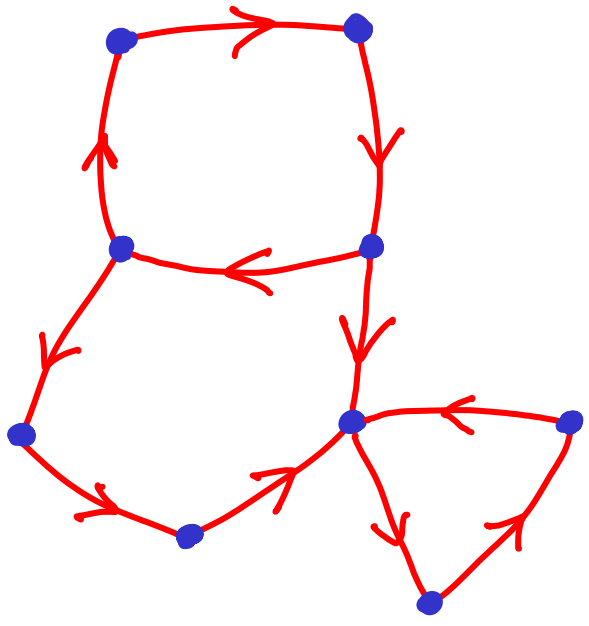
chain reaction:
several vertices
no longer
mutually reachable

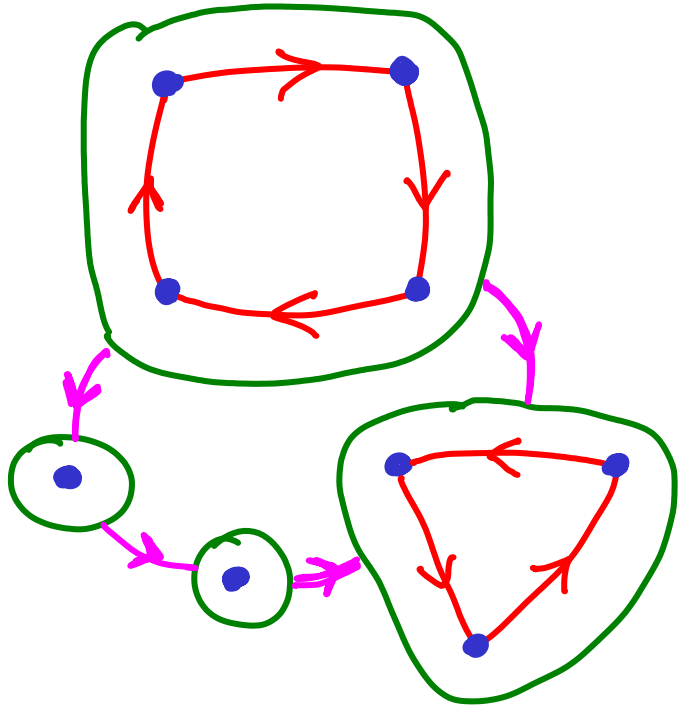


we get groups (components)
each of which is
strongly connected

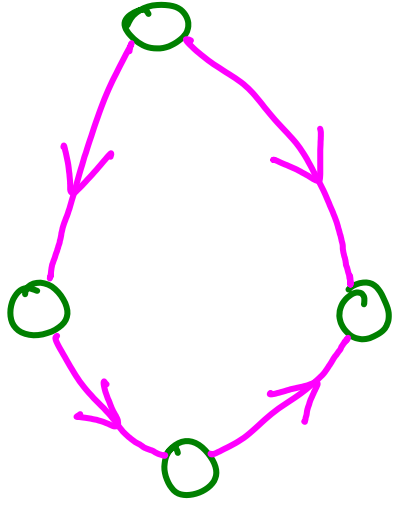






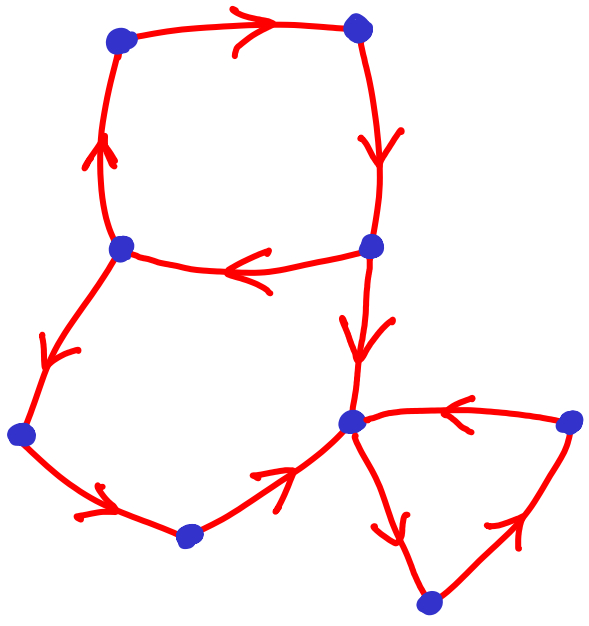


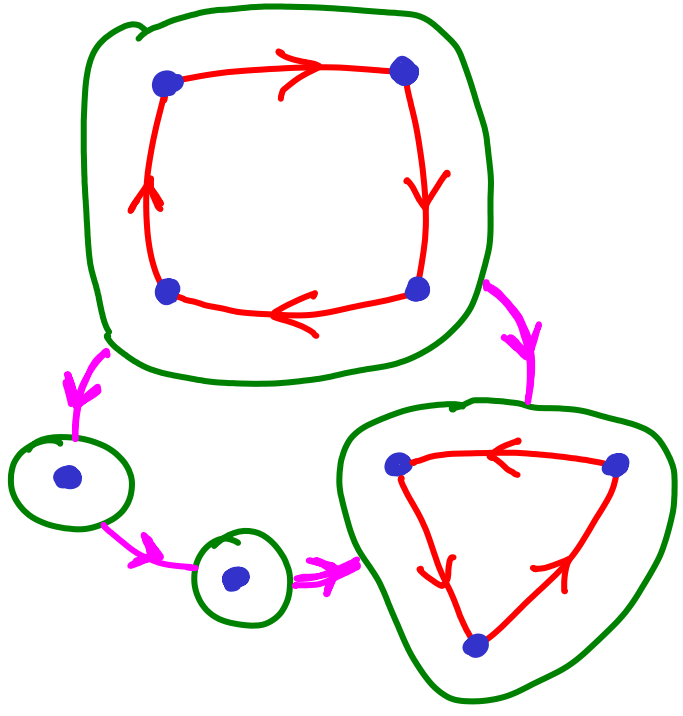
new graph:



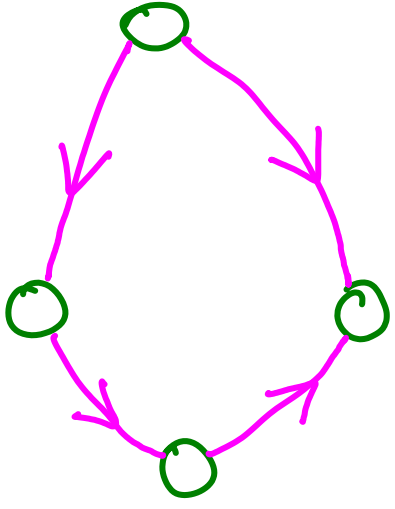
each vertex represents a S.C.C.

What type of graph is this?





new graph:



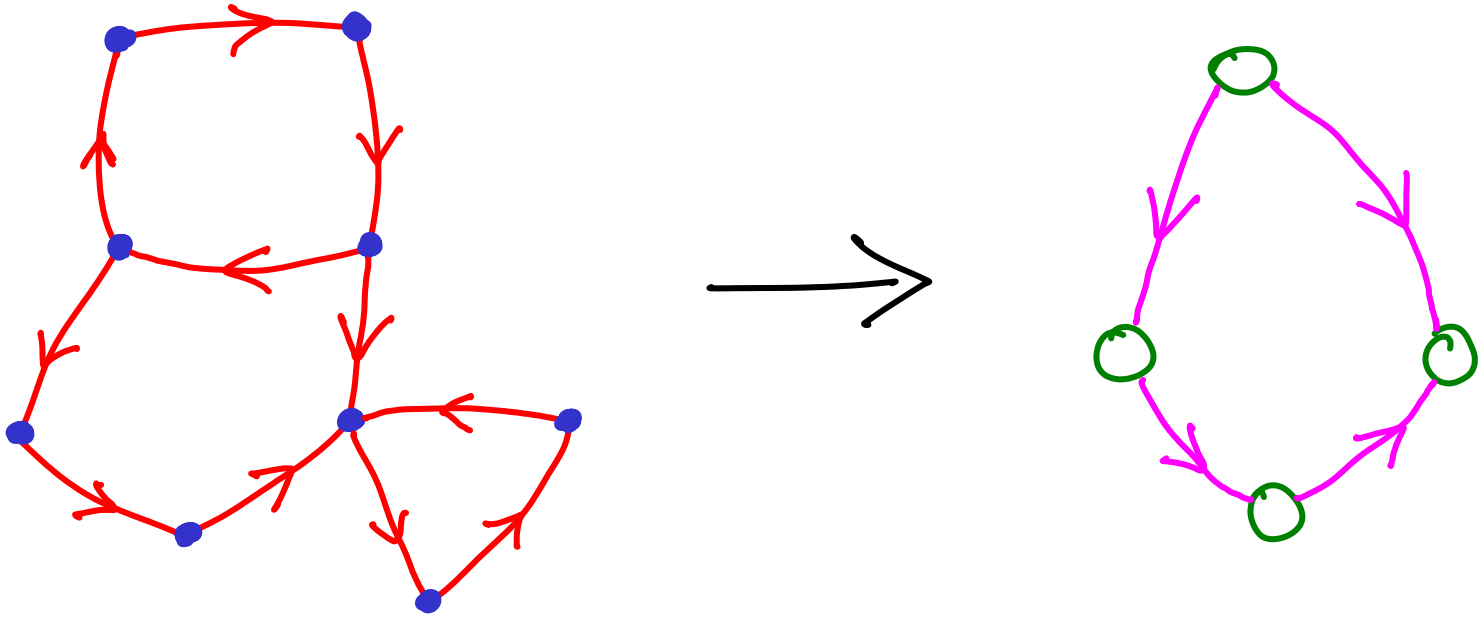
each vertex represents a S.C.C.

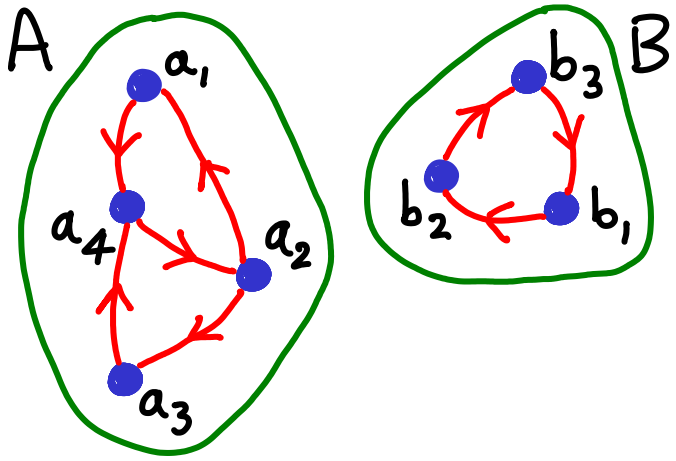
The new graph must be a DAG

↳ (any cycle would merge components)

RECOGNIZING
STRONGLY CONNECTED COMPONENTS

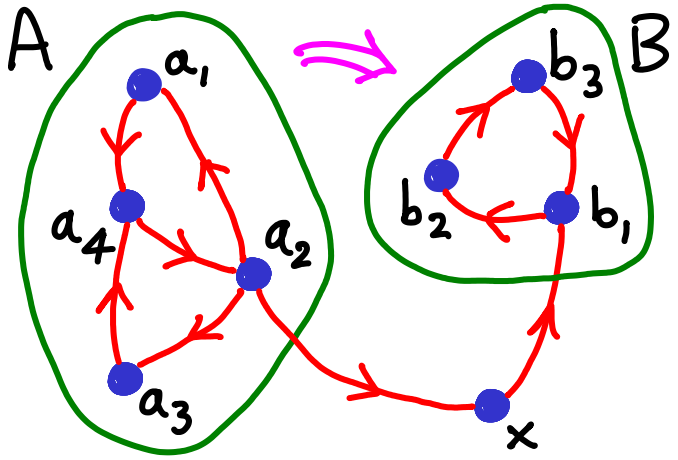
Part 1: useful properties





Consider any two SCC : A & B

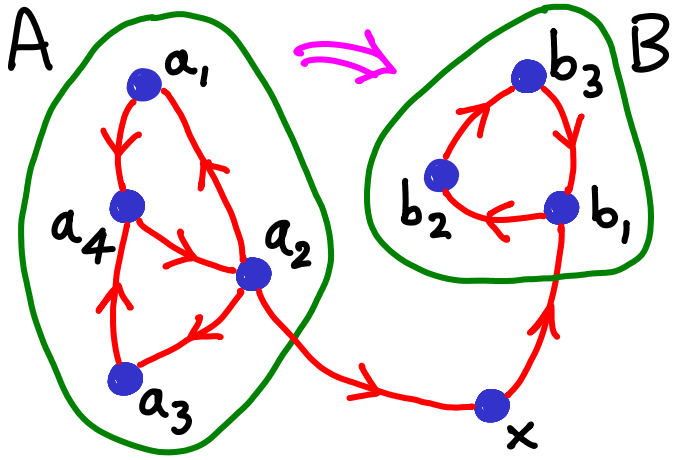
CASE 1: either they're unrelated



Consider any two SCC : A & B

CASE 1 : either they're unrelated

CASE 2 : or precisely one links to the other

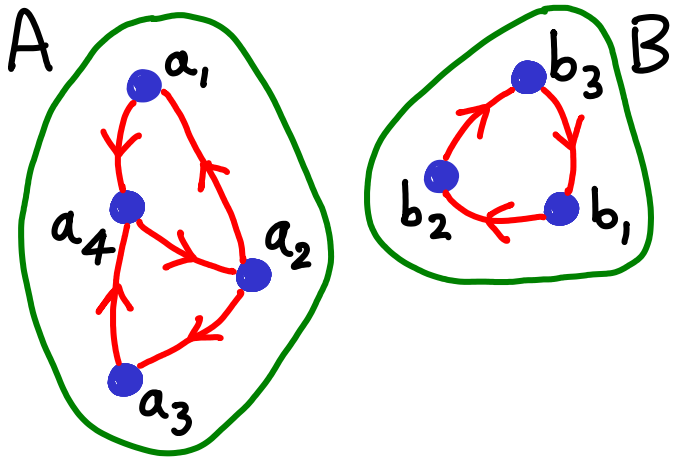


Consider any two SCC : A & B

CASE 1: either they're unrelated

CASE 2: or precisely one links to the other

Which vertices will finish first in a DFS?



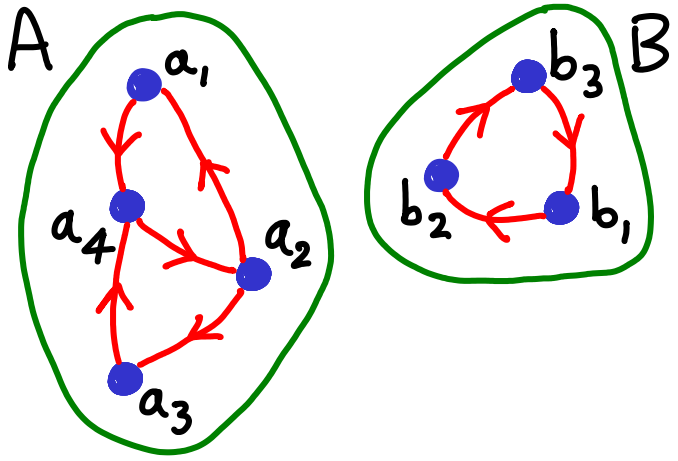
Consider any two SCC : A & B

CASE 1 : either they're unrelated

CASE 2 : or precisely one links to the other

Which vertices will finish first in a DFS?

if CASE 1 ?



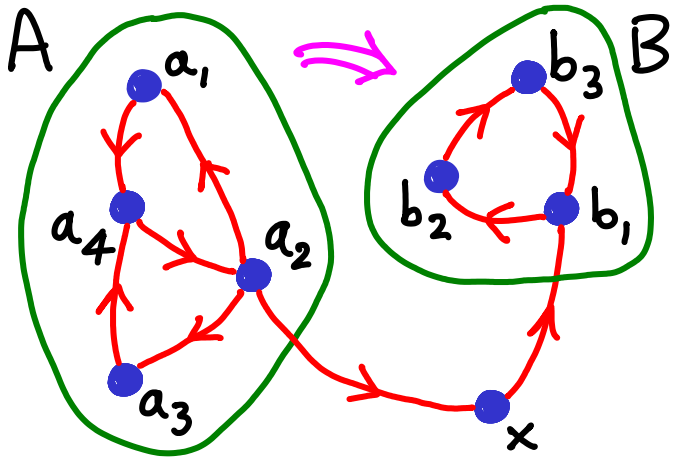
Consider any two SCC : A & B

CASE 1: either they're unrelated

CASE 2: or precisely one links to the other

Which vertices will finish first in a DFS?

if CASE 1, whichever is explored first finishes first.



Consider any two SCC : A & B

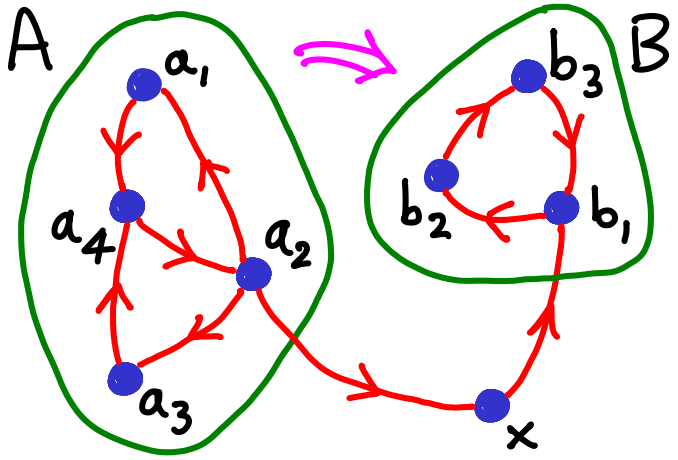
CASE 1: either they're unrelated

CASE 2: or precisely one links to the other

Which vertices will finish first in a DFS?

if CASE 1, whichever is explored first finishes first.

if CASE 2 ?



Consider any two SCC : A & B

CASE 1 : either they're unrelated

CASE 2 : or precisely one links to the other

Which vertices will finish first in a DFS?

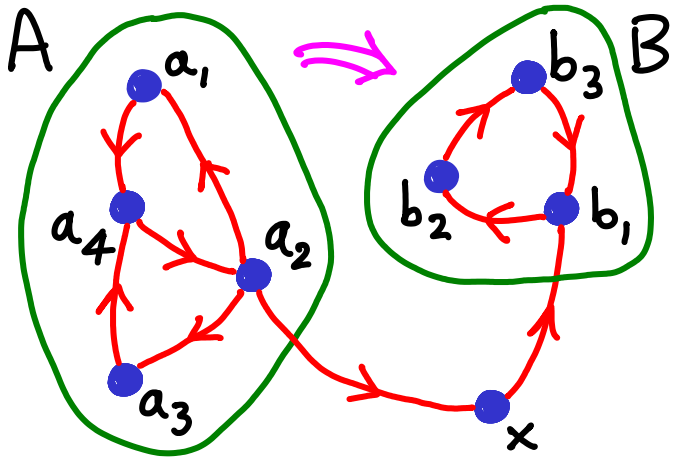
if CASE 1, whichever is explored first finishes first.

---

if CASE 2 :

Either B gets explored first → then?

or A gets explored first



Consider any two SCC : A & B

CASE 1 : either they're unrelated

CASE 2 : or precisely one links to the other

Which vertices will finish first in a DFS?

if CASE 1, whichever is explored first finishes first.

---

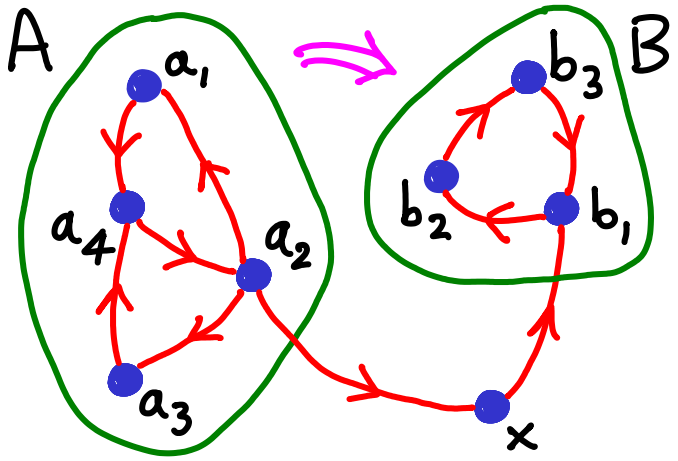
if CASE 2 :

Either B gets explored first (and it can't find A)

or A gets explored first

same as CASE 1

Result : {all A} finishes after {all B}



Consider any two SCC : A & B

CASE 1 : either they're unrelated

CASE 2 : or precisely one links to the other

Which vertices will finish first in a DFS?

if CASE 1, whichever is explored first finishes first.

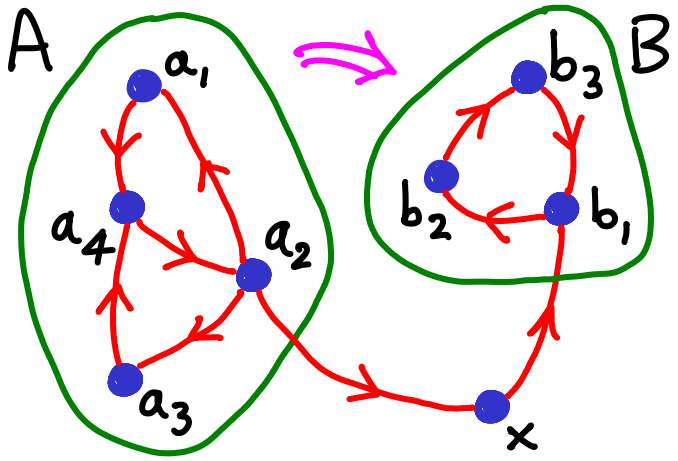
---

if CASE 2 :

Either B gets explored first (and it can't find A)

or A gets explored first, so ?

Result : {all A} finishes after {all B}



Consider any two SCC : A & B

CASE 1 : either they're unrelated

CASE 2 : or precisely one links to the other

Which vertices will finish first in a DFS?

if CASE 1, whichever is explored first finishes first.

---

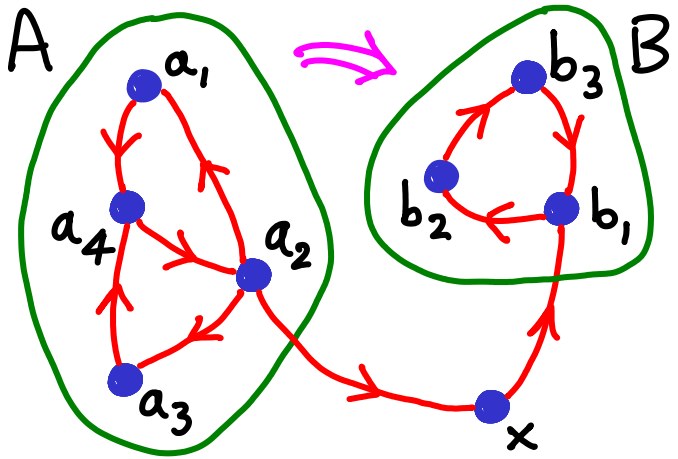
if CASE 2 :

Either B gets explored first (and it can't find A)

or A gets explored first, so some a_i will find all B before finishing

Result : {all A} finishes after {all B}

or {some of A} finishes after {all B} after {some of A}



Consider any two SCC : A & B

CASE 1 : either they're unrelated

CASE 2 : or precisely one links to the other

Which vertices will finish first in a DFS?

if CASE 1, whichever is explored first finishes first.

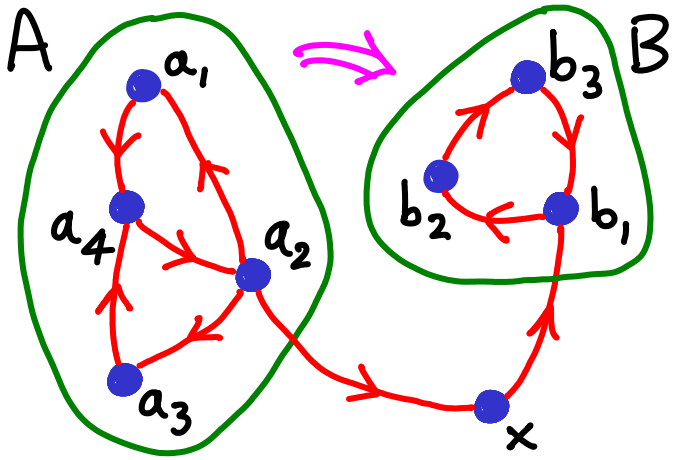
if CASE 2 :

Either B gets explored first (and it can't find A)

or A gets explored first, so some a_i will find all B before finishing

Result : {all A} finishes after {all B}

or {some of A} finishes after {all B} after {some of A}



Consider any two SCC : A & B

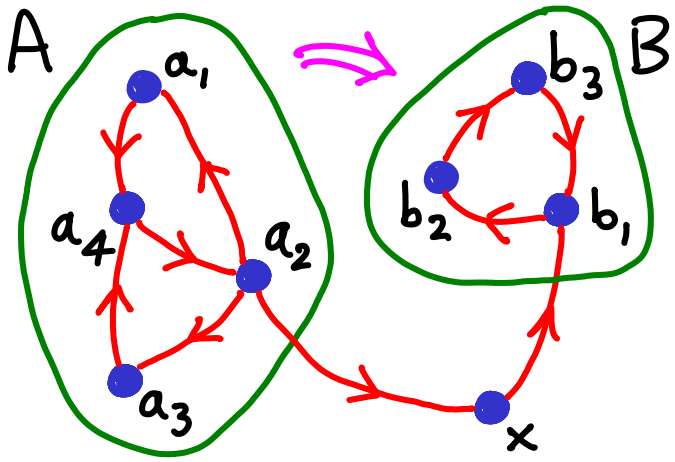
CASE 1: either they're unrelated

CASE 2: or precisely one links to the other

Which vertices will finish first in a DFS?

Example for CASE 2

a_4 a_2 a_3 a_1 x b_1 b_2 b_3 → DFS started at x or b_1 or $a_4 \rightarrow a_2 \rightarrow x$...



Consider any two SCC : A & B

CASE 1 : either they're unrelated

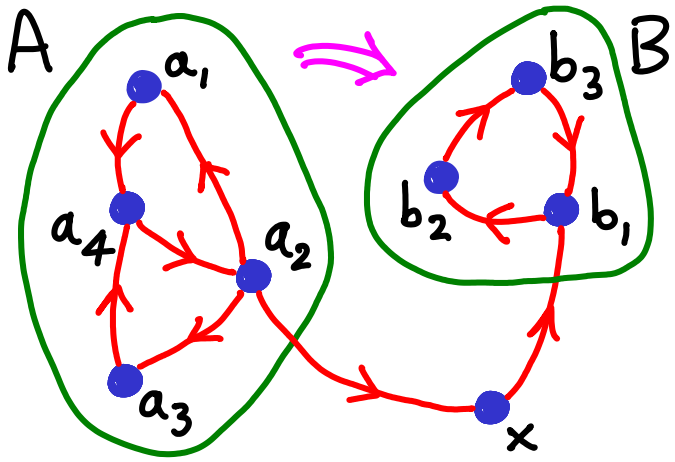
CASE 2 : or precisely one links to the other

Which vertices will finish first in a DFS?

Examples for CASE 2

$a_4\ a_2\ a_3\ a_1\ x\ b_1\ b_2\ b_3$ → DFS started at x or b_1 or $a_4 \to a_2 \to x \ldots$

$a_2\ a_3\ a_1\ a_4\ x\ b_1\ b_2\ b_3$ → DFS started at ?



Consider any two SCC : A & B

CASE 1 : either they're unrelated

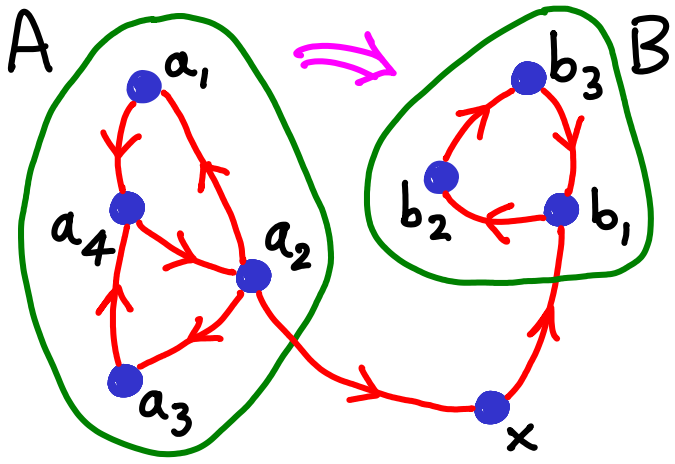
CASE 2 : or precisely one links to the other

Which vertices will finish first in a DFS?

Examples for CASE 2

$a_4\ a_2\ a_3\ a_1\ x\ b_1\ b_2\ b_3$ → DFS started at x or b_1 or $a_4 \to a_2 \to x \ldots$

$a_2\ a_3\ a_1\ a_4\ x\ b_1\ b_2\ b_3$ → DFS started at x or b_1 or $a_2 \to x \ldots$



Consider any two SCC : A & B

CASE 1 : either they're unrelated

CASE 2 : or precisely one links to the other

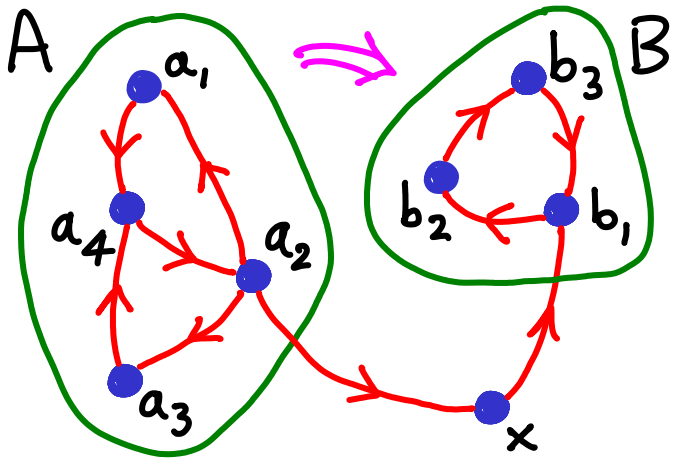
Which vertices will finish first in a DFS?

Examples for CASE 2

$a_4\ a_2\ a_3\ a_1\ x\ b_1\ b_2\ b_3$ → DFS started at x or b_1 or $a_4 \to a_2 \to x \ldots$

$a_2\ a_3\ a_1\ a_4\ x\ b_1\ b_2\ b_3$ → DFS started at x or b_1 or $a_2 \to x \ldots$

$a_3\ a_4\ a_2\ a_1\ x\ b_2\ b_3\ b_1$ → DFS started at ?



Consider any two SCC : A & B

CASE 1 : either they're unrelated

CASE 2 : or precisely one links to the other

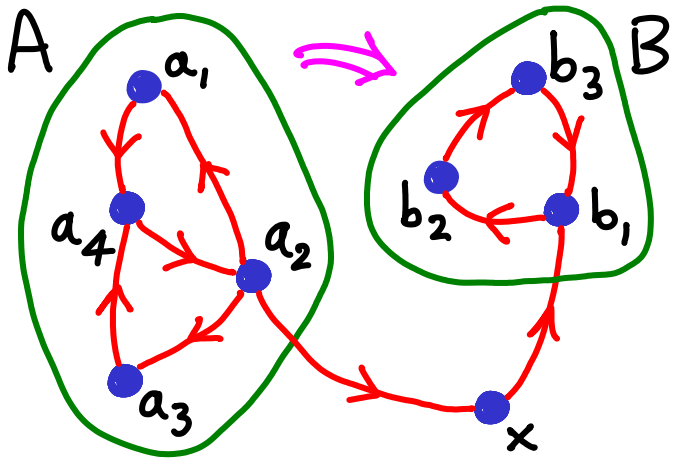
Which vertices will finish first in a DFS?

Examples for CASE 2

$a_4\ a_2\ a_3\ a_1\ x\ b_1\ b_2\ b_3$ → DFS started at x or b_1 or $a_4 \to a_2 \to x \ldots$

$a_2\ a_3\ a_1\ a_4\ x\ b_1\ b_2\ b_3$ → DFS started at x or b_1 or $a_2 \to x \ldots$

$a_3\ a_4\ a_2\ a_1\ x\ b_2\ b_3\ b_1$ → DFS started at b_2, then $a_3 \to a_4 \to a_2 \ldots$



Consider any two SCC : A & B

CASE 1 : either they're unrelated

CASE 2 : or precisely one links to the other

Which vertices will finish first in a DFS?

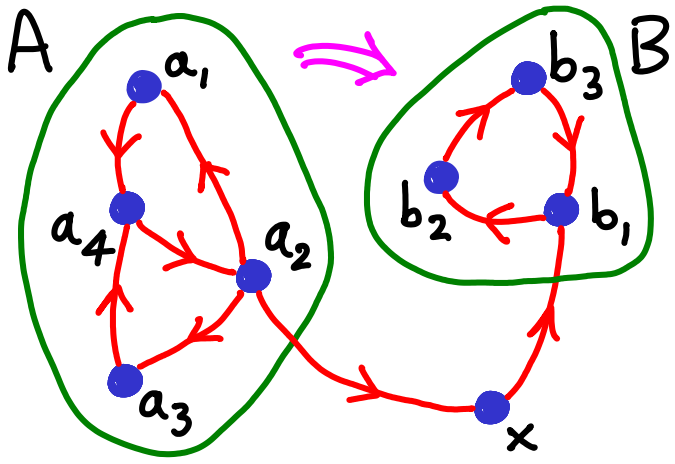
Examples for CASE 2

$a_4\ a_2\ a_3\ a_1\ x\ b_1\ b_2\ b_3$ → DFS started at x or b_1 or $a_4 \to a_2 \to x \ldots$

$a_2\ a_3\ a_1\ a_4\ x\ b_1\ b_2\ b_3$ → DFS started at x or b_1 or $a_2 \to x \ldots$

$a_3\ a_4\ a_2\ a_1\ x\ b_2\ b_3\ b_1$ → DFS started at b_2, then $a_3 \to a_4 \to a_2 \ldots$

$a_2\ a_1\ x\ b_1\ b_2\ b_3\ a_3\ a_4$ → DFS started at ?



Consider any two SCC : A & B

CASE 1 : either they're unrelated

CASE 2 : or precisely one links to the other

Which vertices will finish first in a DFS?

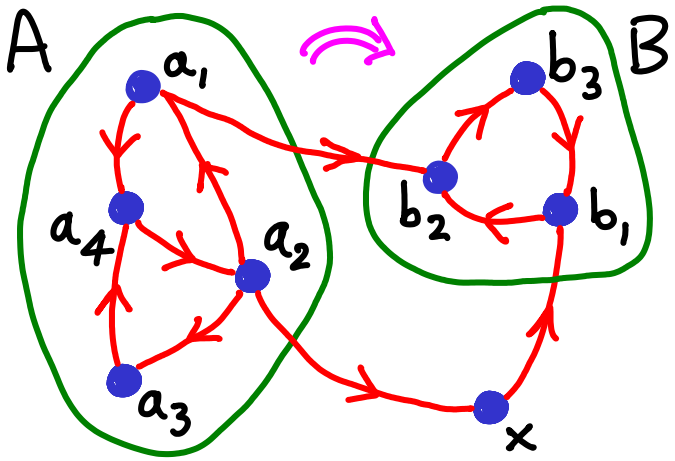
Examples for CASE 2

$a_4\ a_2\ a_3\ a_1\ x\ b_1\ b_2\ b_3$ → DFS started at x or b_1 or $a_4 \to a_2 \to x \ldots$

$a_2\ a_3\ a_1\ a_4\ x\ b_1\ b_2\ b_3$ → DFS started at x or b_1 or $a_2 \to x \ldots$

$a_3\ a_4\ a_2\ a_1\ x\ b_2\ b_3\ b_1$ → DFS started at b_2, then $a_3 \to a_4 \to a_2 \ldots$

$a_2\ a_1\ x\ b_1\ b_2\ b_3\ a_3\ a_4$ → DFS started at $a_2 \to a_3 \ldots$

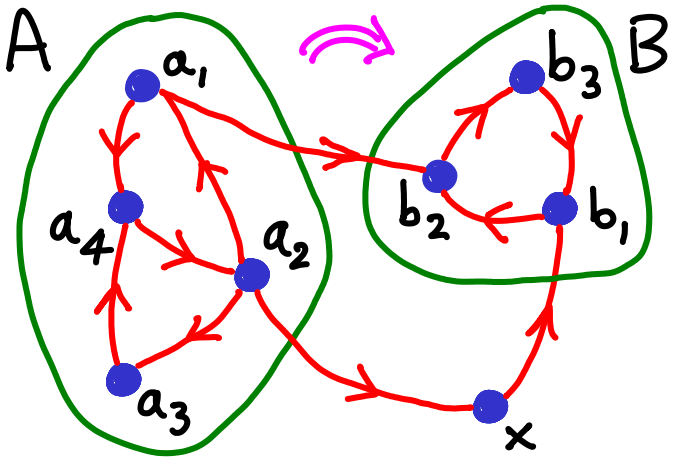


DFS finishing times (focusing on A ⇒ B)

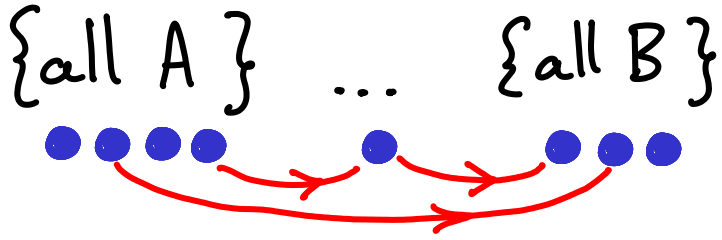
{all A} ... {all B}

(same if A, B unrelated) w.l.o.g.

or {some of A} ... {all B} ... {some of A}

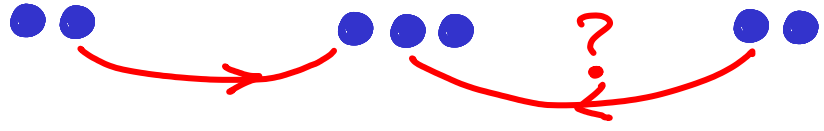


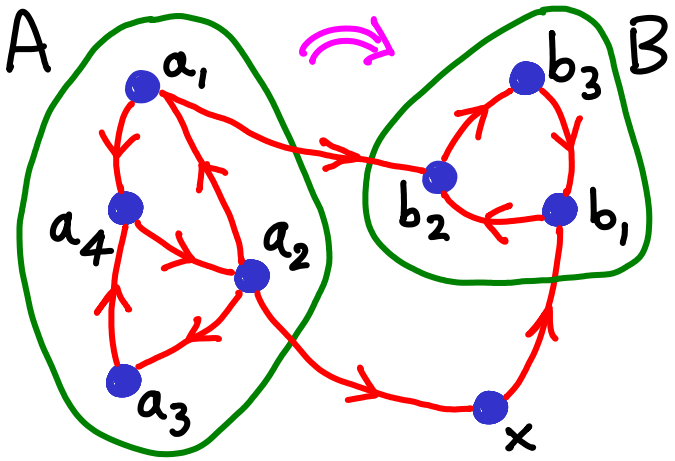
DFS finishing times (focusing on A ⇒ B)



(same if A, B unrelated) w.l.o.g.

or {some of A} ... {all B} ... {some of A}





DFS finishing times (focusing on $A \Rightarrow B$)

{all A} ... {all B}

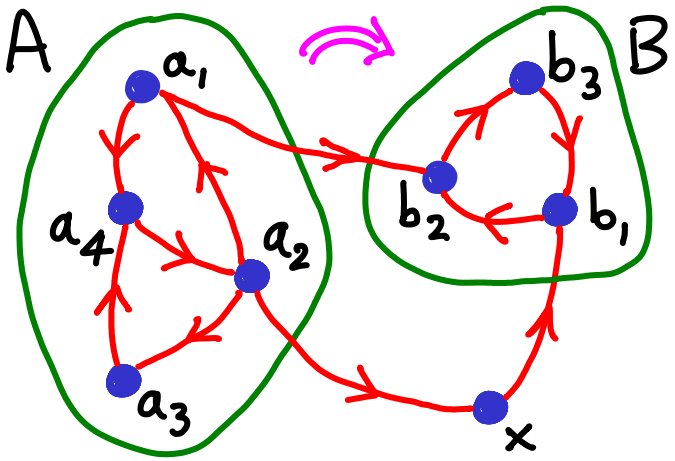
(same if A, B unrelated) w.l.o.g.

or {some of A} ... {all B} ... {some of A}

a_1, a_2

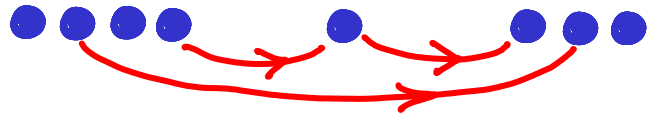
NOT a_1, a_2

not without going via other A that finished after B



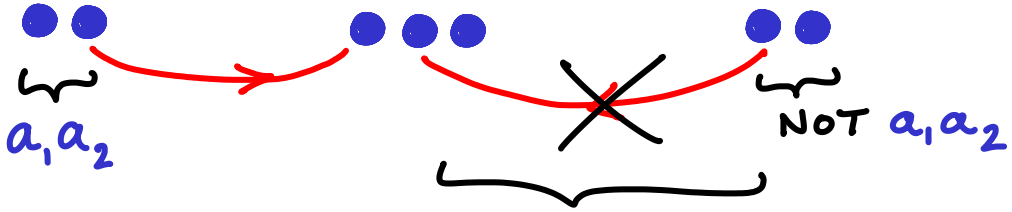
DFS finishing times (focusing on A ⇒ B)

{all A} ... {all B}

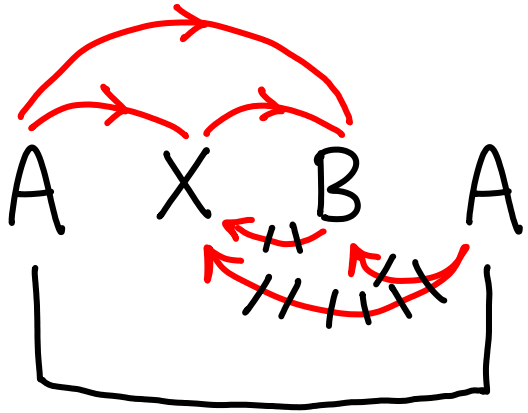


(same if A, B unrelated) w.l.o.g.

or {some of A} ... {all B} ... {some of A}



not without going via other A that finished after B



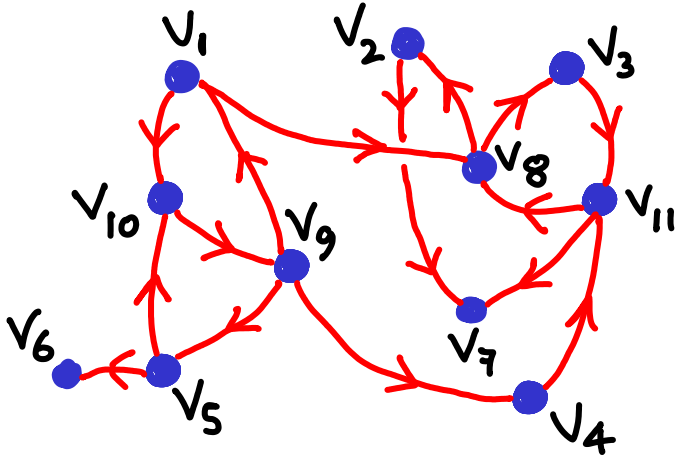
: nested groups

quasi-topologically sorted

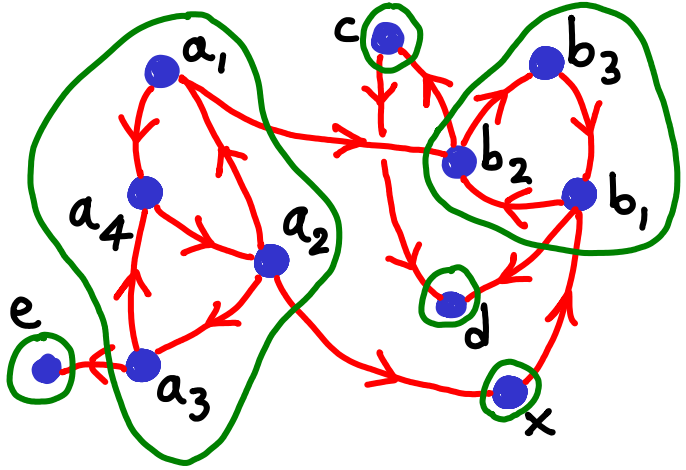
(back ← arrows only within SCC)

FINDING ALL STRONGLY CONNECTED COMPONENTS

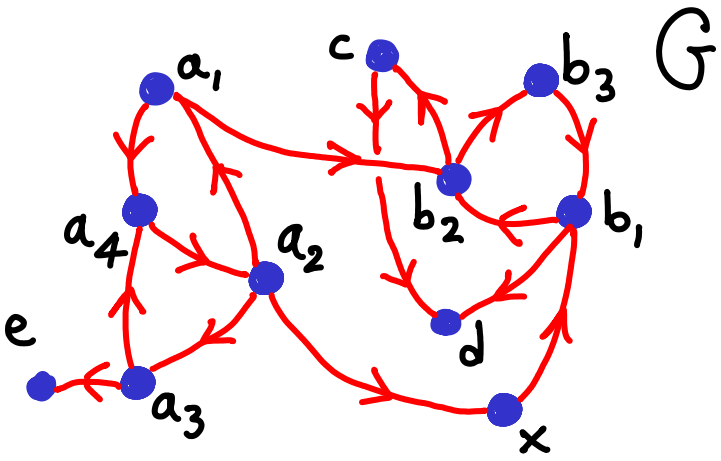
Part 2: algorithm



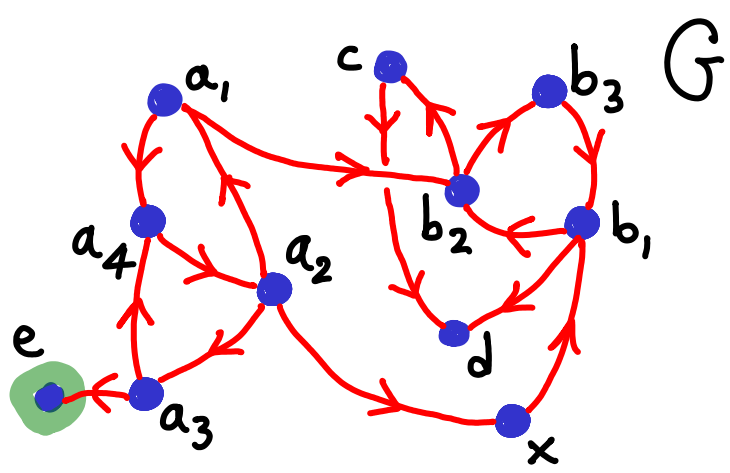
what we see



what we want

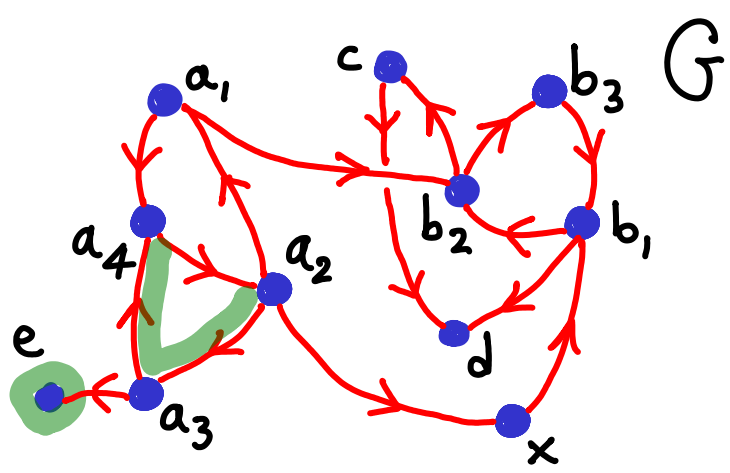


DFS(G)



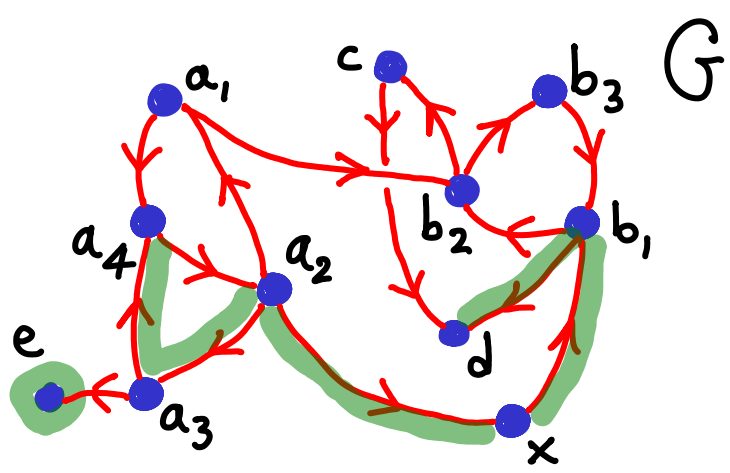
DFS(G)
finishing times

e



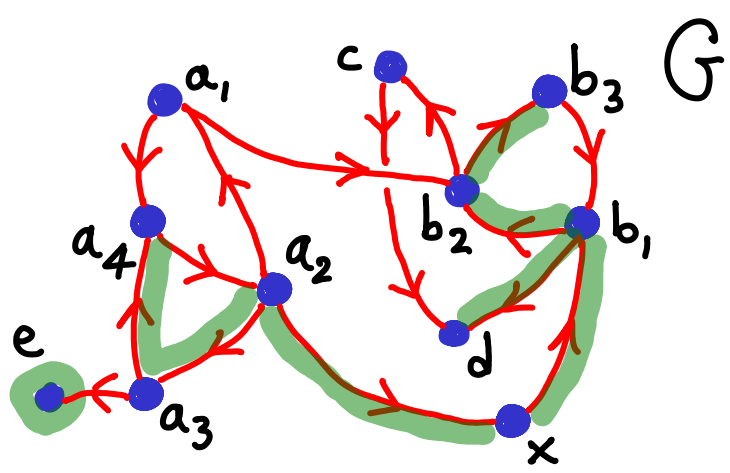
DFS(G)
finishing times

$a_3\ a_4\ e$



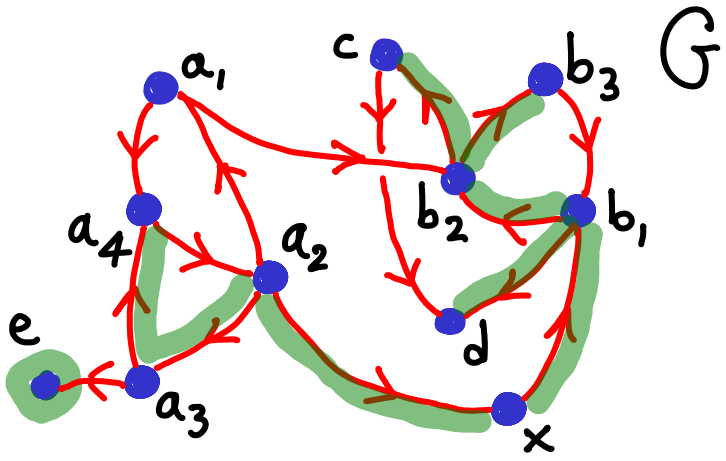
DFS(G)
finishing times

d a_3 a_4 e



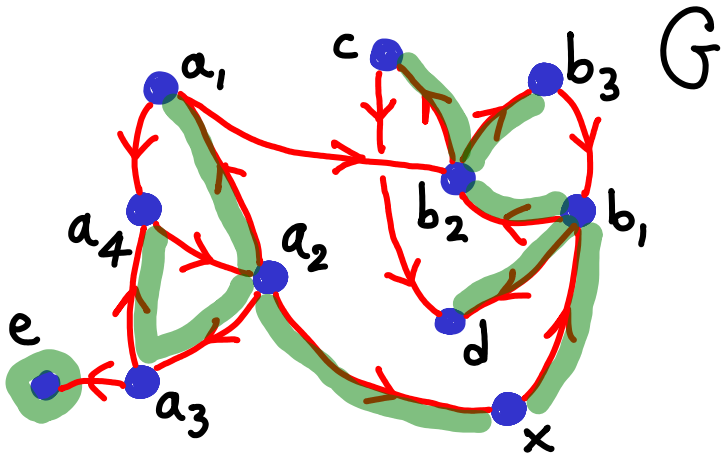
DFS(G)
finishing times

b_3 d a_3 a_4 e



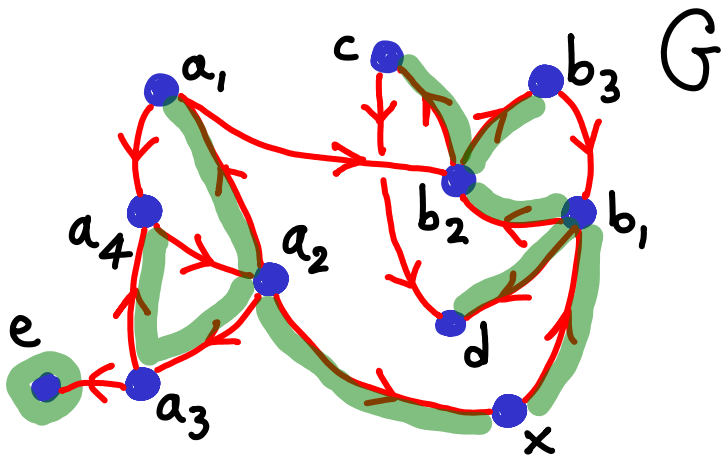
DFS(G)
finishing times

c b_3 d a_3 a_4 e

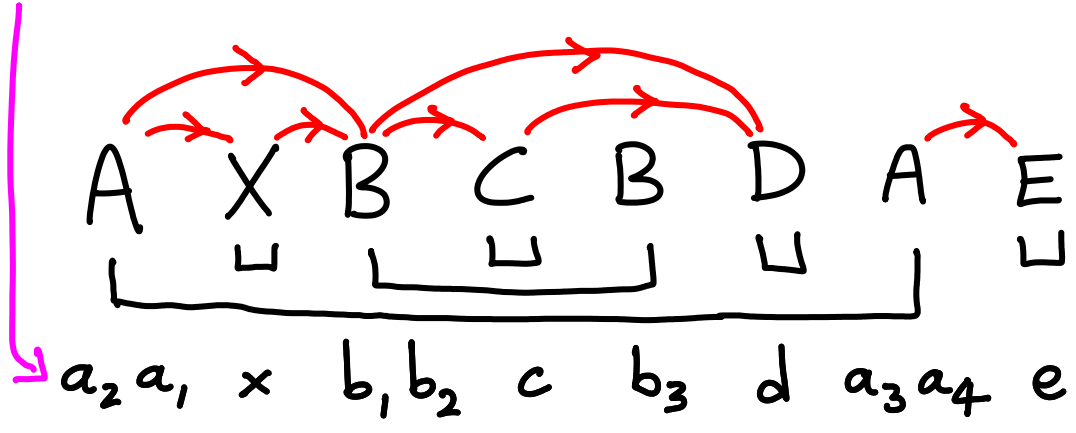


DFS(G)
finishing times

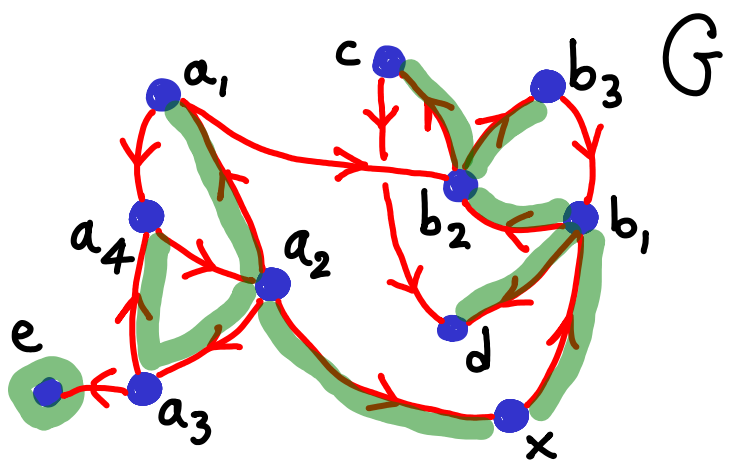
$a_2\ a_1\ x\ b_1\ b_2\ c\ b_3\ d\ a_3\ a_4\ e$



DFS(G)
finishing times



: nested groups
quasi-topologically sorted
(back ← arrows only within SCC)

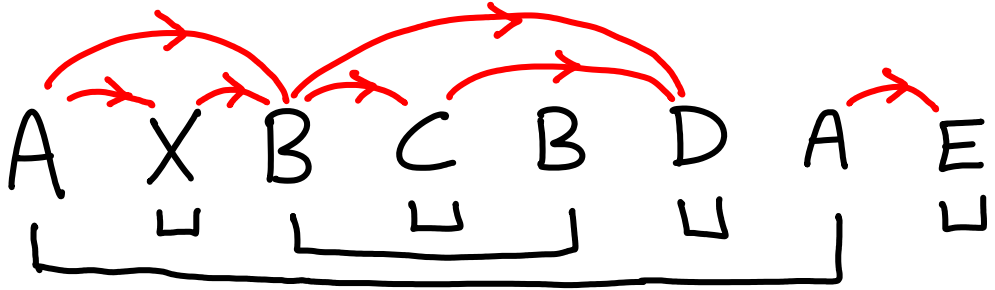


We don't know any of these groups yet

To us, looks like

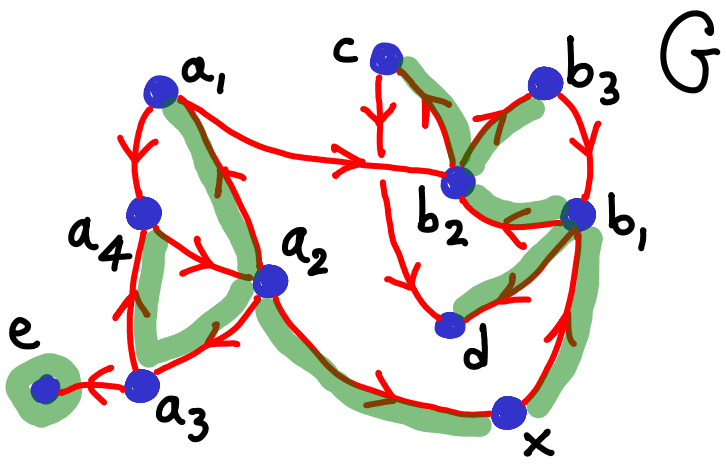
a_2	a_1	x	b_1	b_2	c	b_3	d	a_3	a_4	e
v_9	v_1	v_4	v_{11}	v_8	v_2	v_3	v_7	v_5	v_{10}	v_6

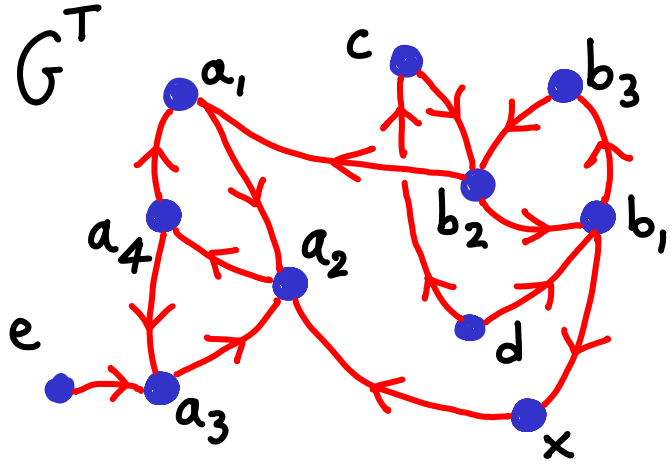
DFS(G) finishing times



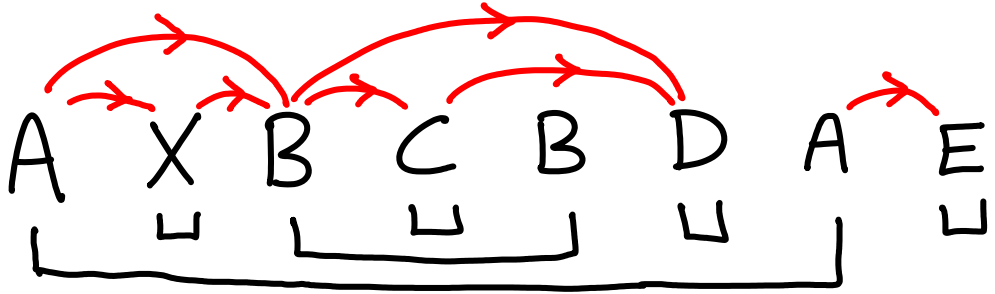
a_2 a_1 x b_1 b_2 c b_3 d a_3 a_4 e

: nested groups
quasi-topologically sorted
(back ← arrows only within SCC)



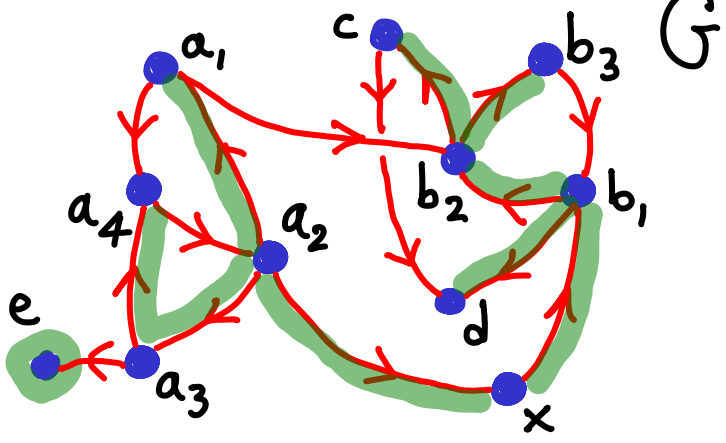


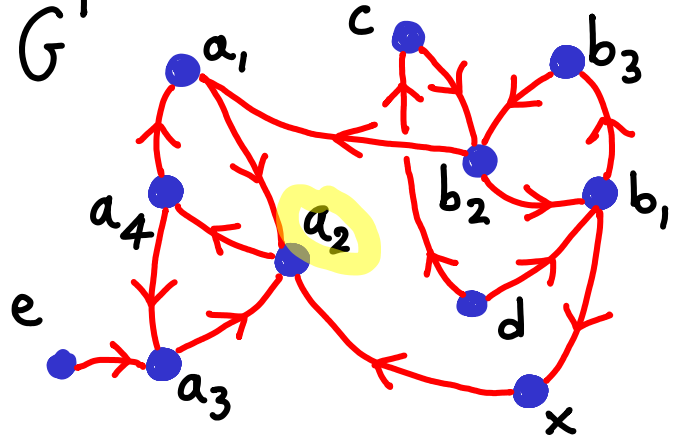
DFS(G)
finishing times



$a_2\ a_1\ x\ b_1\ b_2\ c\ b_3\ d\ a_3\ a_4\ e$

: nested groups
quasi-topologically sorted
(back ← arrows only within SCC)

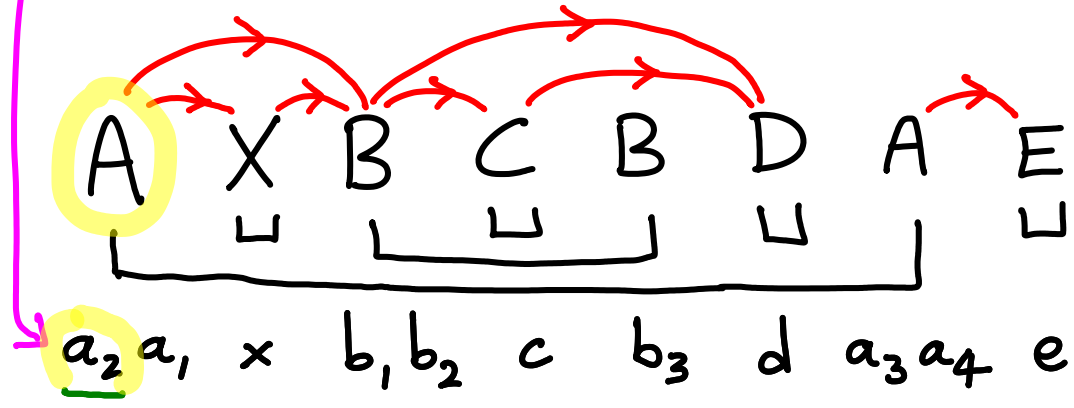




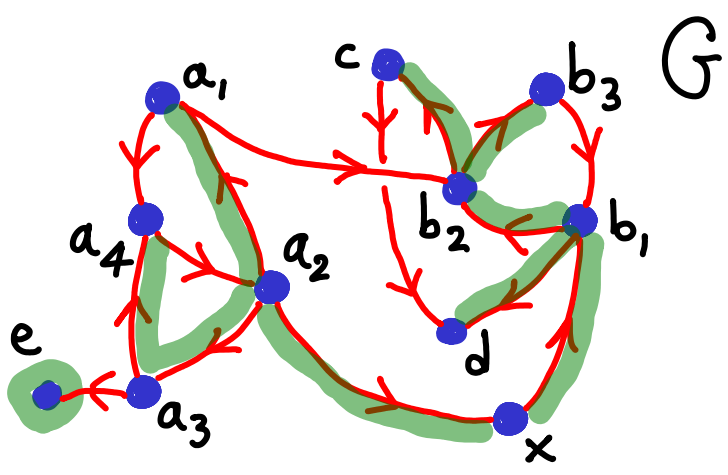
Run DFS on G^T, processing vertices in order of finishing times obtained in DFS(G)

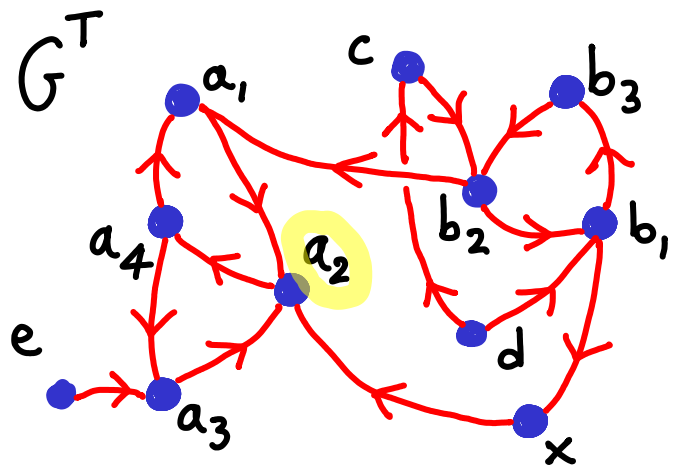
- Start DFS(G^T) on some a_i (whatever finished last)

DFS(G) finishing times



: nested groups
quasi-topologically sorted
(back ← arrows only within SCC)

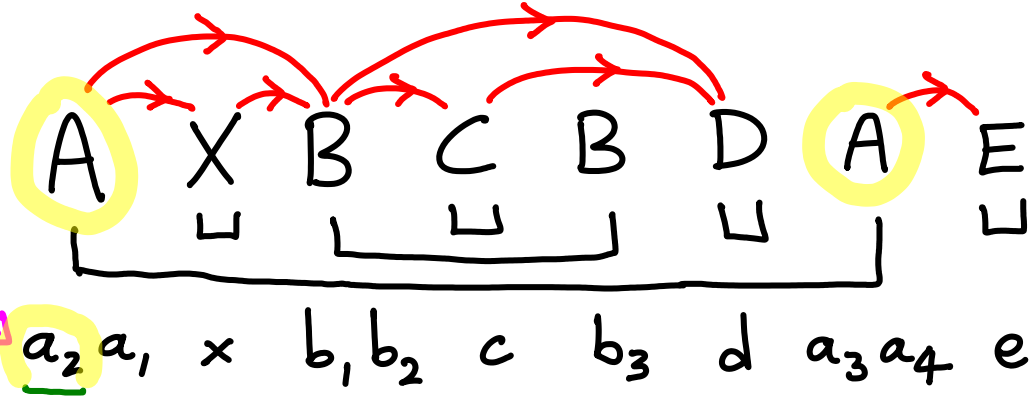




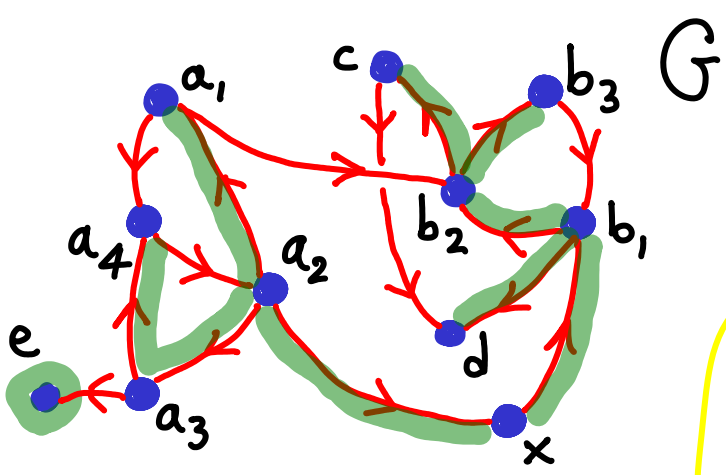
Run DFS on G^T, processing vertices in order of finishing times obtained in DFS(G)

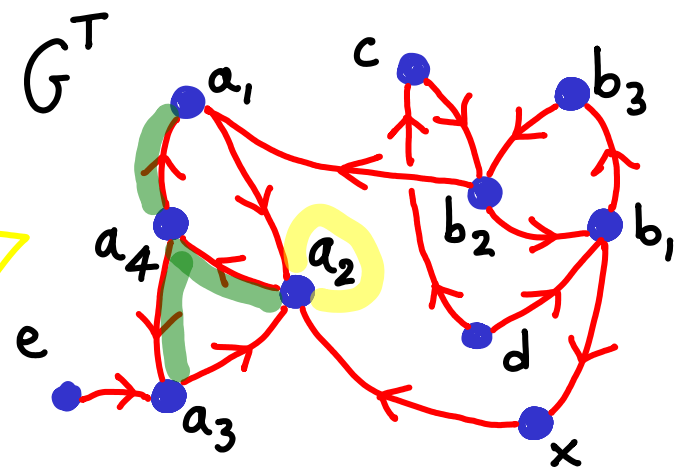
- Start DFS(G^T) on some a_i (whatever finished last)
- Only A can link to a_i in G

DFS(G) finishing times



: nested groups quasi-topologically sorted (back ← arrows only within SCC)

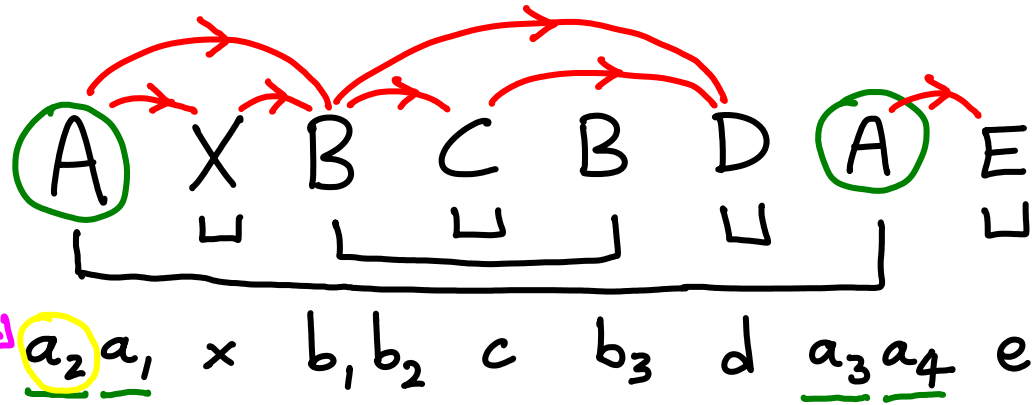




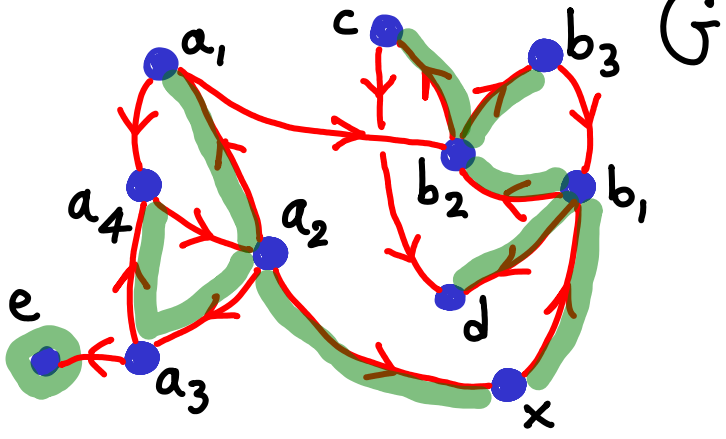
Run DFS on G^T, processing vertices in order of finishing times obtained in DFS(G)

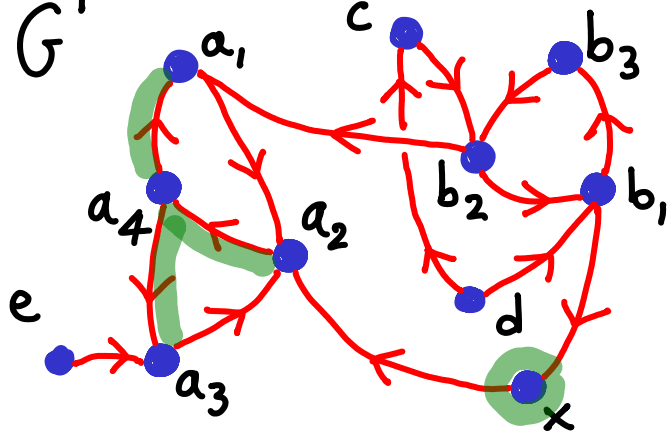
- Start DFS(G^T) on some a_i (whatever finished last)
- Only A can link to a_i in G
  - ↳ a_i will find A and nothing else in G^T

DFS(G) finishing times



: nested groups quasi-topologically sorted (back←arrows only within SCC)

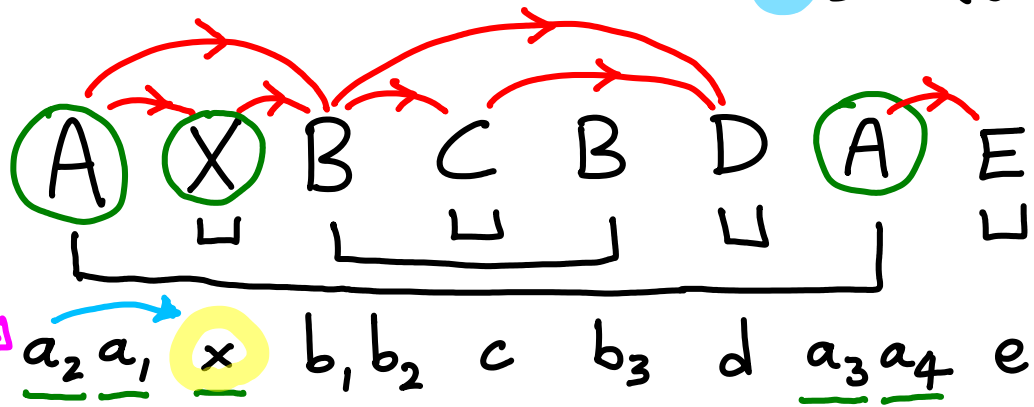




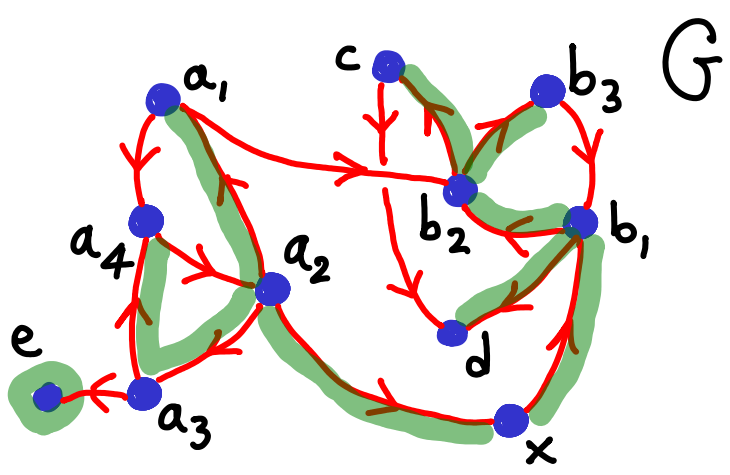
Run DFS on G^T, processing vertices in order of finishing times obtained in DFS(G)

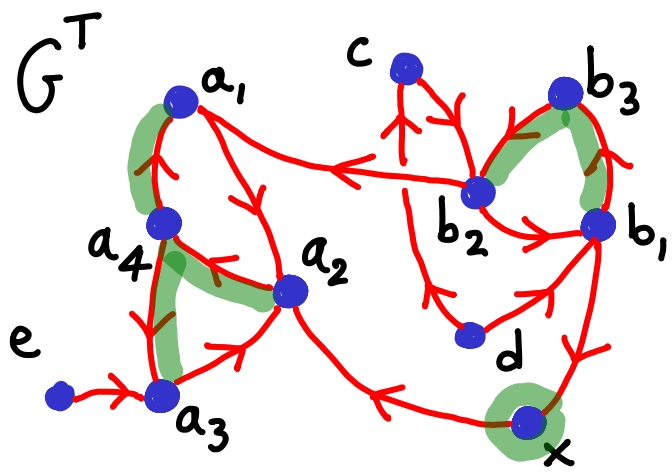
- Start DFS(G^T) on some a_i (whatever finished last)
- Only A can link to a_i in G
  - ↳ a_i will find A and nothing else in G^T
- DFS(G^T) will continue on next unmarked vertex
  - ↳ some x_i : finds only X ... etc

DFS(G) finishing times



: nested groups quasi-topologically sorted (back←arrows only within SCC)

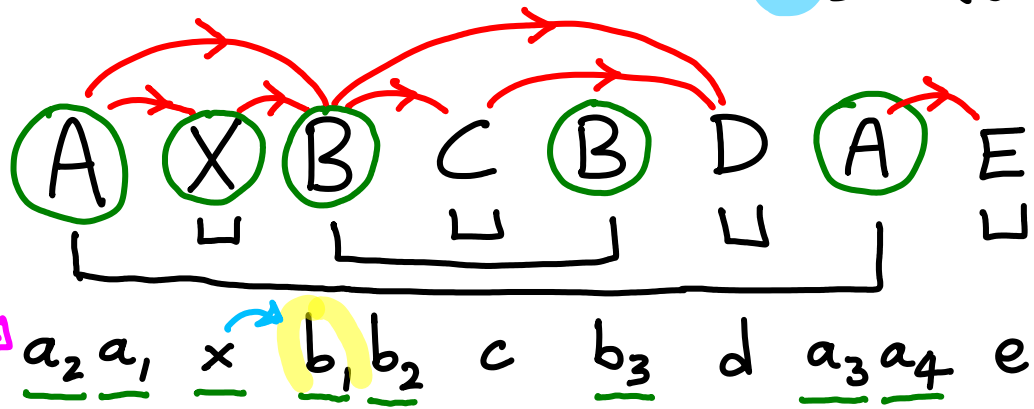




Run DFS on G^T, processing vertices in order of finishing times obtained in DFS(G)

- Start DFS(G^T) on some a_i (whatever finished last)
- Only A can link to a_i in G
  - ↳ a_i will find A and nothing else in G^T
- DFS(G^T) will continue on next unmarked vertex
  - ↳ some x_i : finds only X
  - ... etc

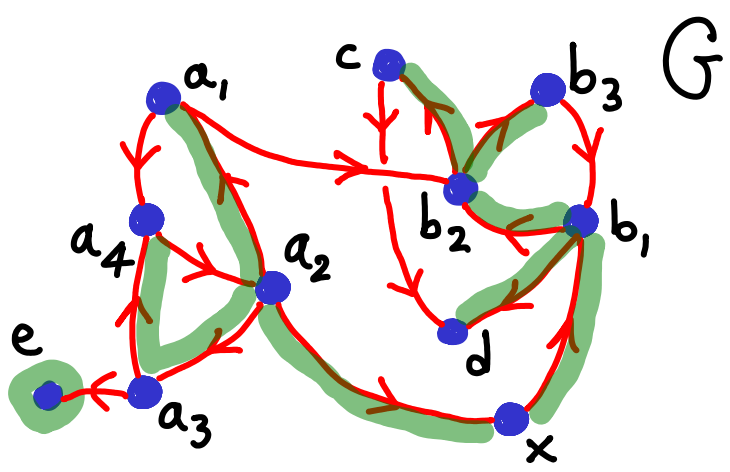
DFS(G) finishing times

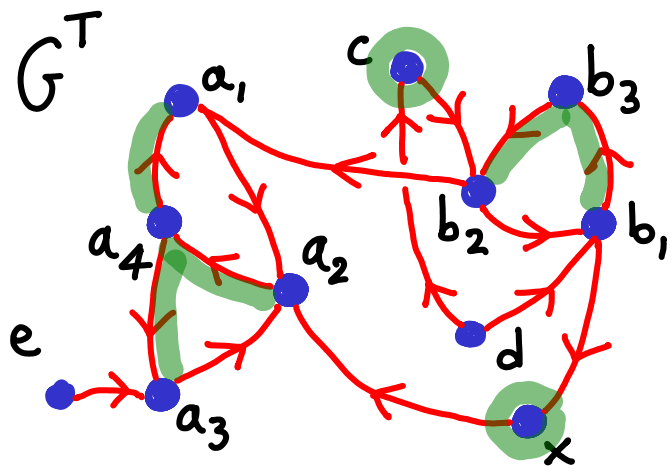


: nested groups

quasi-topologically sorted

(back←arrows only within SCC)

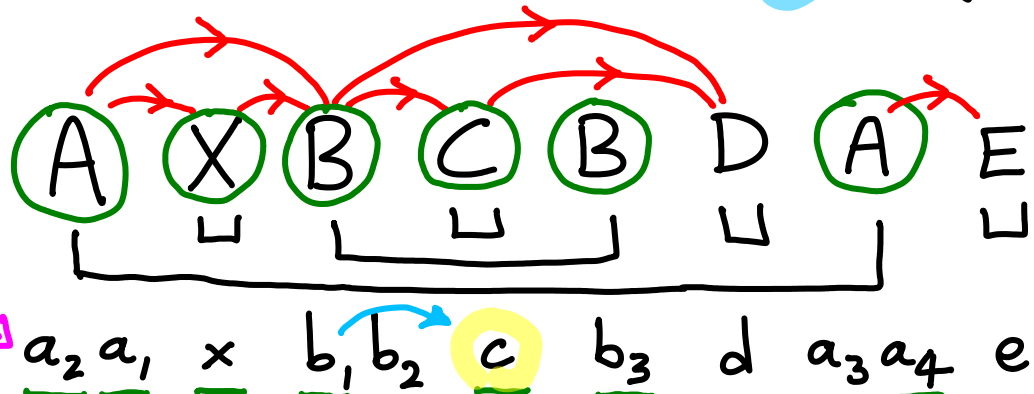




Run DFS on G^T, processing vertices in order of finishing times obtained in DFS(G)

- Start DFS(G^T) on some a_i (whatever finished last)
- Only A can link to a_i in G
  - ↳ a_i will find A and nothing else in G^T
- DFS(G^T) will continue on next unmarked vertex
  - ↳ some X_i : finds only X
  - ... etc

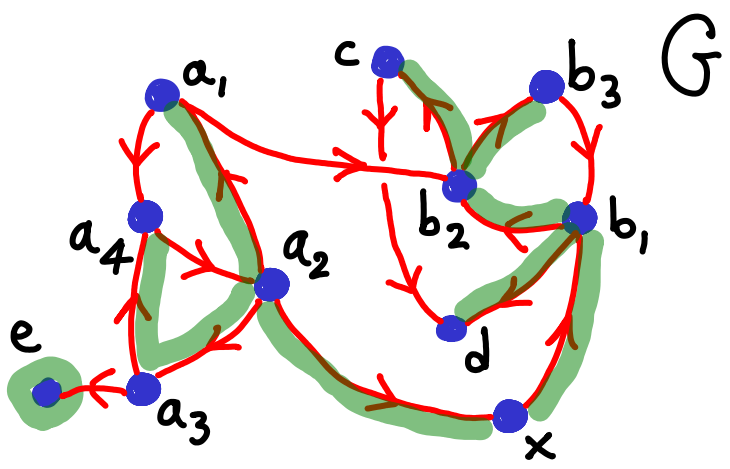
DFS(G) finishing times

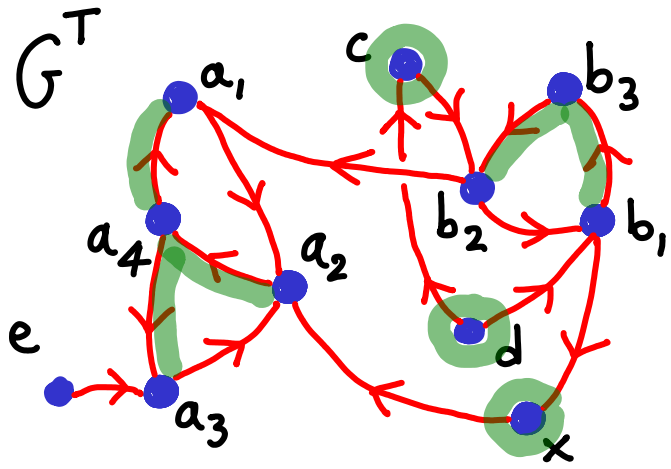


: nested groups

quasi-topologically sorted

(back←arrows only within SCC)

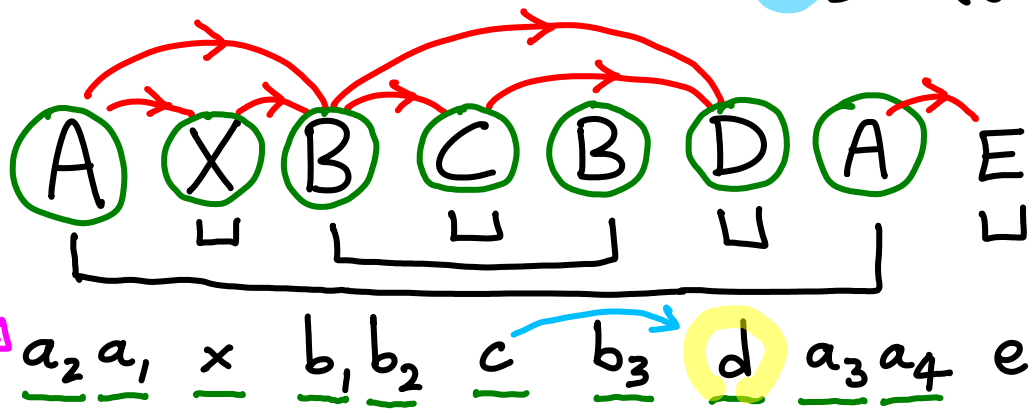




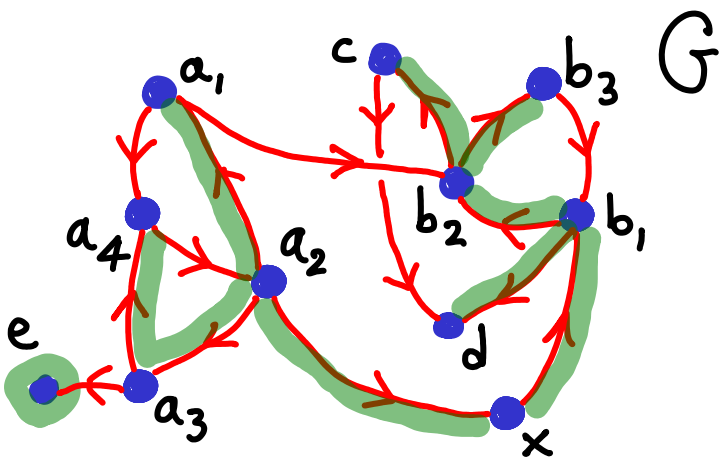
Run DFS on G^T, processing vertices in order of finishing times obtained in DFS(G)

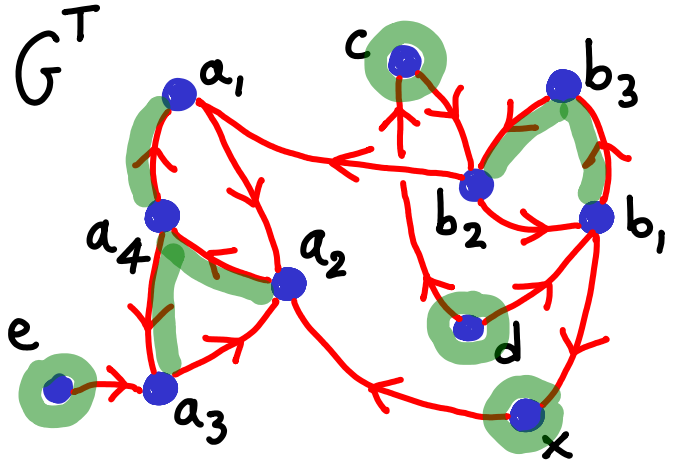
- Start DFS(G^T) on some a_i (whatever finished last)
- Only A can link to a_i in G
  - ↳ a_i will find A and nothing else in G^T
- DFS(G^T) will continue on next unmarked vertex
  - ↳ some X_i : finds only X
  - ... etc

DFS(G) finishing times



: nested groups quasi-topologically sorted (back ← arrows only within SCC)

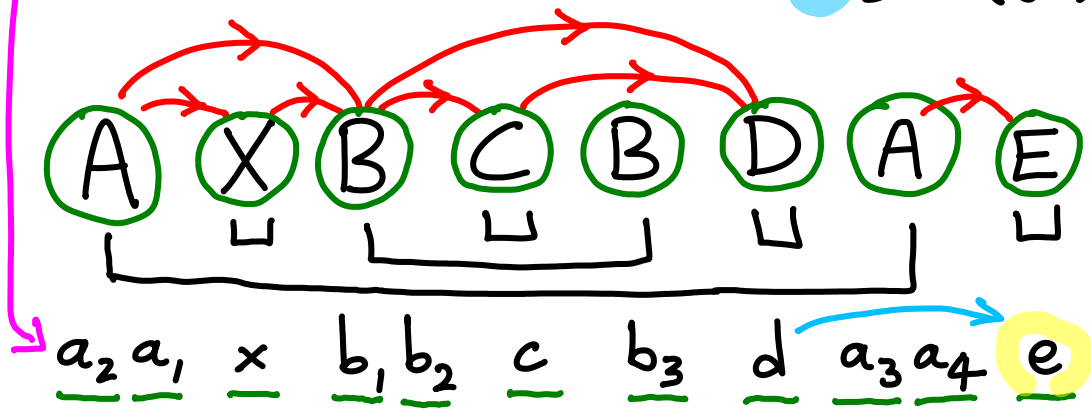




Run DFS on G^T, processing vertices in order of finishing times obtained in DFS(G)

- Start DFS(G^T) on some a_i (whatever finished last)
- Only A can link to a_i in G
  - ↳ a_i will find A and nothing else in G^T
- DFS(G^T) will continue on next unmarked vertex
  - ↳ some x_i : finds only X
  - ... etc

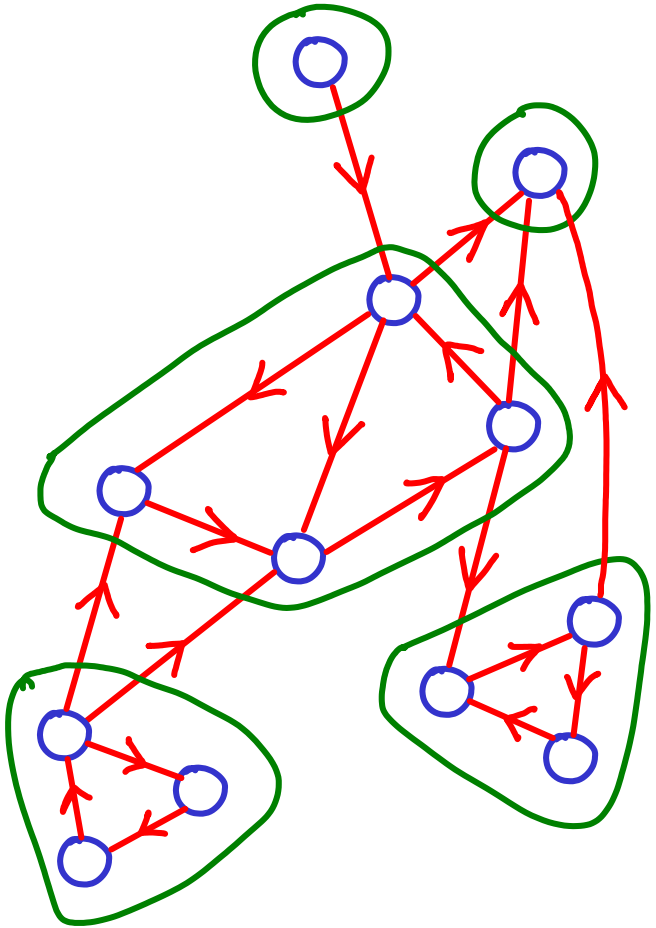
DFS(G) finishing times

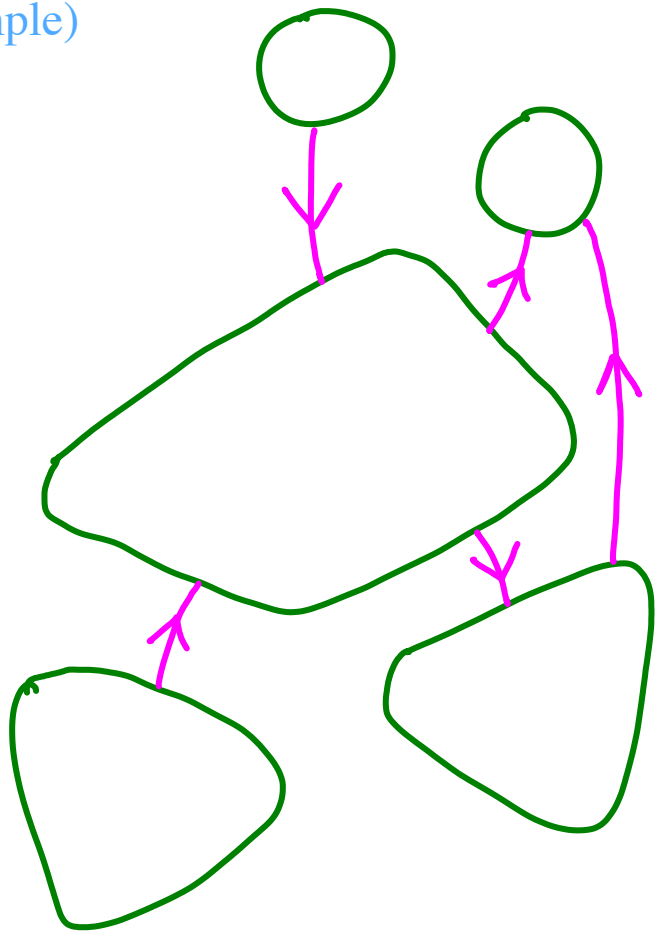


: nested groups quasi-topologically sorted (back←arrows only within SCC)

FINDING STRONGLY CONNECTED COMPONENTS

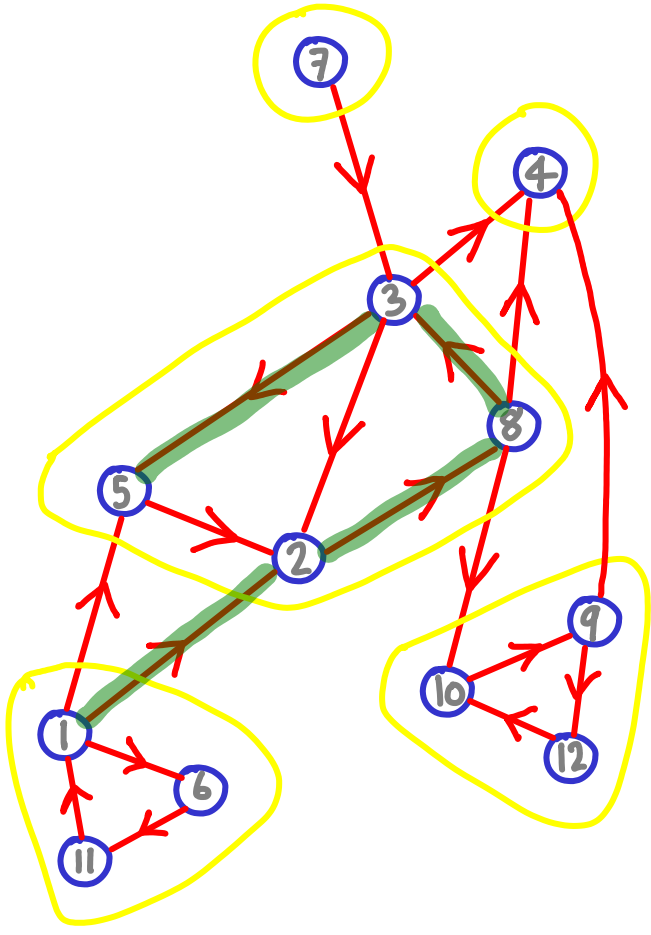
(another example)





5

finished

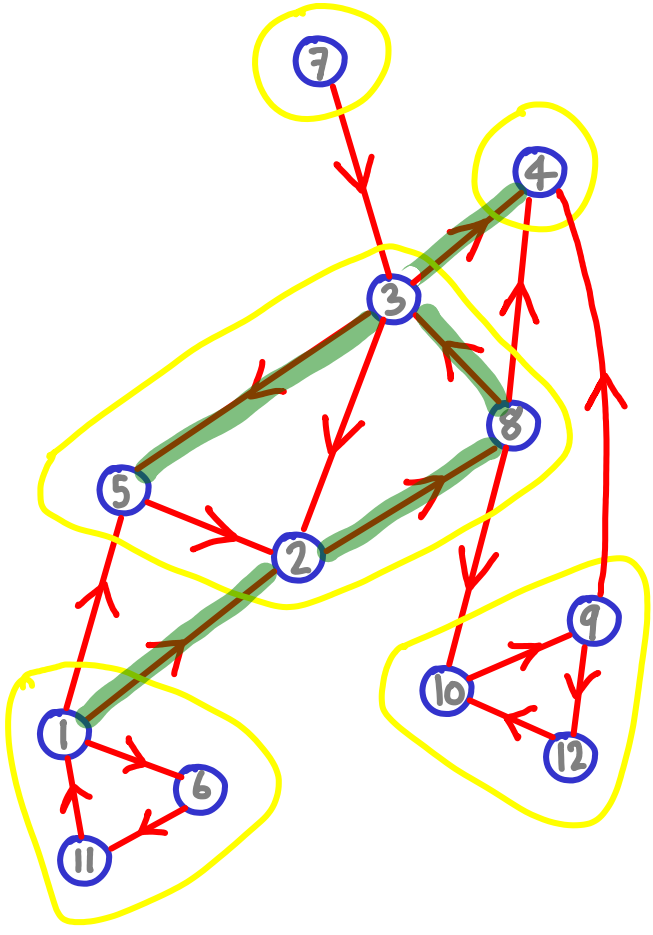


DFS 1→2→8→3→5

DFS from arbitrary vertex

4 5

finished



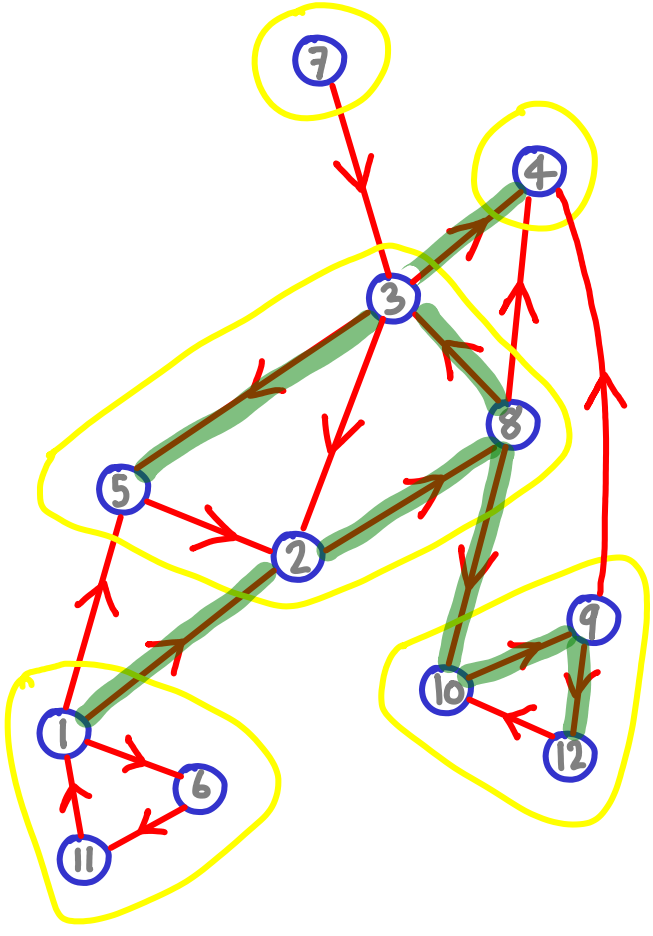
DFS 1→2→8→3→5

3→4

DFS from arbitrary vertex

12 3 4 5

finished



DFS from arbitrary vertex

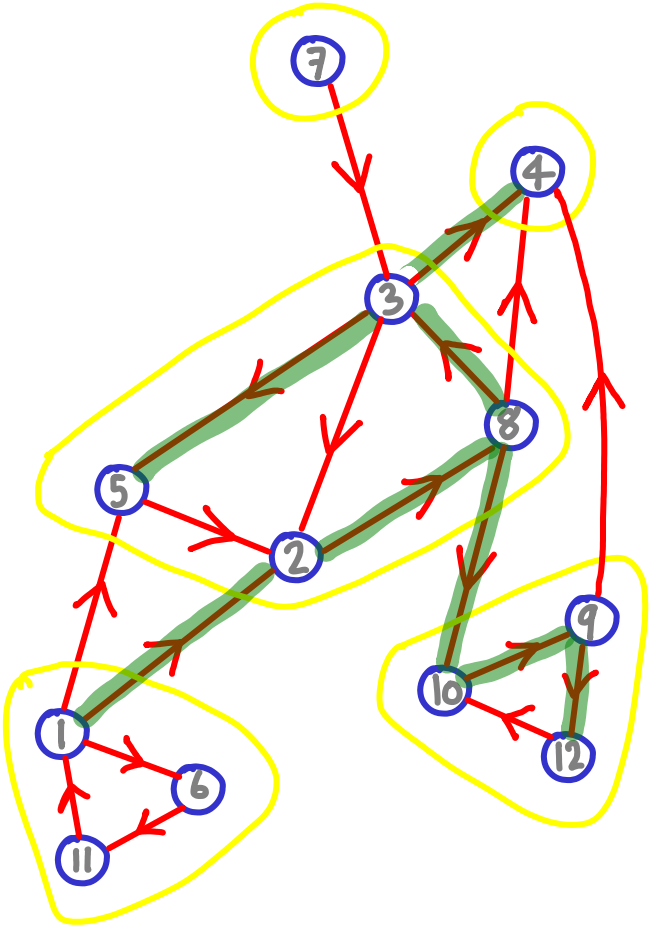
DFS 1→2→8→3→5

3→4

8→10→9→12

2 8 10 9 12 3 4 5

finished



DFS from arbitrary vertex

DFS 1→2→8→3→5

3→4

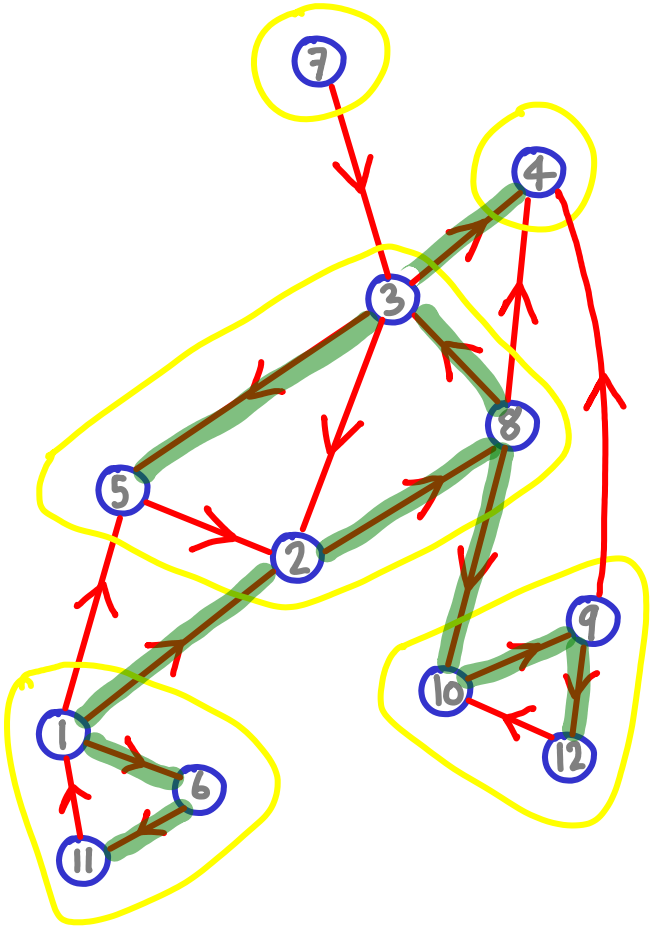
8→10→9→12

2

1

2 8 10 9 12 3 4 5

finished



DFS from arbitrary vertex

DFS 1→2→8→3→5

3→4

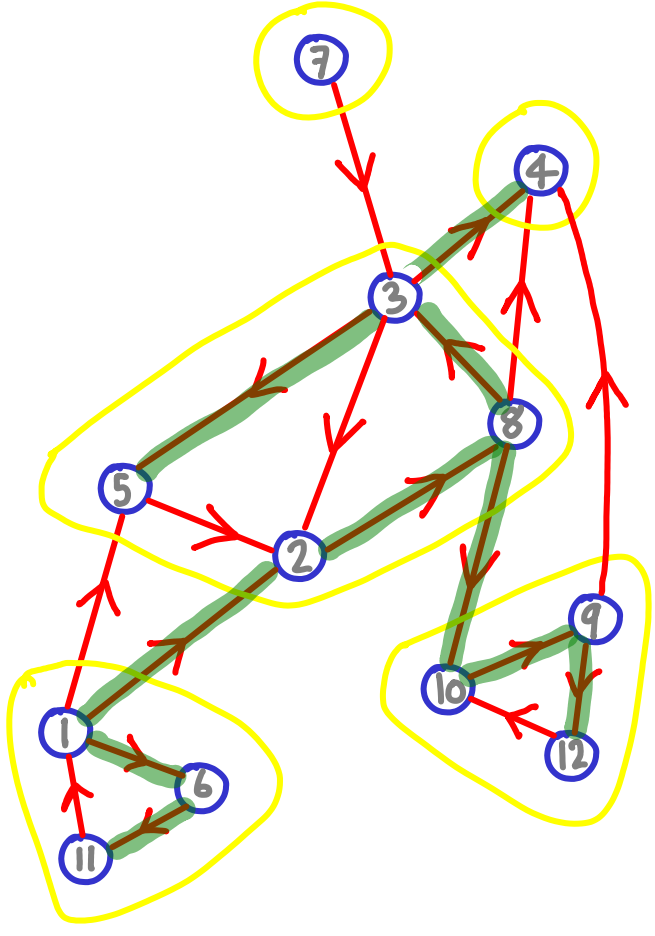
8→10→9→12

2

1→6→11

1 6 11 2 8 10 9 12 3 4 5

finished



DFS 1→2→8→3→5

3→4

8→10→9→12

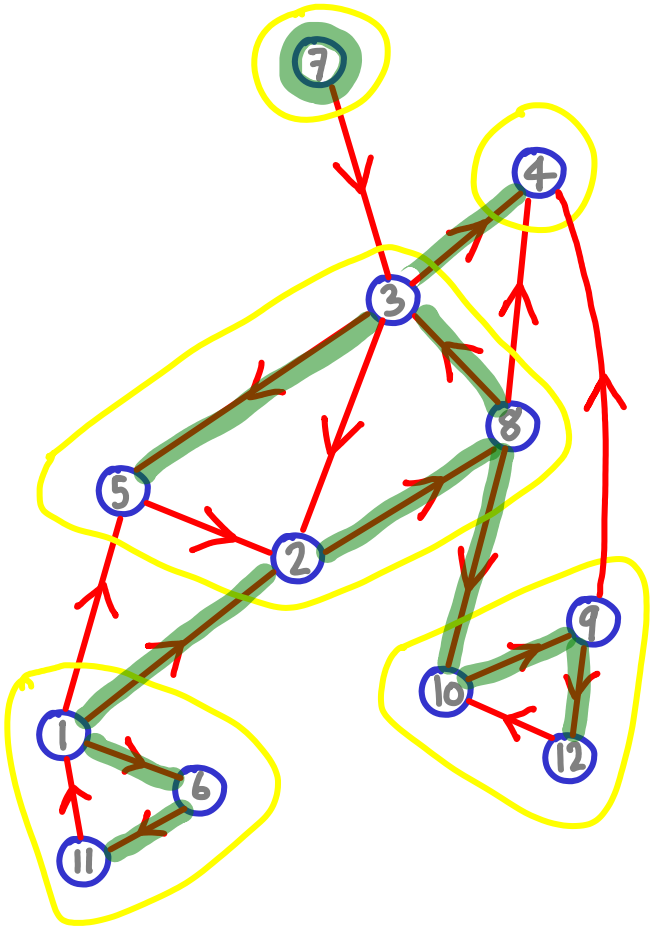
2

1→6→11

DFS from arbitrary vertex

7 1 6 11 2 8 10 9 12 3 4 5

finished



DFS from arbitrary vertex

DFS 1→2→8→3→5

3→4

8→10→9→12

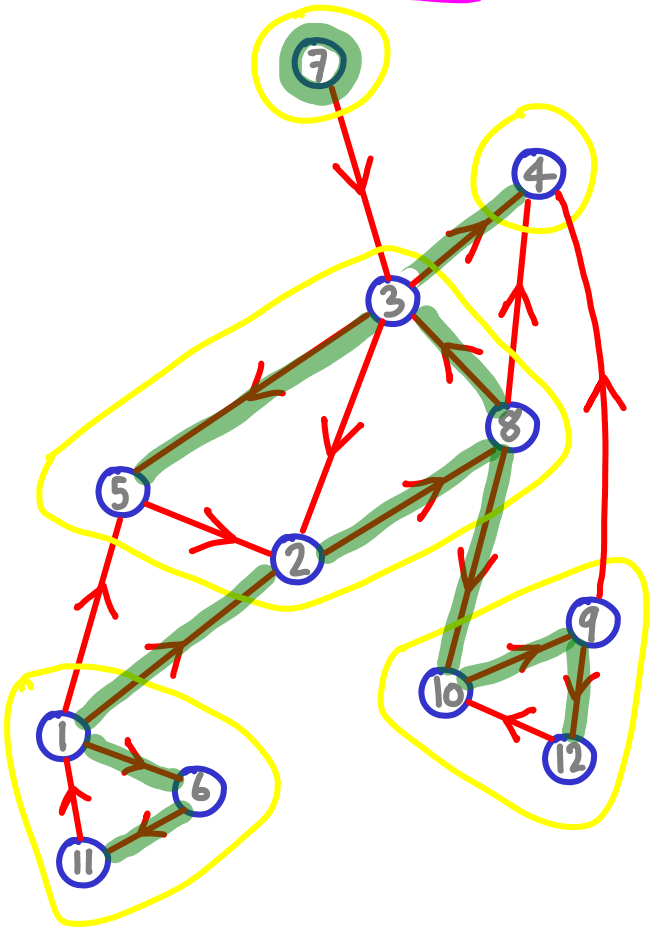
2

1→6→11

7

Finishing times: 7 1 6 11 2 8 10 9 12 3 4 5

finished first

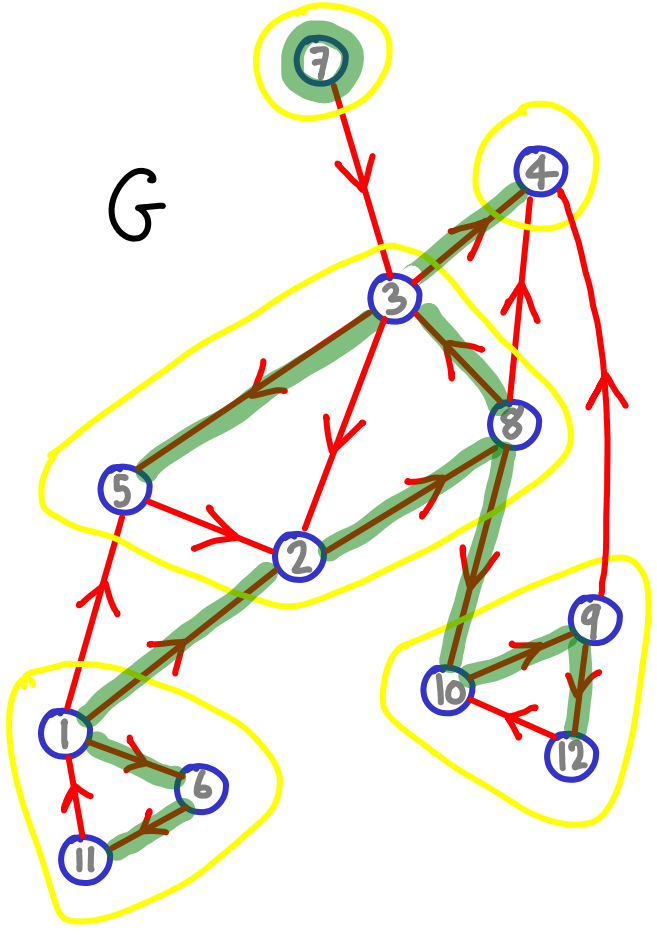


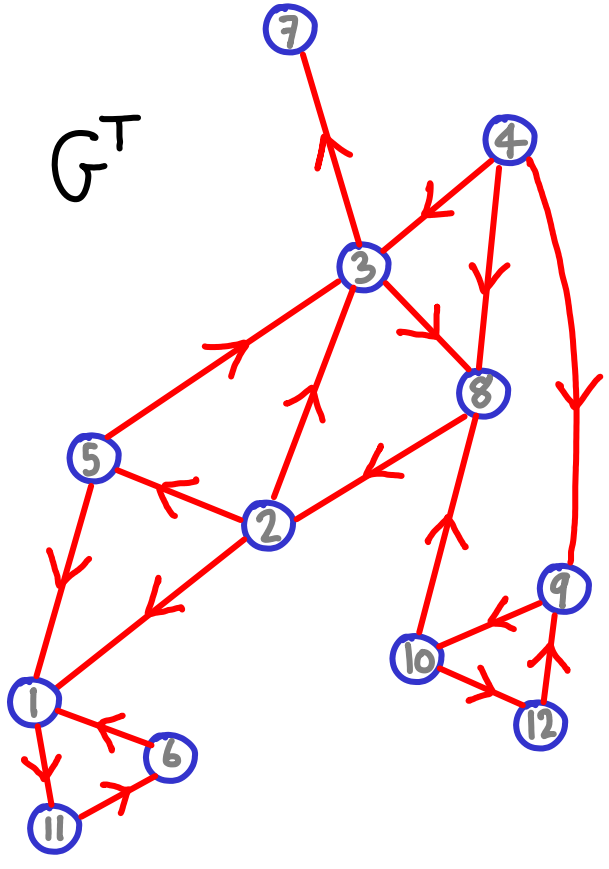
Finishing times don't give us the scc but they help a lot.

DFS from arbitrary vertex

Finishing times: 7 1 6 11 2 8 10 9 12 3 4 5

finished first



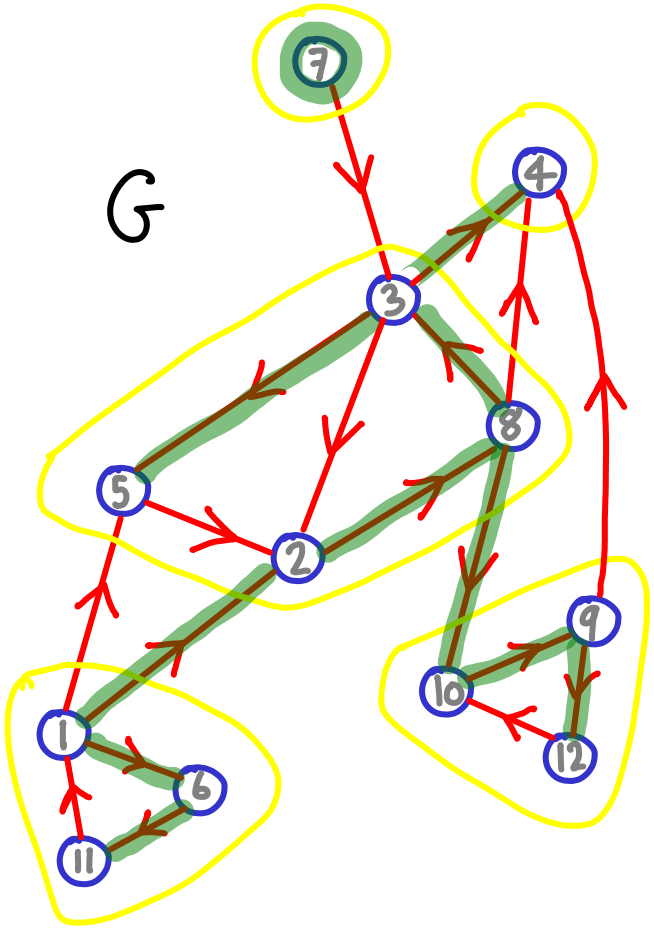


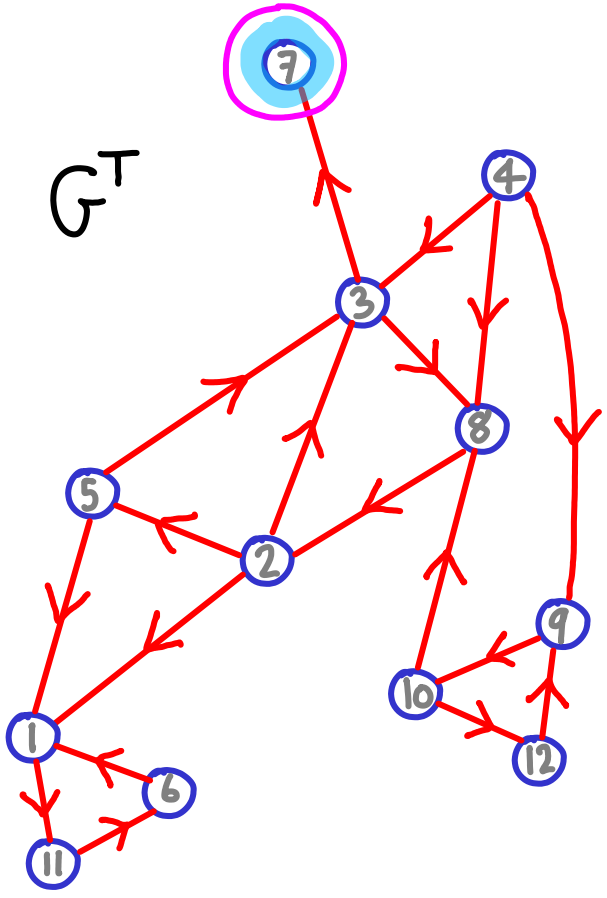
DFS from arbitrary vertex

Finishing times: 7 1 6 11 2 8 10 9 12 3 4 5

finished first

DFS(G^T) in this order





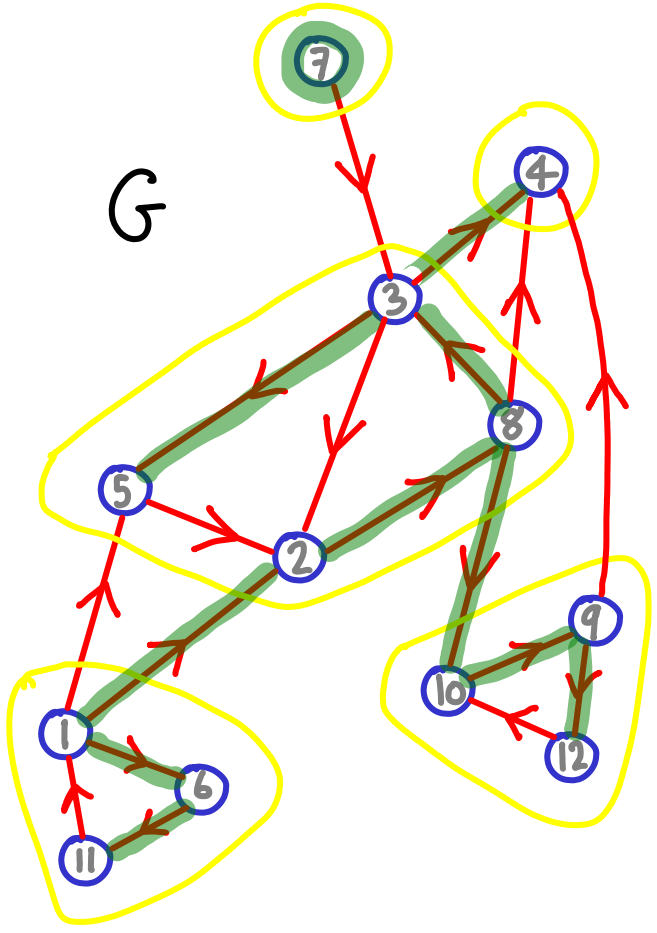
DFS from arbitrary vertex

Finishing times: 7 1 6 11 2 8 10 9 12 3 4 5

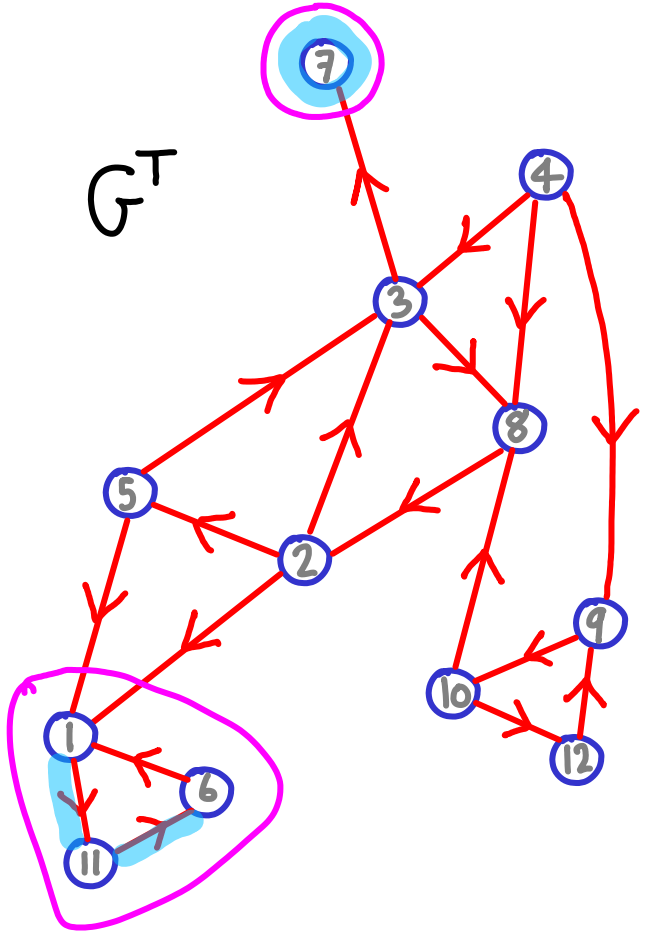
finished first

DFS(G^T) in this order

G



G^T

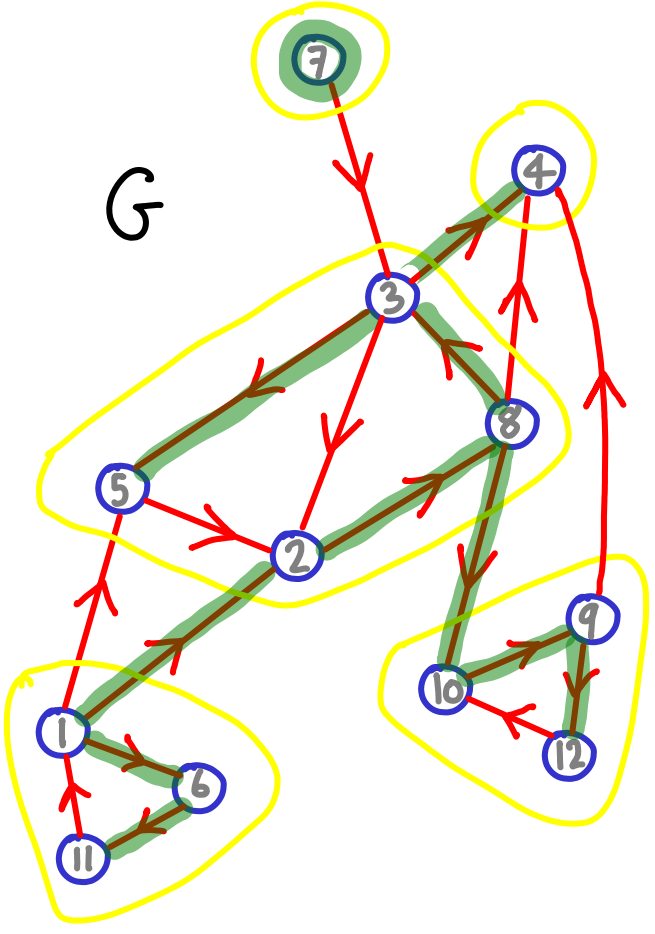


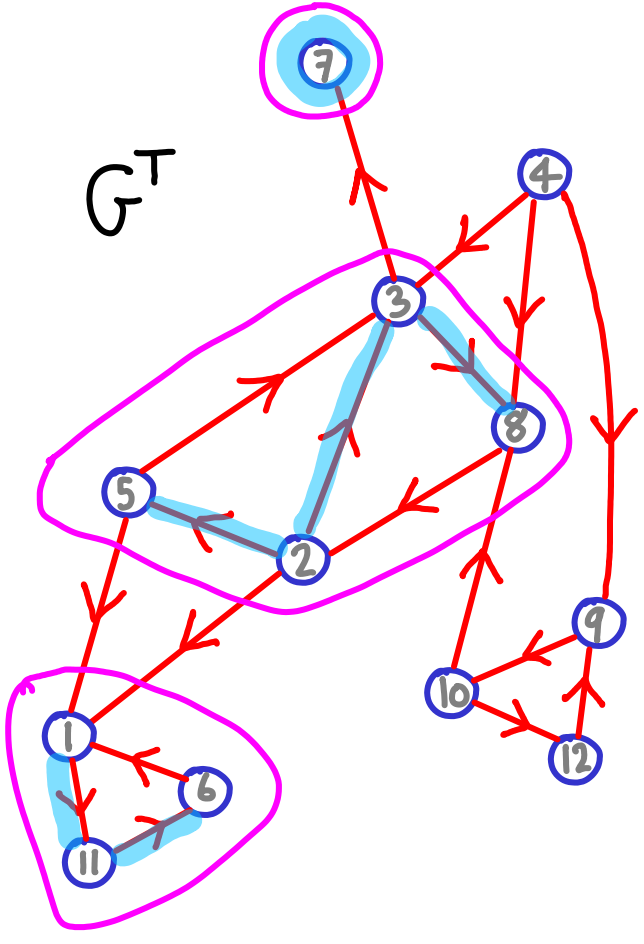
DFS from arbitrary vertex

Finishing times: 7 | 1 6 11 | 2 8 10 9 12 | 3 | 4 | 5

finished first

DFS(G^T) in this order





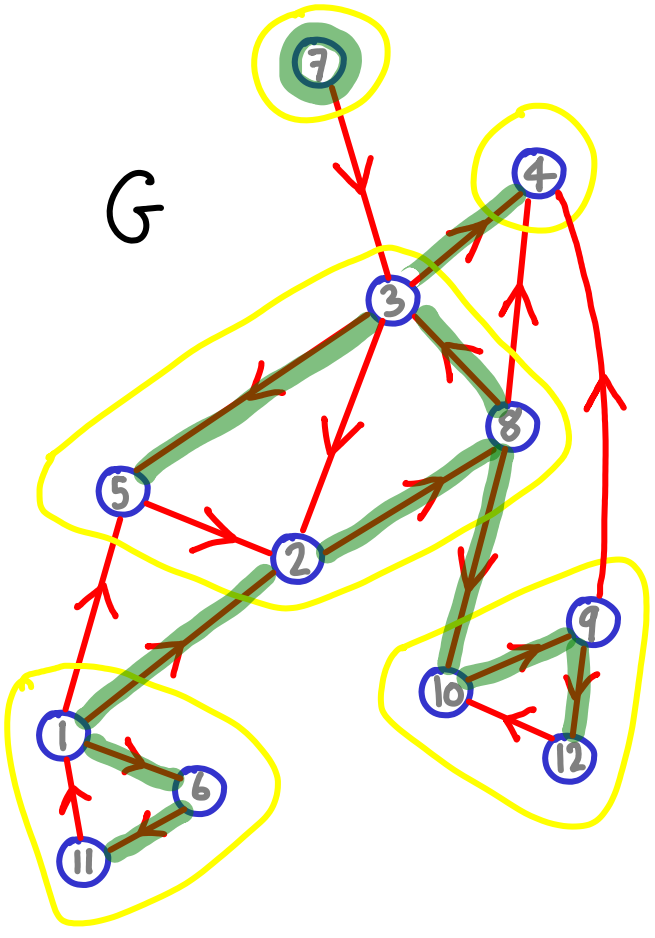
DFS from arbitrary vertex

Finishing times: 7 | 1 6 11 | 2 8 | 10 9 12 | 3 | 4 | 5

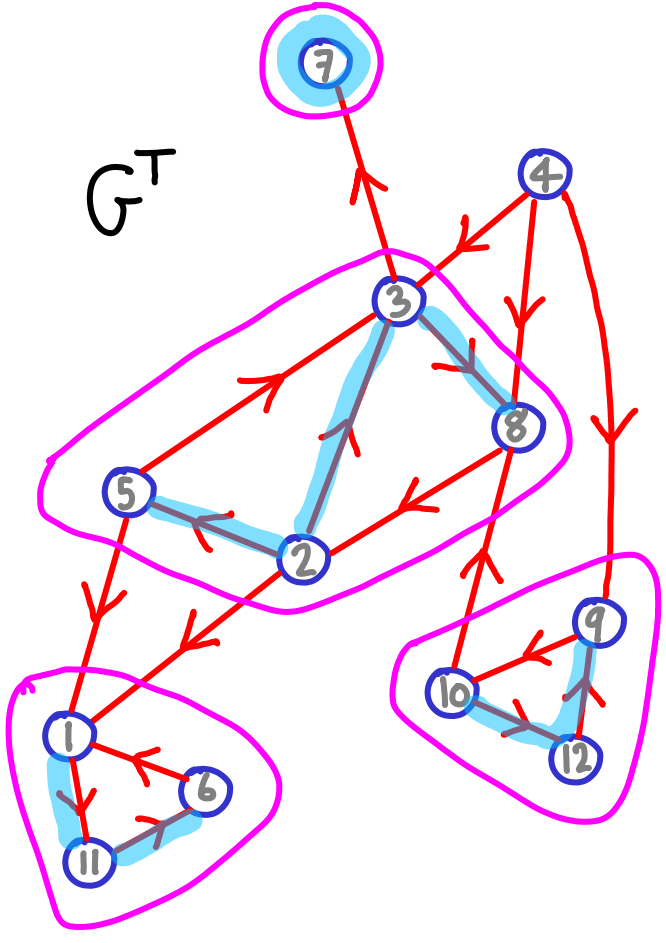
finished first

DFS(G^T) in this order

G



G^T



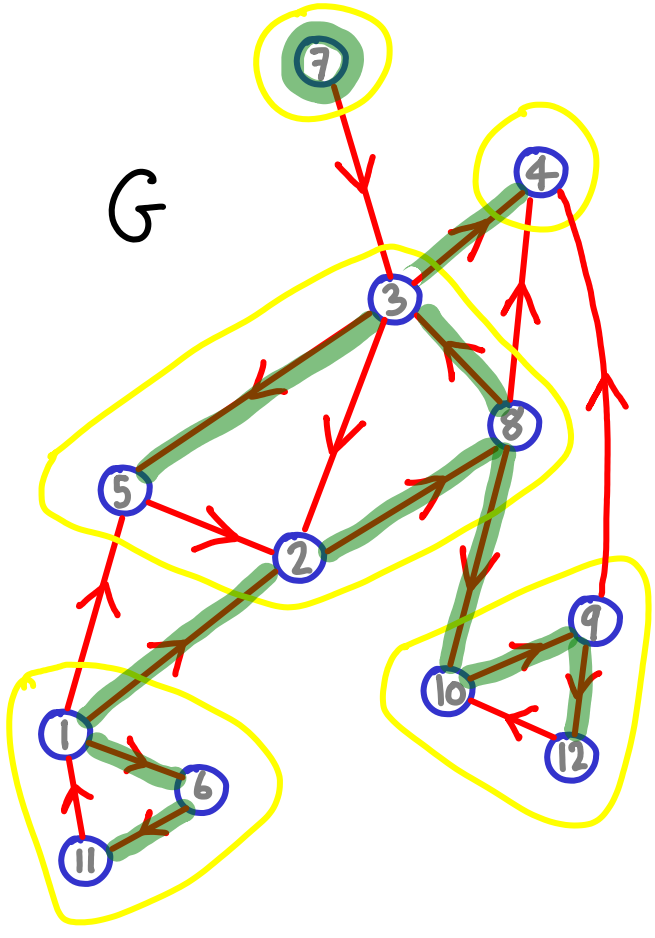
DFS from arbitrary vertex

Finishing times: 7 | 1 6 11 | 2 8 | 10 9 12 | 3 | 4 | 5

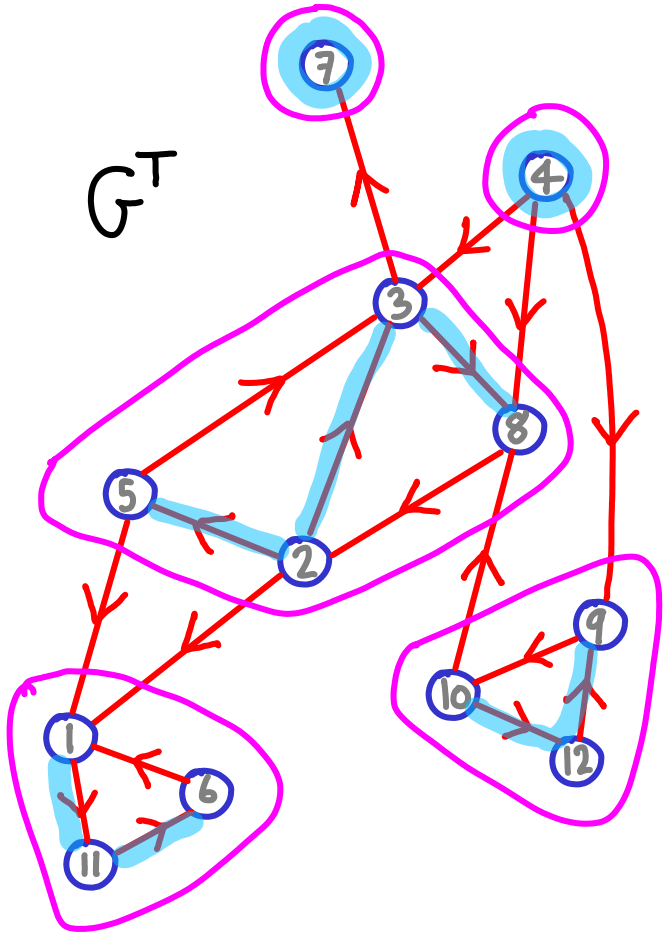
finished first

DFS(G^T) in this order

G



G^T



DFS from arbitrary vertex

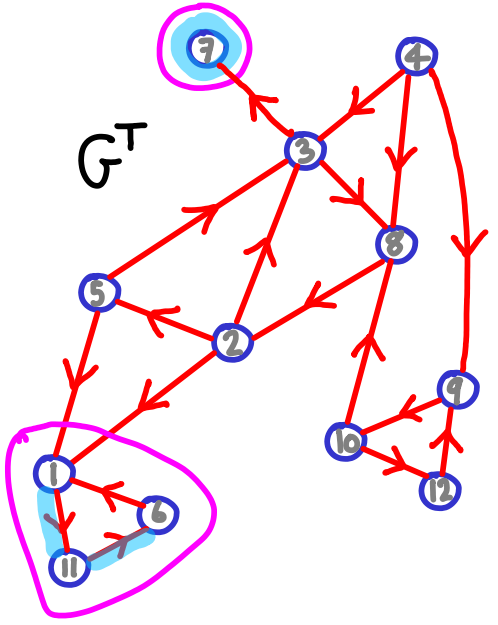
Correctness

7 | 1 6 11 | 2 | 8 | 10 | 9 | 12 | 3 | 4 | 5

u

Claim:

When processing a vertex u (e.g. 2) in DFS(G^T),
such that u is the leftmost vertex of SCC(u),
u will find all of SCC(u) and nothing else



Correctness

7 | 1 6 11 | 2 8 10 9 12 3 4 5

u

Claim:

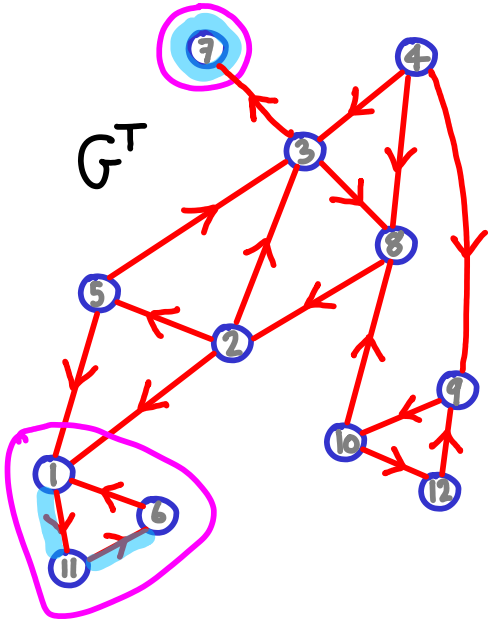
When processing a vertex u (e.g. 2) in DFS(G^T), such that u is the leftmost vertex of SCC(u),

u will find all of SCC(u) and nothing else

part 1 part 2

Proof:

part 1: ?



Correctness

7 | 1 6 11 | 2 | 8 | 10 | 9 | 12 | 3 | 4 | 5

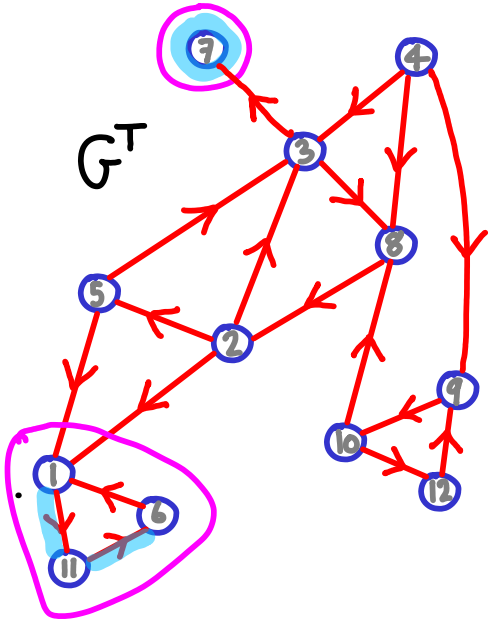
u

Claim:

When processing a vertex u (e.g. 2) in DFS(G^T), such that u is the leftmost vertex of SCC(u),

u will find all of SCC(u) and nothing else

part 1 part 2



Proof:

part 1: Will SCC(u) be unmarked ?

Correctness

7 | 1 6 11 | 2 | 8 | 10 | 9 | 12 | 3 | 4 | 5

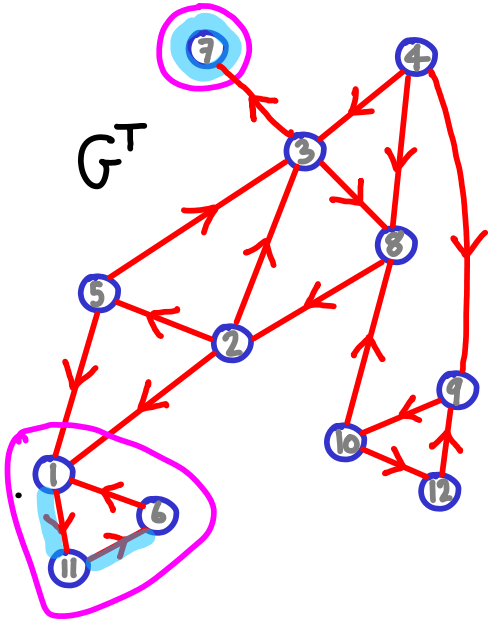
u

Claim:

When processing a vertex u (e.g. 2) in DFS(G^T), such that u is the leftmost vertex of SCC(u),

u will find all of SCC(u) and nothing else

part 1 part 2



Proof:

part 1: By induction, SCC(u) will be unmarked

↳ nothing left of u has marked SCC(u)

Correctness

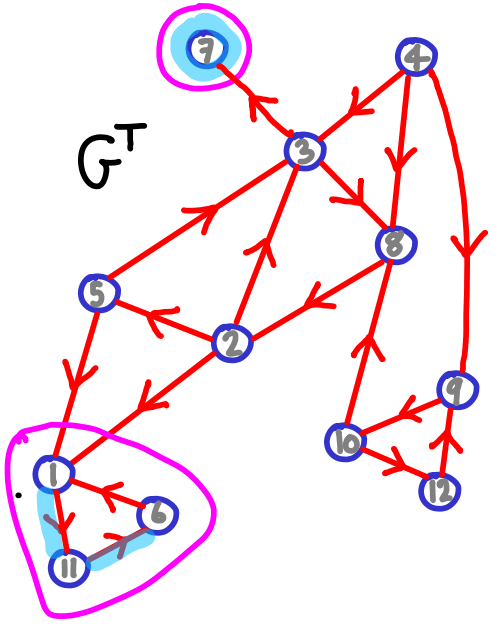
7 | 1 6 11 | 2 | 8 | 10 | 9 | 12 | 3 | 4 | 5

u

Claim:

When processing a vertex u (e.g. 2) in DFS(G^T), such that u is the leftmost vertex of SCC(u), u will find all of SCC(u) and nothing else

part 1 part 2



Proof:

part 1: By induction, SCC(u) will be unmarked

↳ nothing left of u has marked SCC(u)

part 2: ?

Correctness

7 | 1 6 11 | 2 | 8 | 10 | 9 | 12 | 3 | 4 | 5

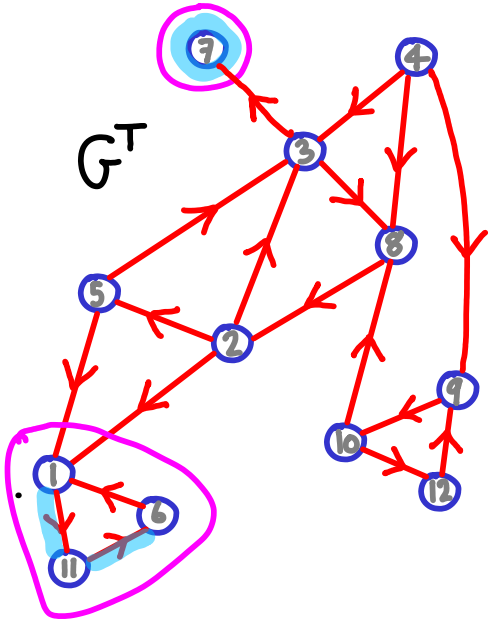
u

Claim:

When processing a vertex u (e.g. 2) in DFS(G^T), such that u is the leftmost vertex of SCC(u),

u will find all of SCC(u) and nothing else

part 1 part 2



Proof:

part 1: By induction, SCC(u) will be unmarked

↳ nothing left of u has marked SCC(u)

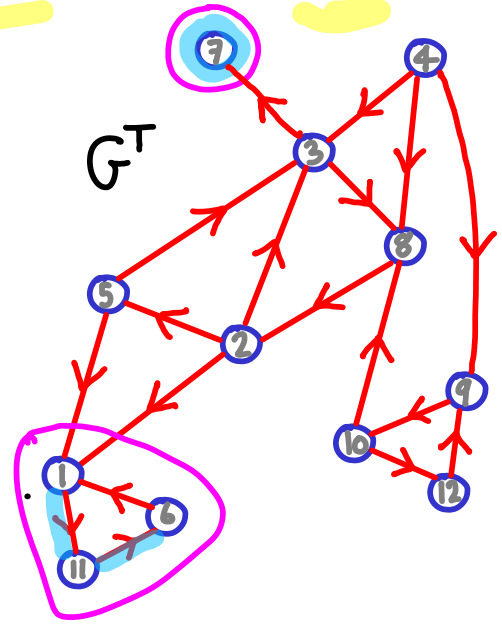
part 2: By contradiction...

Correctness

7 | 1 6 11 | 2 | 8 | 10 | 9 | 12 | 3 | 4 | 5

u

× × × ×



Claim:

When processing a vertex u (e.g. 2) in DFS(G^T), such that u is the leftmost vertex of SCC(u),

u will find all of SCC(u) and nothing else

part 1 part 2

Proof:

part 1: By induction, SCC(u) will be unmarked

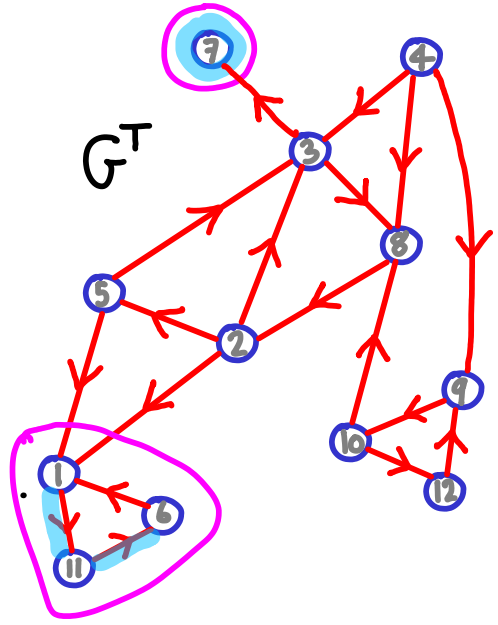
↳ nothing left of u has marked SCC(u)

part 2: Suppose $\exists$ edge SCC(u) → x in G^T, s.t. $x \notin$ SCC(u), unmarked, & right of u.

Correctness

7 | 1 6 11 | 2 | 8 | 10 | 9 | 12 | 3 | 4 | 5

u, ×, ×, ×, ×



Claim:

When processing a vertex u (e.g. 2) in DFS(G^T), such that u is the leftmost vertex of SCC(u), u will find all of SCC(u) and nothing else

part 1 part 2

Proof:

part 1: By induction, SCC(u) will be unmarked

↳ nothing left of u has marked SCC(u)

part 2: Suppose $\exists$ edge SCC(u) → x in G^T, s.t. $x \notin$ SCC(u), unmarked, & right of u.

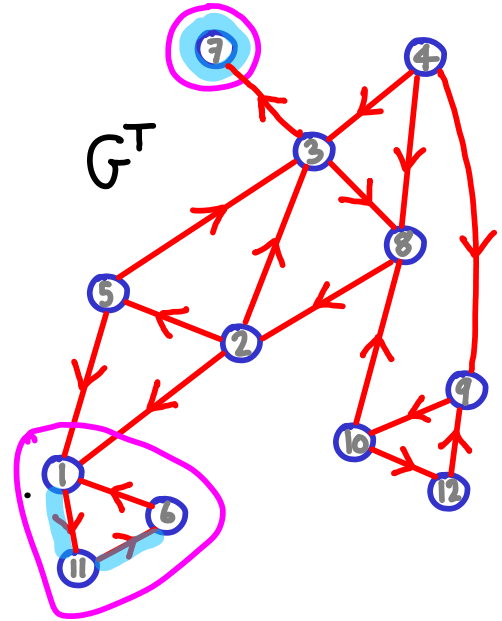
↳ Then, $\exists$ edge from x to SCC(u) in G.

what else?

Correctness

7 | 1 6 11 | 2 | 8 | 10 | 9 | 12 | 3 | 4 | 5

u × × × ×



Claim:

When processing a vertex u (e.g. 2) in DFS(G^T), such that u is the leftmost vertex of SCC(u),

u will find all of SCC(u) and nothing else

part 1 part 2

Proof:

part 1: By induction, SCC(u) will be unmarked

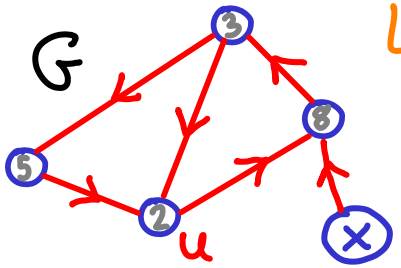
↳ nothing left of u has marked SCC(u)

part 2: Suppose $\exists$ edge SCC(u) $\to x$ in G^T, s.t. $x \notin$ SCC(u), unmarked, & right of u.

↳ Then, $\exists$ edge from x to SCC(u) in G.

↳ $\nexists$ path from SCC(u) to x in G [otherwise $x \in$ SCC(u), contradiction]

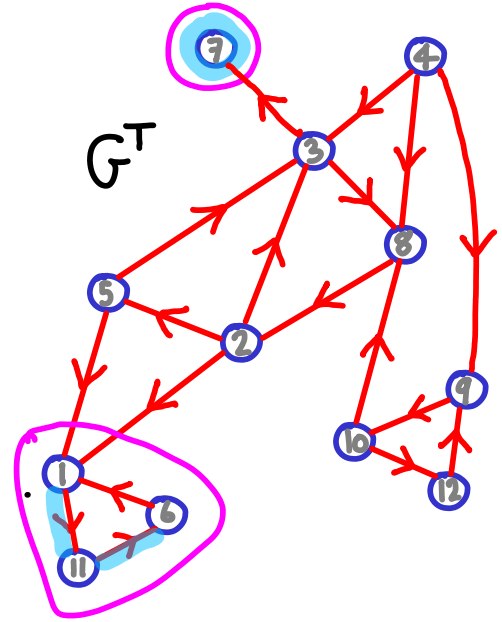
↳ implies...?



Correctness

7 | 1 6 11 | 2 | 8 | 10 | 9 | 12 | 3 | 4 | 5

u × × × ×



Claim:

When processing a vertex u (e.g. 2) in DFS(G^T), such that u is the leftmost vertex of SCC(u),

u will find all of SCC(u) and nothing else

part 1 part 2

Proof:

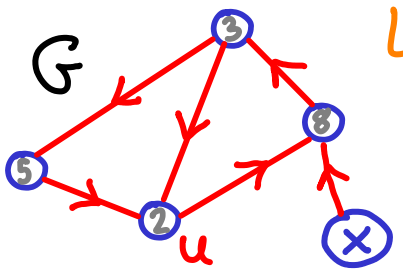
part 1: By induction, SCC(u) will be unmarked

↳ nothing left of u has marked SCC(u)

part 2: Suppose $\exists$ edge SCC(u) $\to x$ in G^T, s.t. $x \notin$ SCC(u), unmarked, & right of u.

↳ Then, $\exists$ edge from x to SCC(u) in G.

↳ $\nexists$ path from SCC(u) to x in G [otherwise $x \in$ SCC(u), contradiction]



Then in DFS(G) x will finish after u

Why?

Correctness

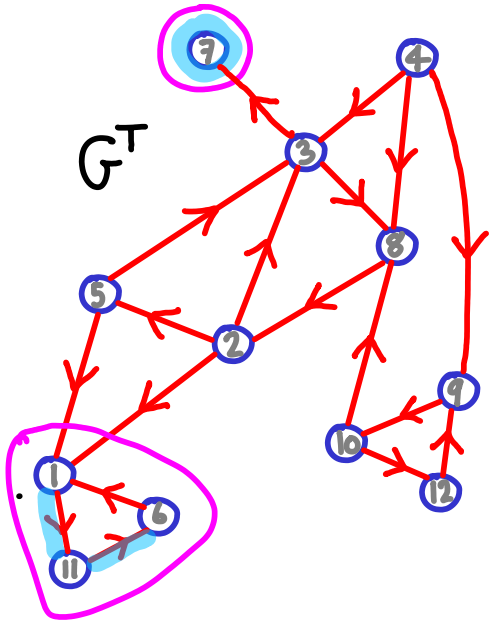
7 | 1 6 11 | 2 | 8 | 10 | 9 | 12 | 3 | 4 | 5

u, ×, ×, ×, ×

Claim:

When processing a vertex u (e.g. 2) in DFS(G^T), such that u is the leftmost vertex of SCC(u), u will find all of SCC(u) and nothing else

part 1 part 2



Proof:

part 1: By induction, SCC(u) will be unmarked

↳ nothing left of u has marked SCC(u)

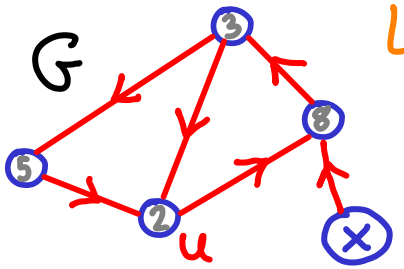
part 2: Suppose $\exists$ edge SCC(u) → x in G^T, s.t. $x \notin$ SCC(u), unmarked, & right of u.

↳ Then, $\exists$ edge from x to SCC(u) in G.

↳ $\nexists$ path from SCC(u) to x in G [otherwise $x \in$ SCC(u), contradiction]

Then in DFS(G) x will finish after u

↳ trivial IF SCC(u) found first; ELSE: x not done until it finds SCC(u)



Correctness

7 | 1 6 11 | 2 | 8 | 10 | 9 | 12 | 3 | 4 | 5

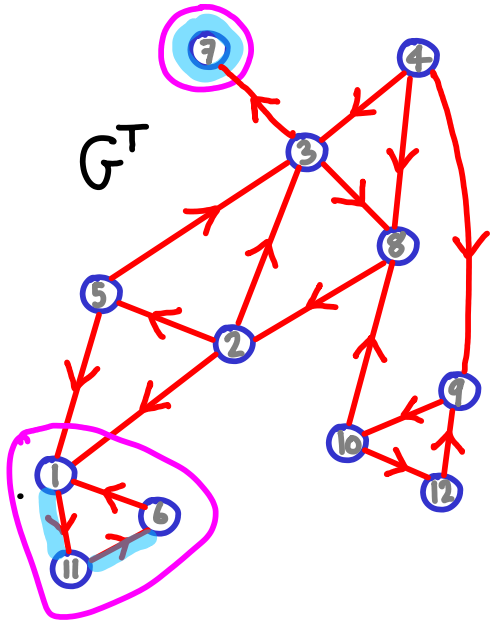
u, ×, ×, ×, ×

Claim:

When processing a vertex u (e.g. 2) in DFS(G^T), such that u is the leftmost vertex of SCC(u),

u will find all of SCC(u) and nothing else

part 1 part 2



Proof:

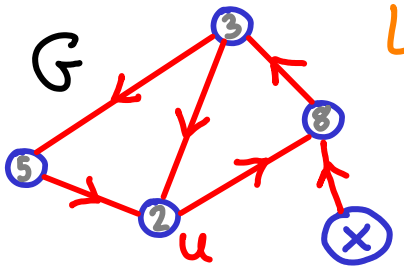
part 1: By induction, SCC(u) will be unmarked

↳ nothing left of u has marked SCC(u)

part 2: Suppose $\exists$ edge SCC(u) $\to x$ in G^T, s.t. $x \notin$ SCC(u), unmarked, & right of u.

↳ Then, $\exists$ edge from x to SCC(u) in G.

↳ $\nexists$ path from SCC(u) to x in G [otherwise $x \in$ SCC(u), contradiction]



Then in DFS(G) x will finish after u [x left of u: contradiction]

↳ trivial IF SCC(u) found first; ELSE: x not done until it finds SCC(u) □