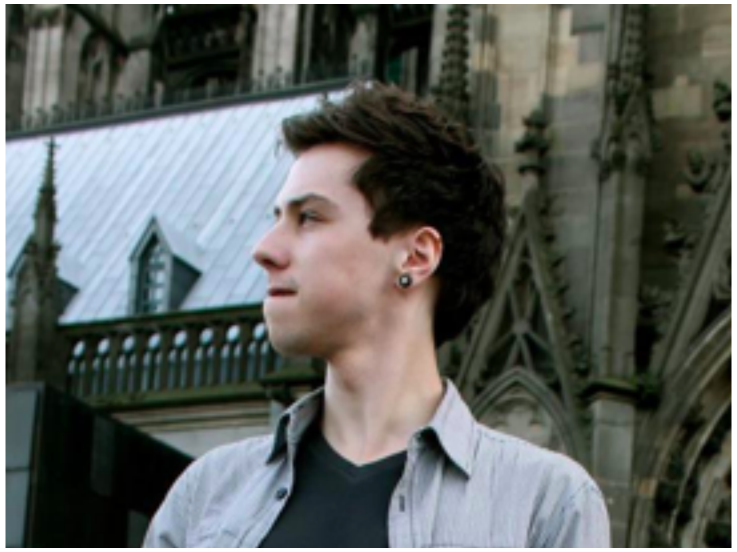
Embedding planar graphs using PQ trees

Phillip Kessels

~whoami

Phillip

fellow student,
computer science
B.Sc., ...

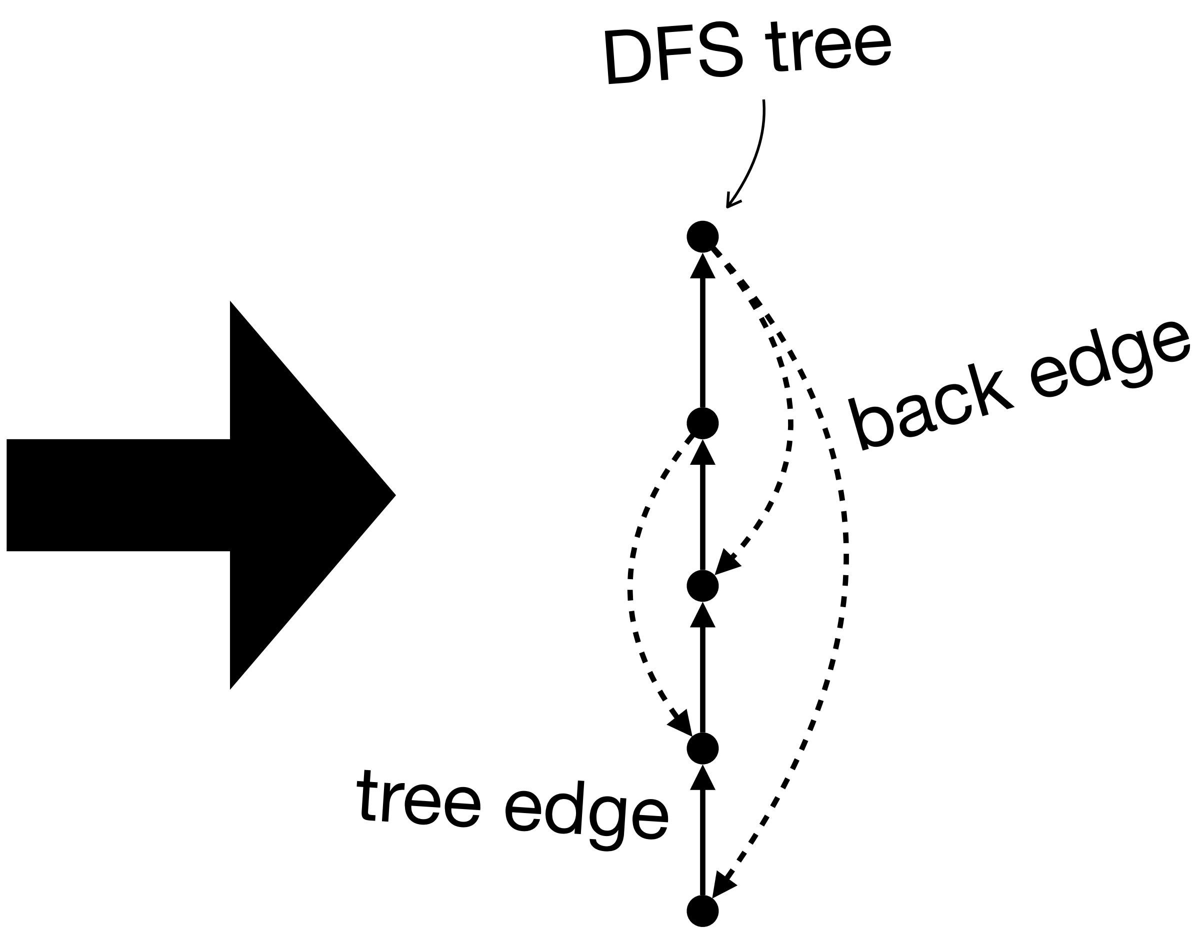


cologne!

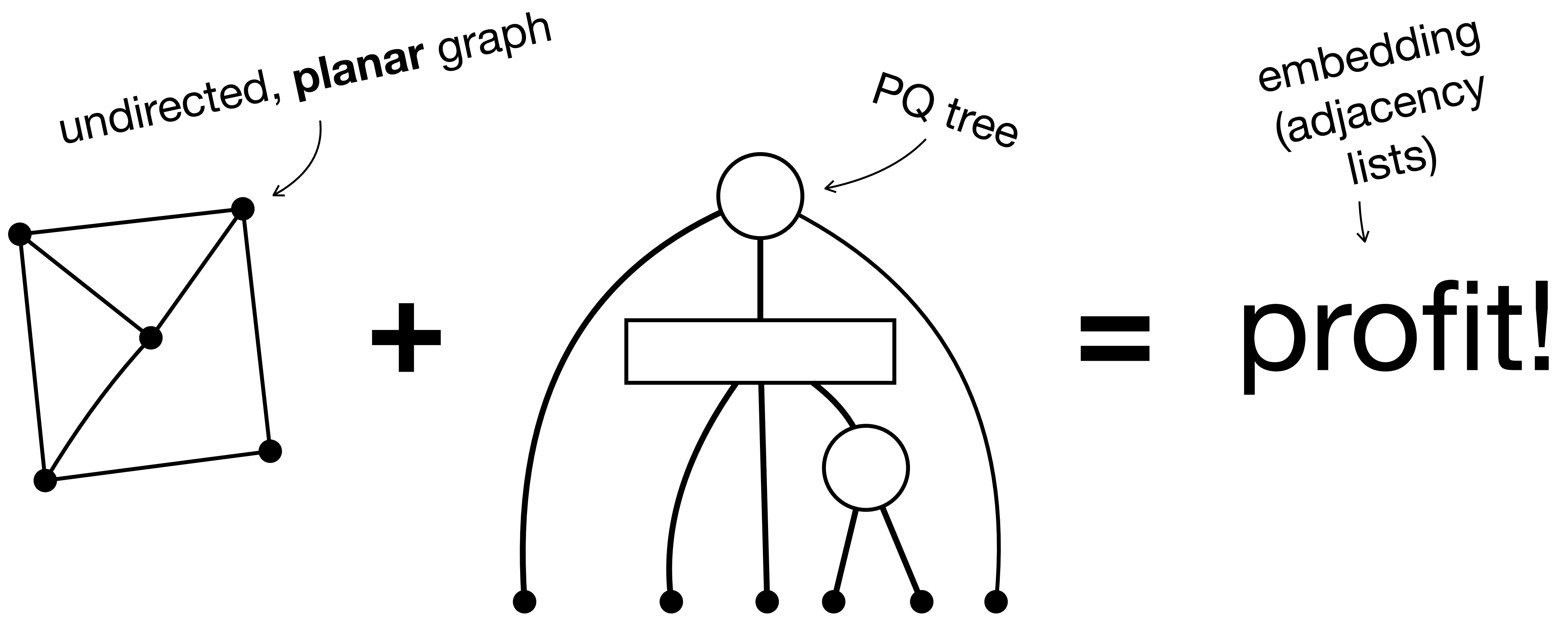
UX, Desktop/mobile
experiences, music,
theoretical computer
science, ...

Last Week

planarity testing

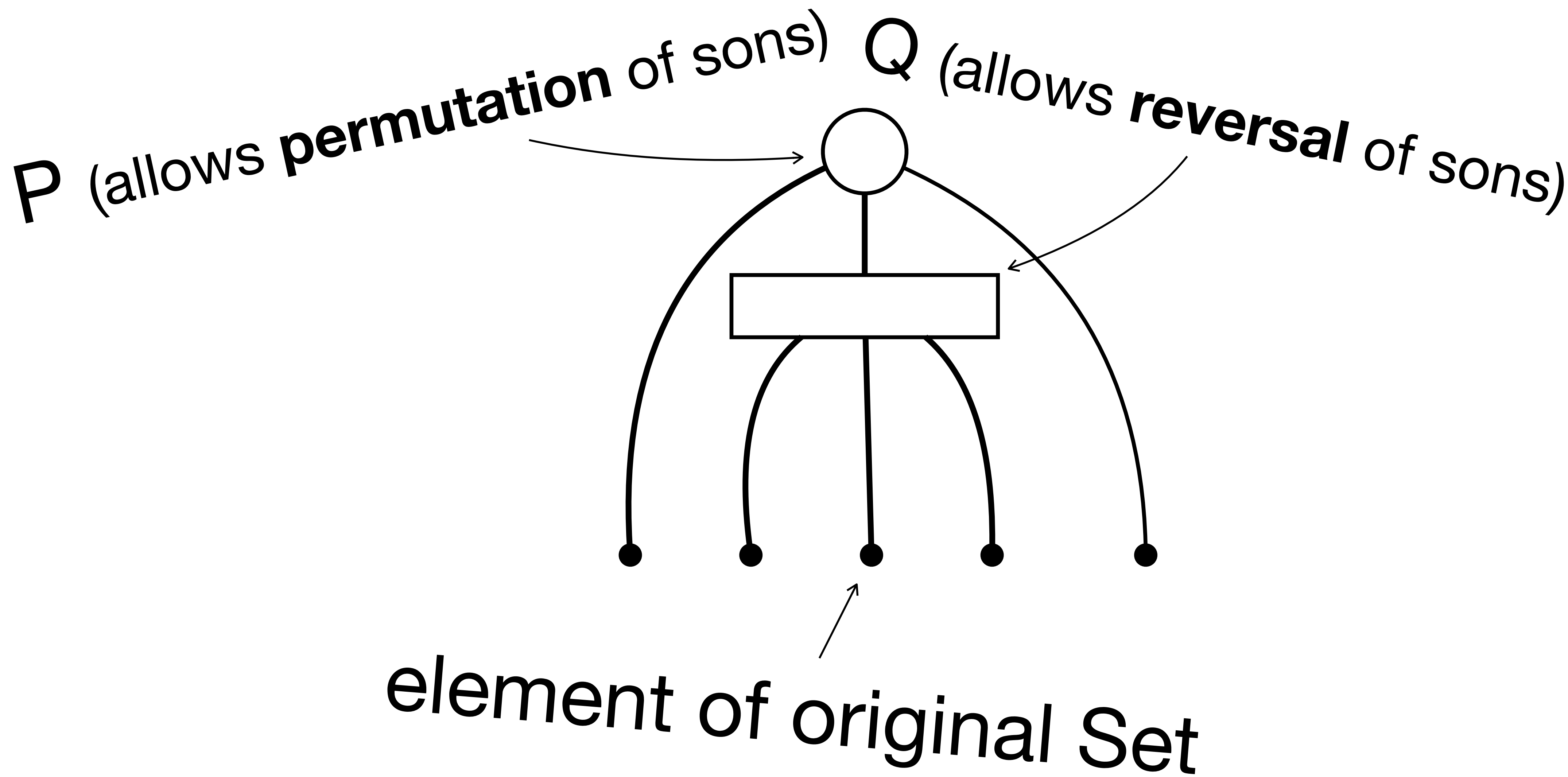


Today


undirected, **planar** graph
+
PQ tree
=
embedding
(adjacency
lists)
profit!

PQ trees

represent (specific) permutations of a Set



PQ trees - example

Phil was a scout

in morning meeting:
align **compatible** children

PQ trees - example

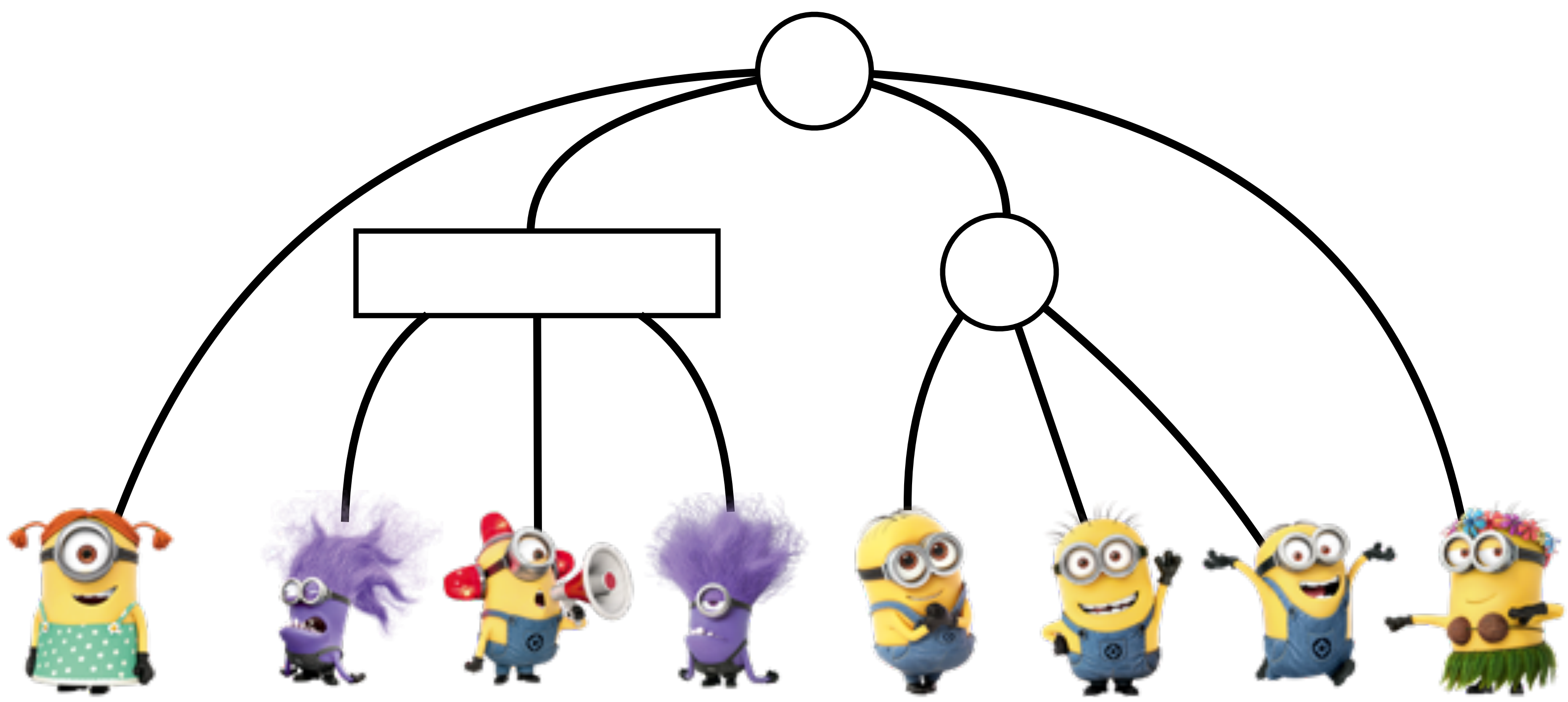
- girls = {  ,  } ← they can mix with anybody
- nice guys = {  ,  ,  }

they can mix, but want to stay together

- bad guys = {  ,  ,  }
  +supervisor

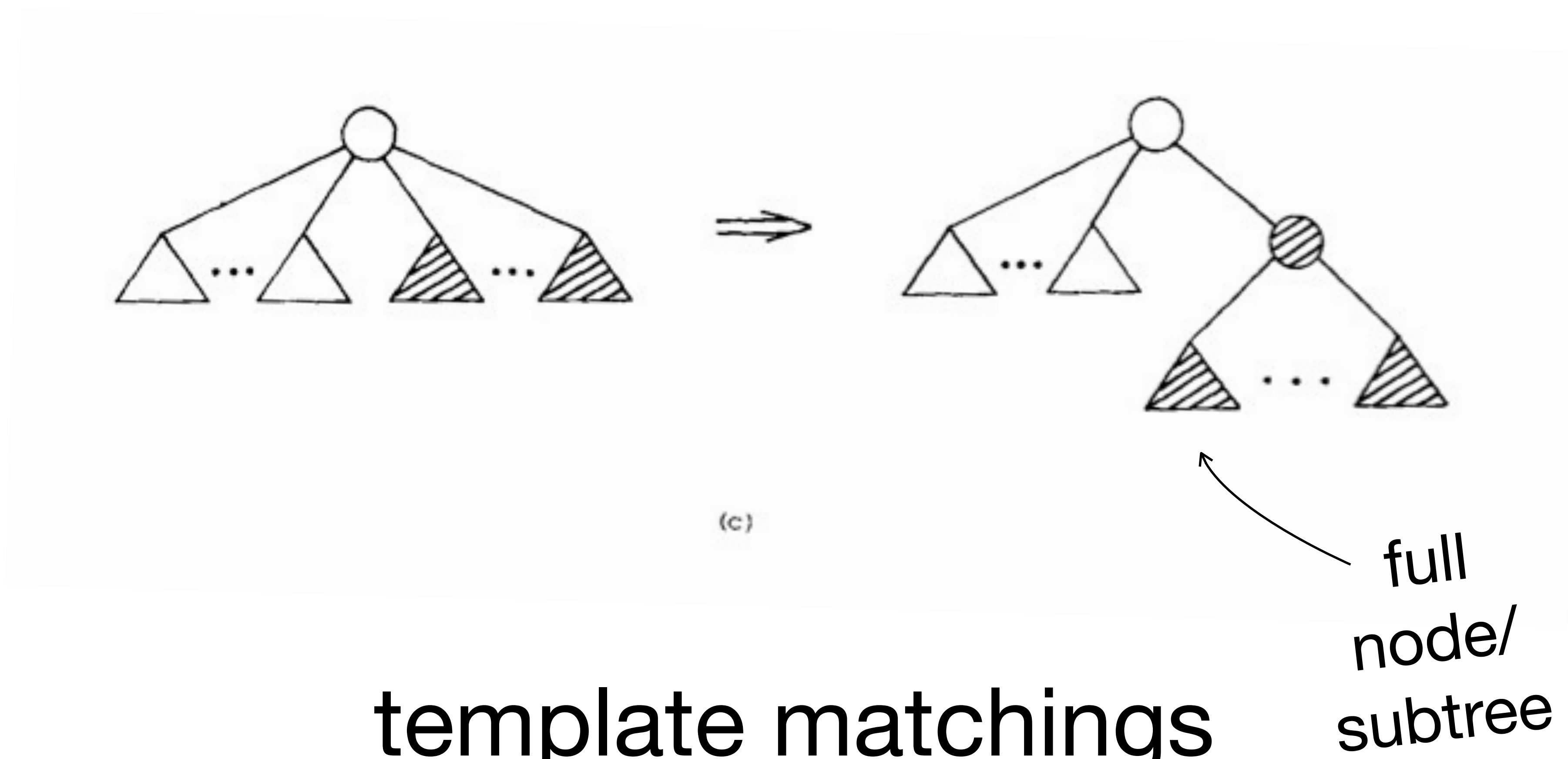
they should never meet!

PQ trees - example



PQ trees - template matchings

How can we operate on a PQ tree?
i.e. what are equivalent PQ trees?
i.e. what are „allowed“ permutations?



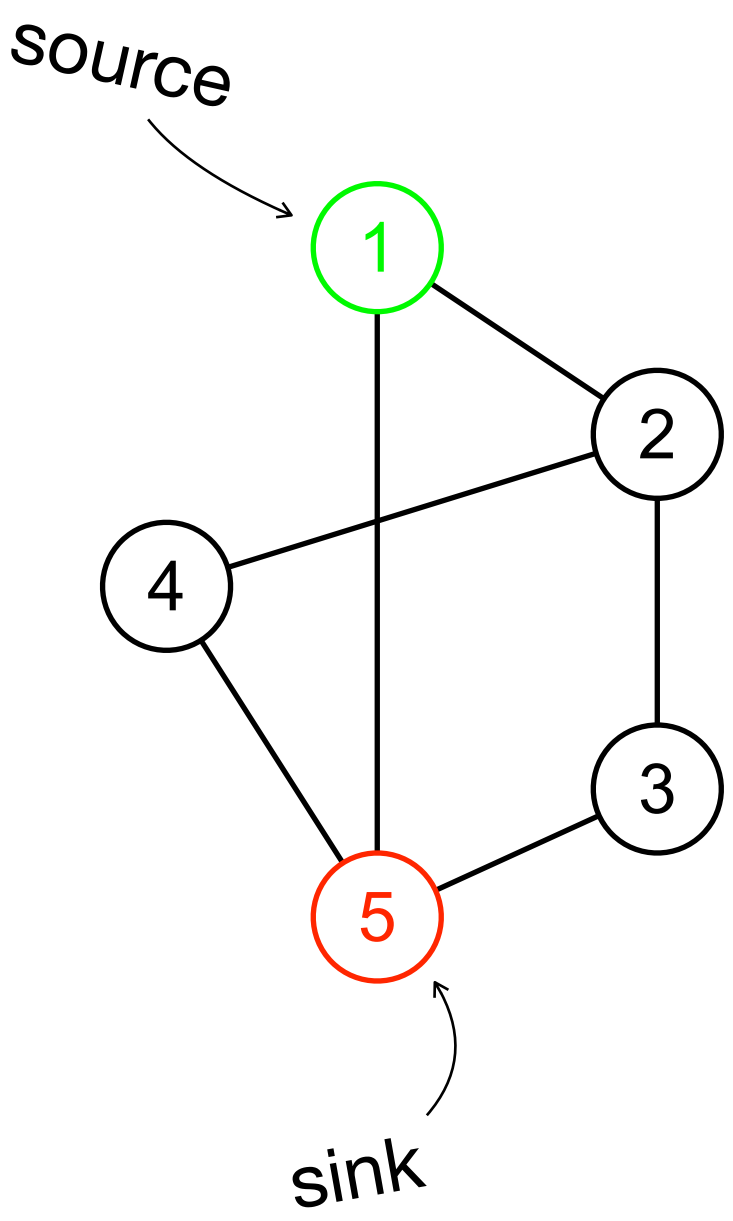
template matchings

st-numbering

- each vertex becomes a **unique** number $\in \{0, ..., n\}$
- each vertex v satisfies

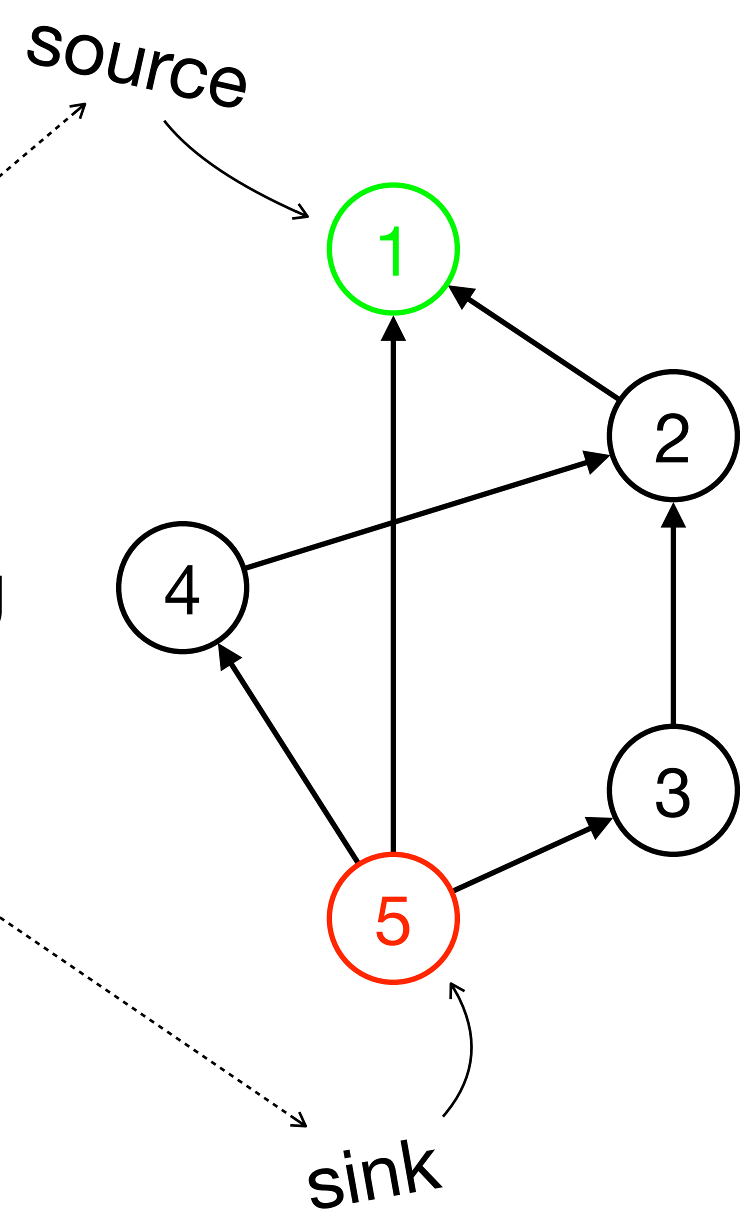
$$\exists u, w : u < v < w$$

except for **source** and **sink**

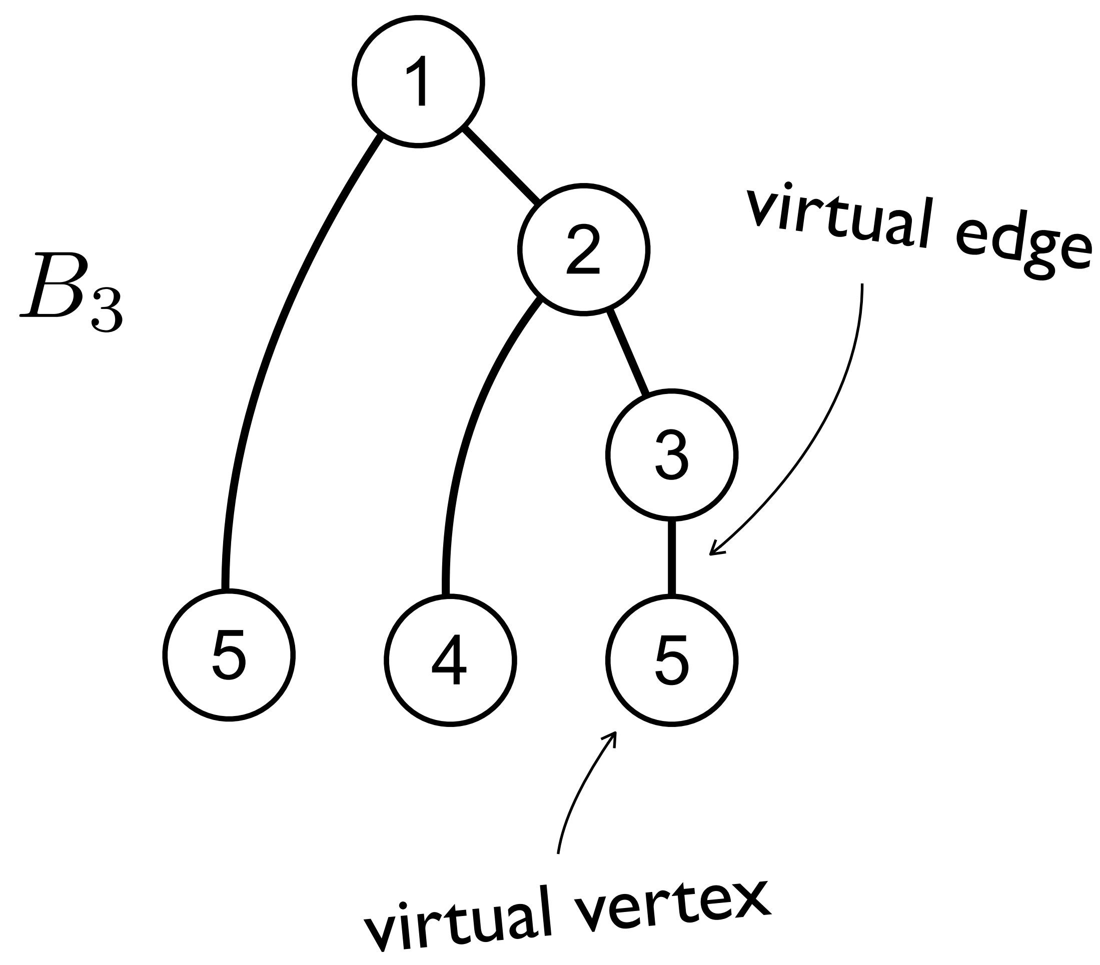


upward graph

now these become confusing



bush form



- embedding
- induced subgraph,

  but include „outgoing“ edges
- if exists $\{u, v\} \in E$ and $\{w, v\} \in E$

  where v lies „outside“ and $\{u, w\}$

  lie „inside“ then v is

  included **twice**

vertex addition algorithm

- two-phase
- tests for **planarity**
- does **not** give an embedding

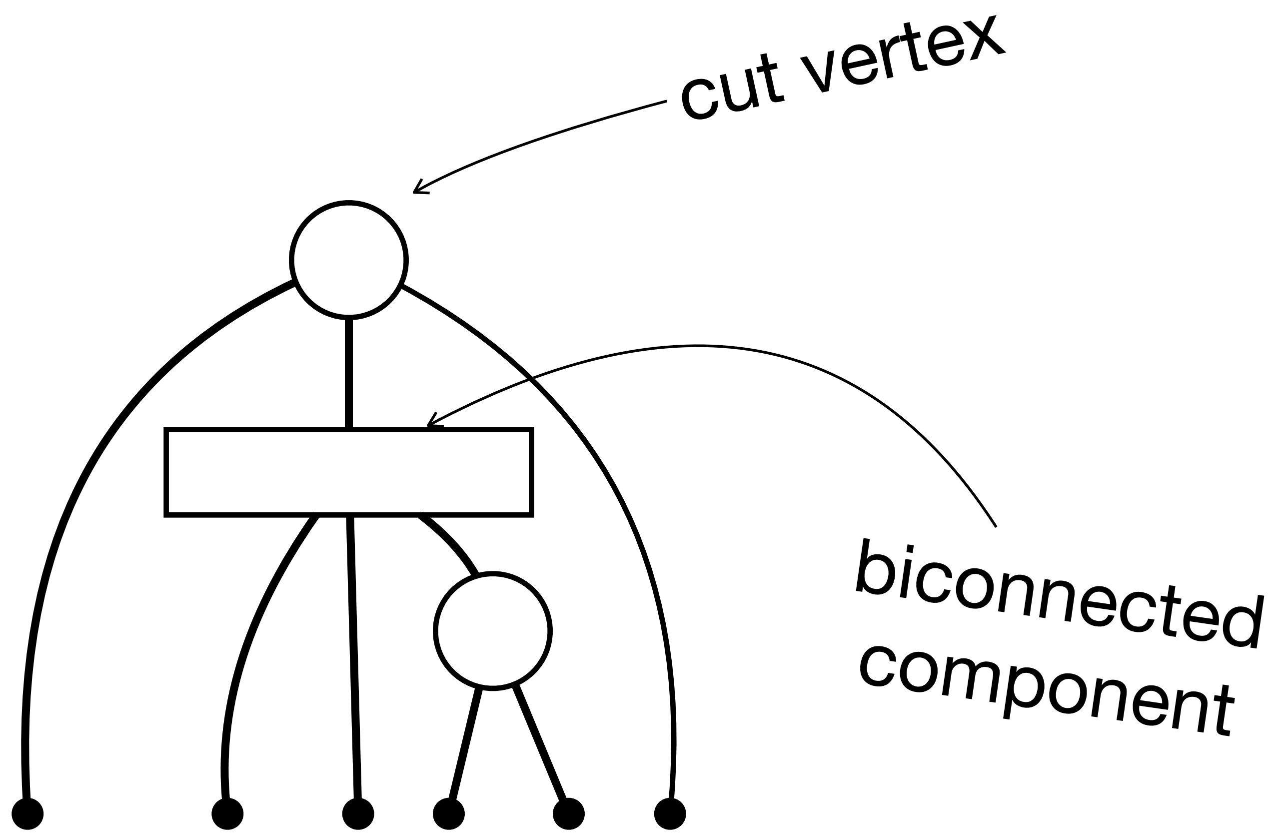
vertex addition algorithm

- **assign st-numbers** to all vertices
- construct **PQ tree** corresponding to G'_1 ← how?
- for all other vertices $v = 2, ..., n$:

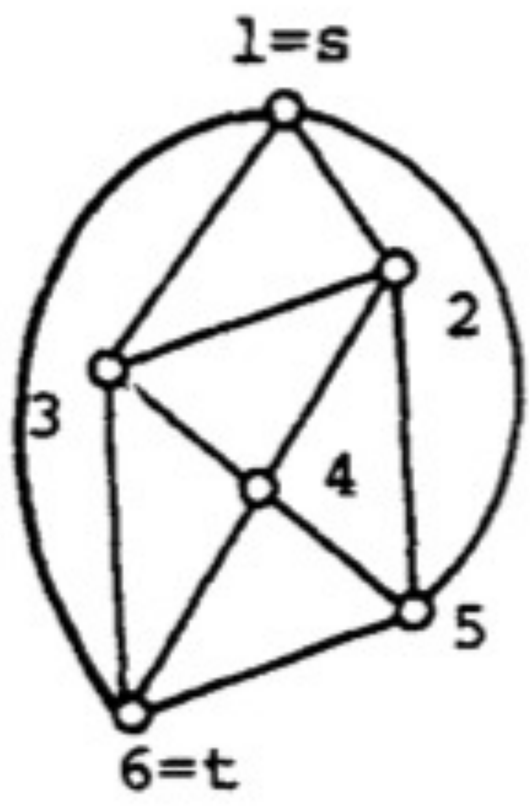
phase I
- apply **template matchings** to relevant subtree to align all vertices $v+1$ to consecutive positions ← can fail here

phase II
- replace all **full nodes** by new P node
- insert all greater vertices adjacent to $v+1$ as sons of the new node

PQ trees and graphs



vertex addition - example



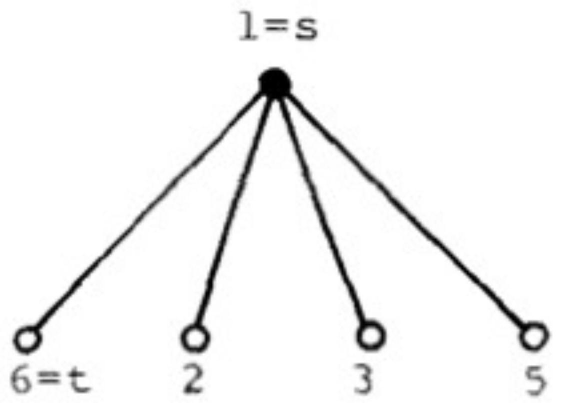
vertex addition - example

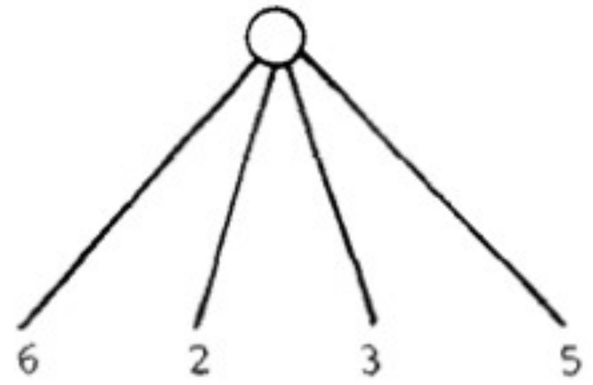
- **assign st-numbers** to all vertices ✔
- construct **PQ tree** corresponding to G'_1
- for all other vertices $v = 2, ..., n:$

phase I {
- apply **template matchings** to relevant subtree to align all vertices $v+1$ to consecutive positions

phase II {
- replace all **full nodes** by new P node
- insert all greater vertices adjacent to $v+1$ as sons of the new node

vertex addition - example





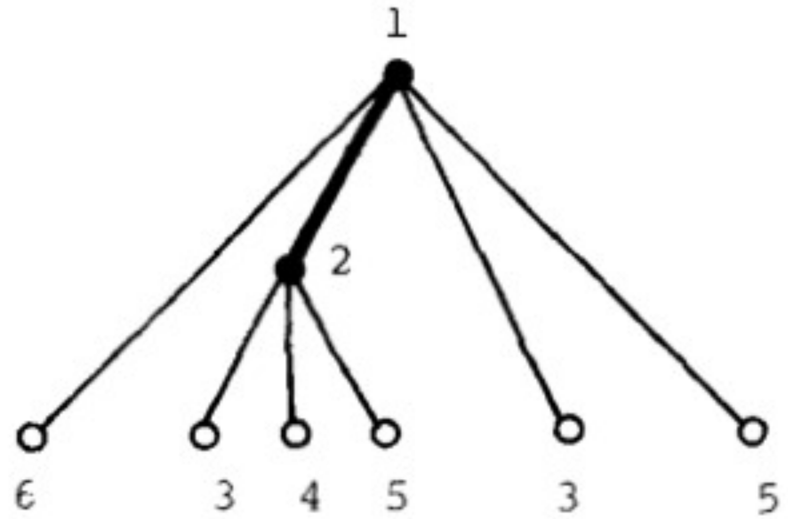
vertex addition - example

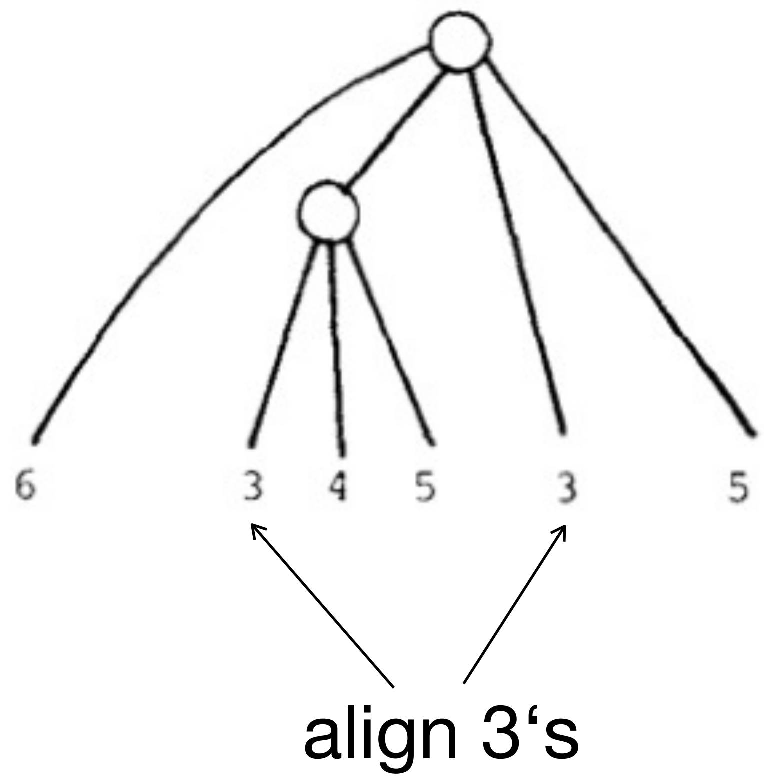
- **assign st-numbers** to all vertices ✔
- construct **PQ tree** corresponding to G_1' ✔
- for all other vertices $v = 2, ..., n$:

phase I {
- apply **template matchings** to relevant subtree to align all vertices $v+1$ to consecutive positions

phase II {
- replace all **full nodes** by new P node
- insert all greater vertices adjacent to $v+1$ as sons of the new node

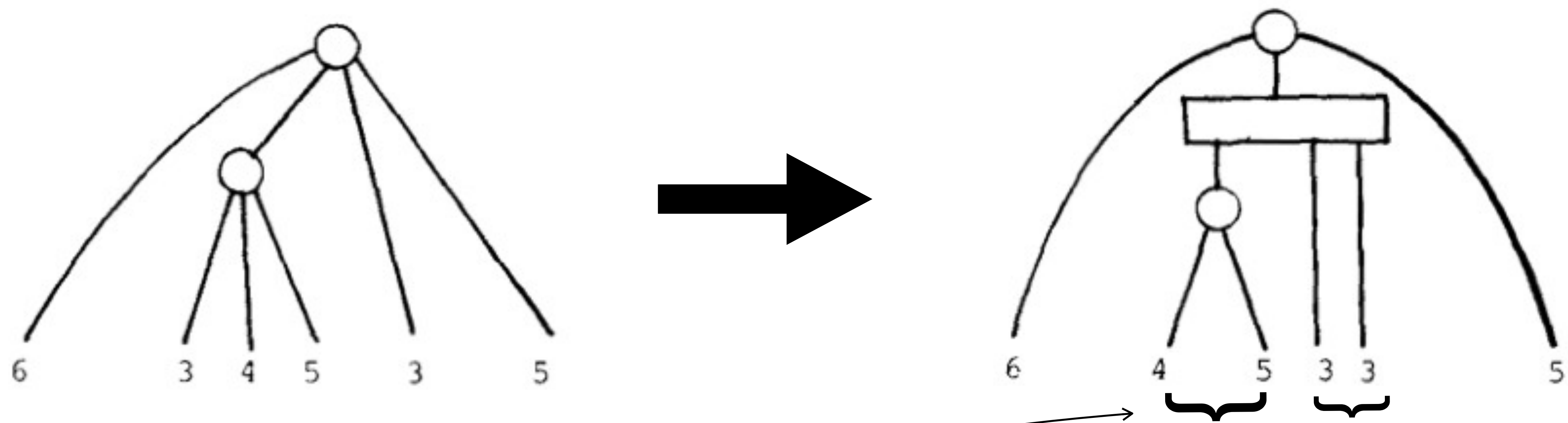
vertex addition - example





Why? → Make vertex appear „in the same place“ for all adjacent vertices

vertex addition - example

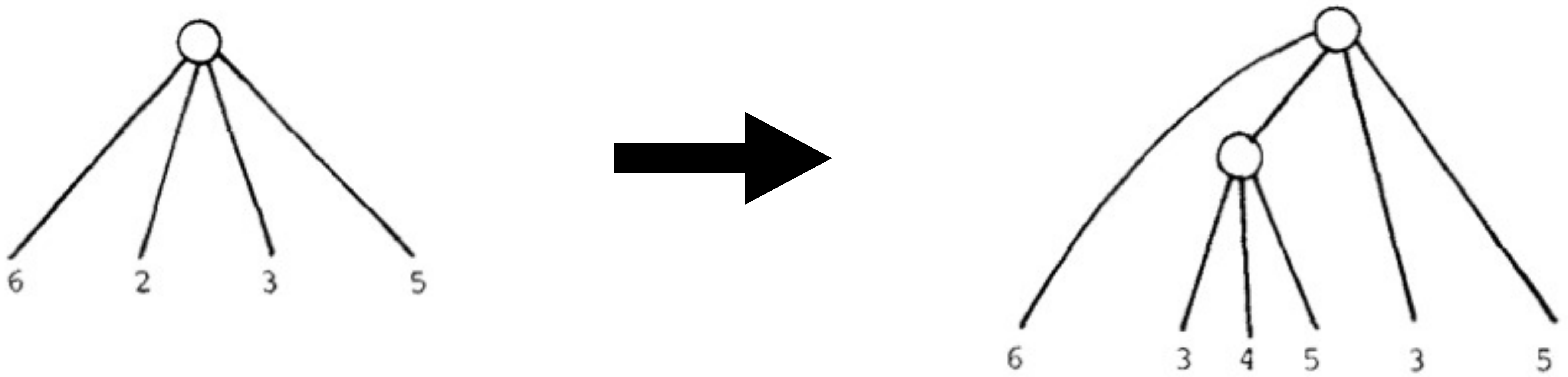


will stay together (but can be reversed)

can still be freely moved

and so on...

vertex addition - example



vertex addition - example

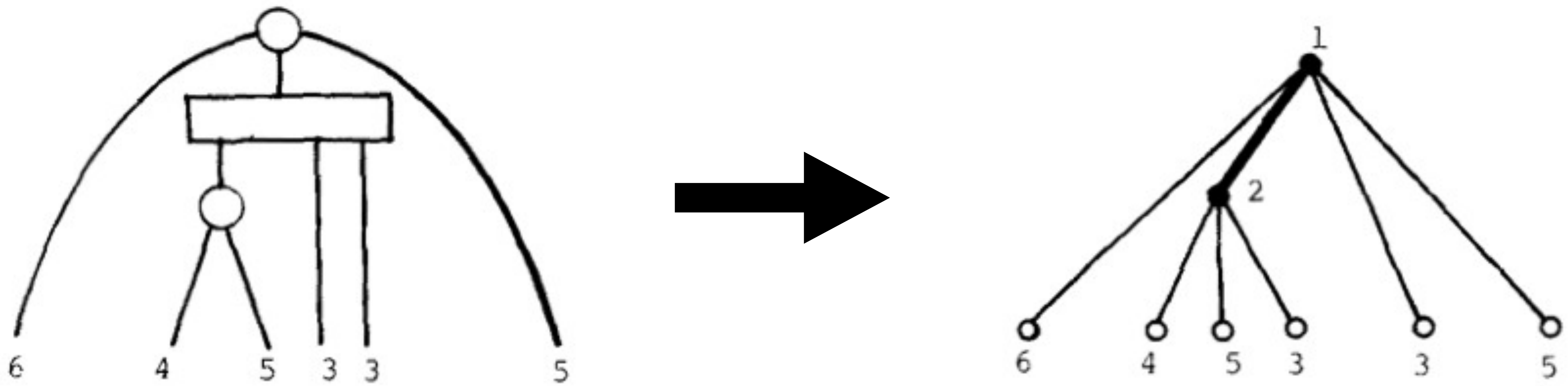
- only operating on **PQ tree**
- no record of **adjacency list** stored/ updated
- leads to naive algorithm

naive embedding algorithm

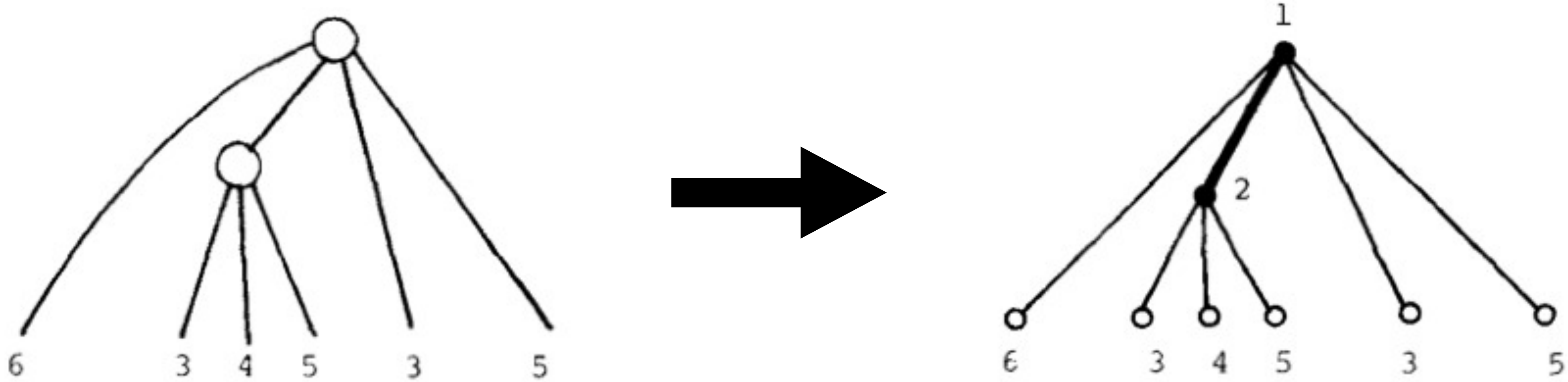
- modified **vertex addition algorithm**
- when applying **template matching**: reflect modification of PQ tree in **adjacency lists** of graph

„write down" the corresponding bush form as in example

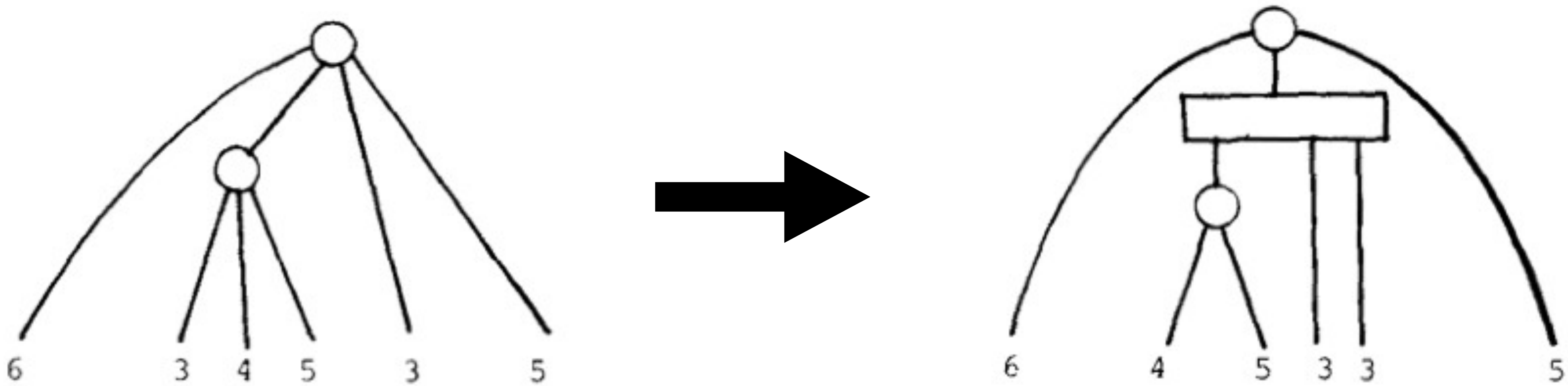
PQ tree ≠ bush form



nearly look the same, can be expressed by other PQ trees



naive algorithm - example



$$Adj(1) = 6, 2, 3, 5$$
$$Adj(2) = 3, 4, 5$$

$$Adj(1) = 6, 2, 3, 5$$
$$Adj(2) = 4, 5, 3$$

counter-clockwise appearance!

naive algorithm - complexity

- for every **step** $O(n)$
  - **reduction** $O(n)$ (Booth/Lueker)
  - **vertex addition** $O(m) = O(n)$ ← because $m \leq n$
- for every **re-write** of adjacency list $O(n)$
- total $O(n^2)$

heart of this talk

- **EMBED** (Nishizeki/Chiba)
- two-phased
  - generates an **embedding** (similar to naive algorithm) of **upward graph**
  - constructs **entire embedding** out of upward embedding

phase II

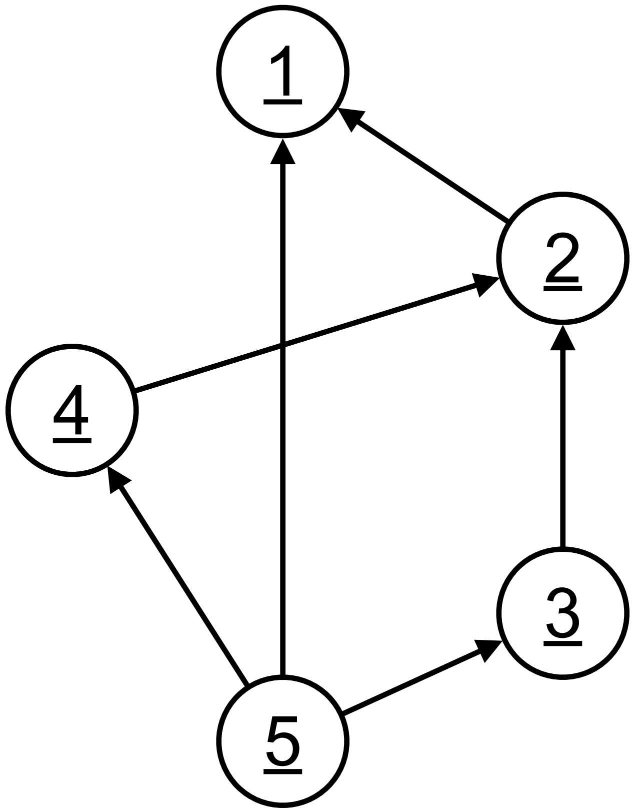
- given **upward embedding**
- use **adjacency lists** for DFS (yields $O(n)$)

phase II - DFS

- mark all vertices „new“
- begin on t (**largest** st-number)
- for each neighbor y
    - insert t in the beginning of $Adj(y)$
    - if y is „new“ proceed with it

here!

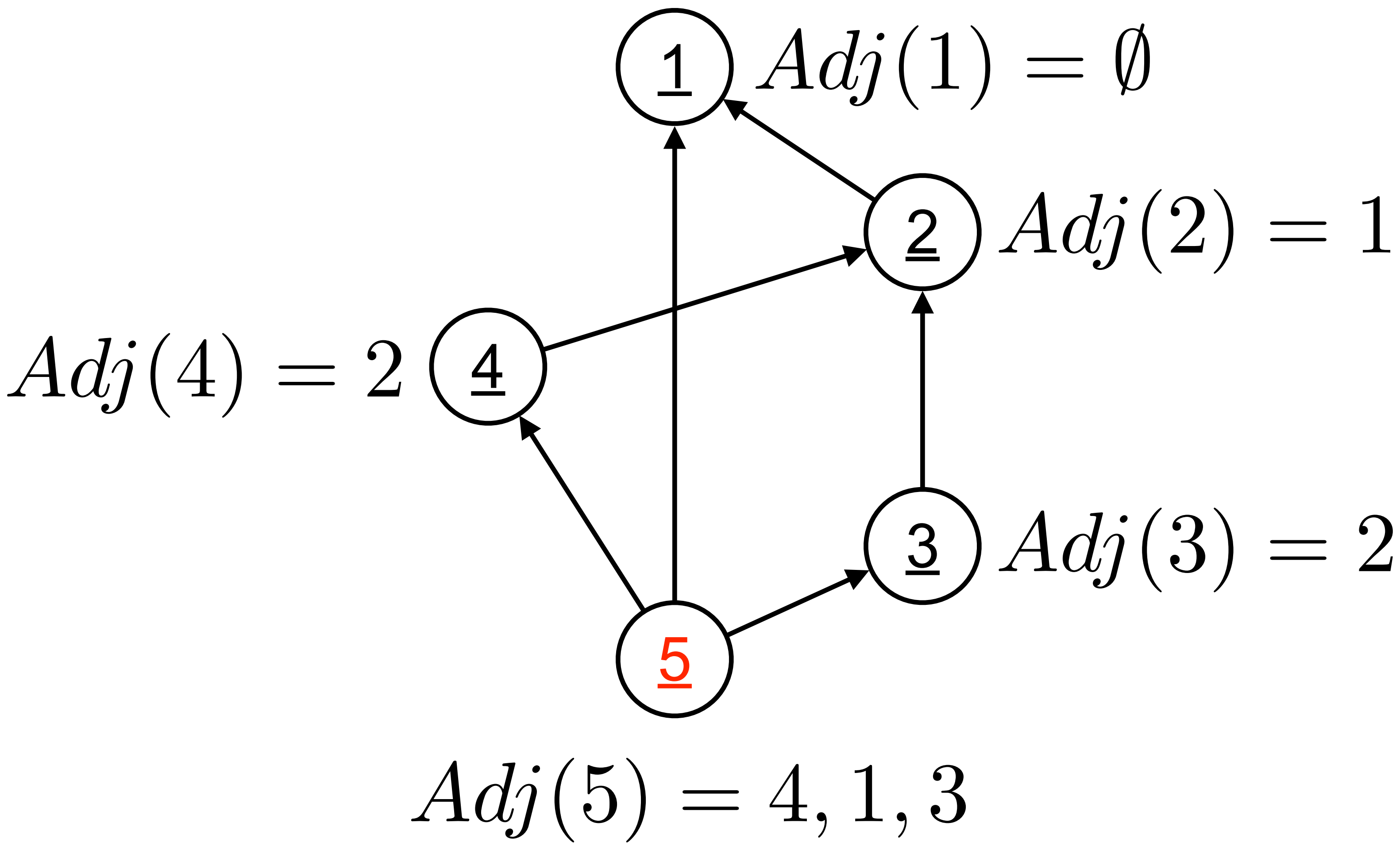
phase II - example



phase II - example

- mark all vertices „new“ ✔
- begin on t (**largest** st-number)
- for each neighbor y
  - insert t in the beginning of $Adj(y)$
  - if y is „new“ proceed with it

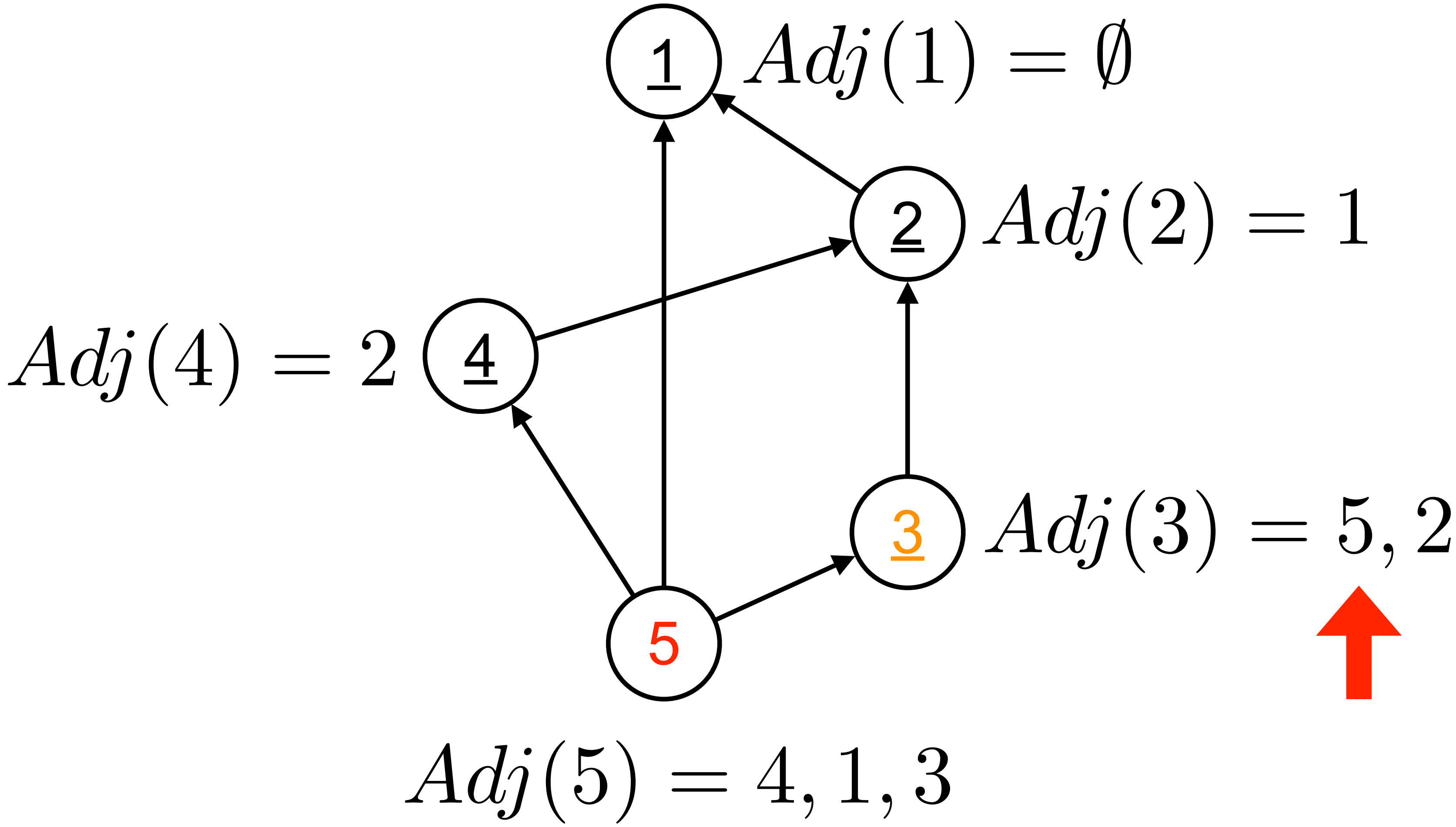
phase II - example



phase II - example

- mark all vertices „new“ ✔
- begin on t (**largest** st-number) ✔
- for each neighbor y
  - insert t in the beginning of $Adj(y)$
  - if y is „new“ proceed with it

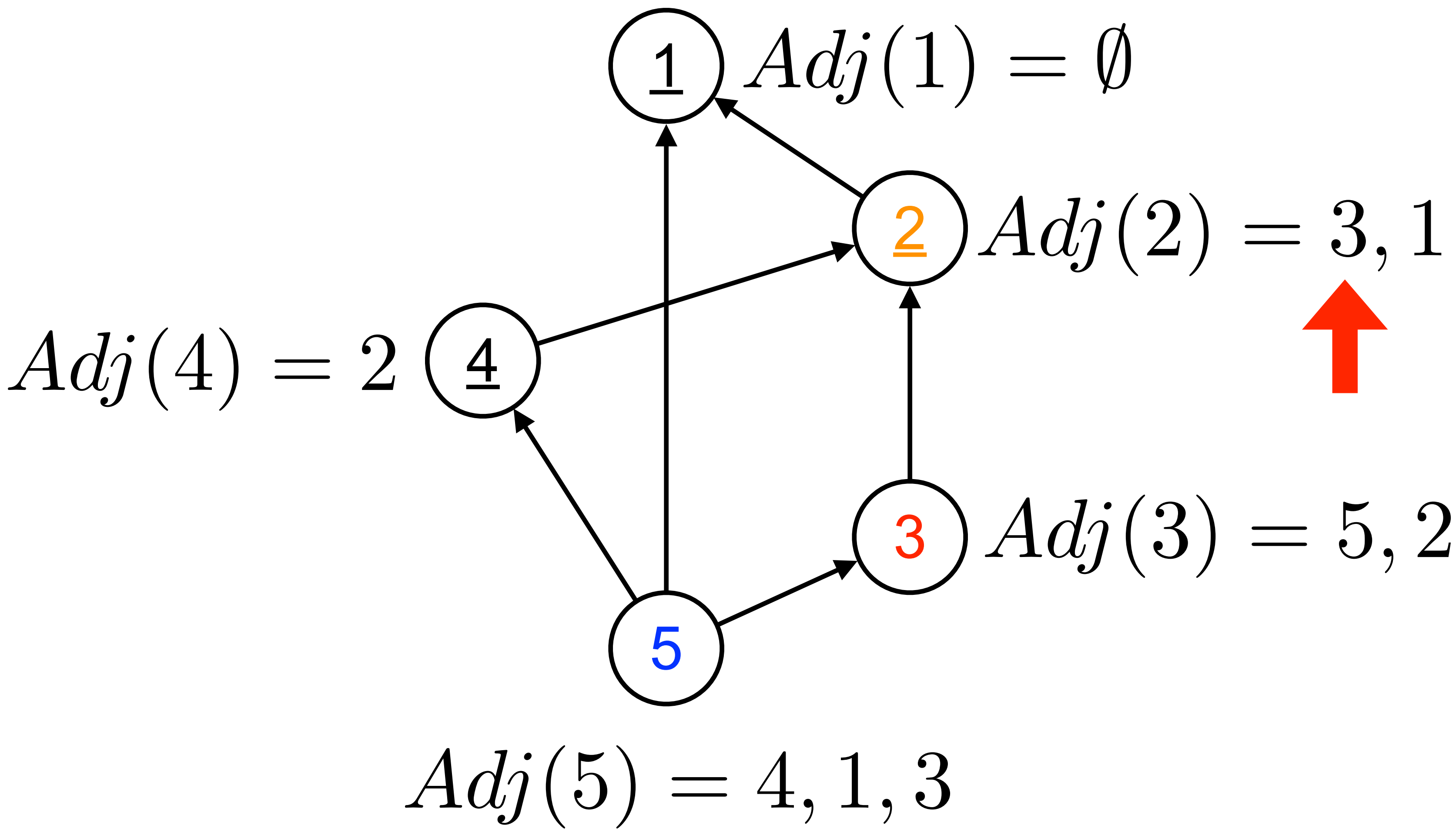
phase II - example



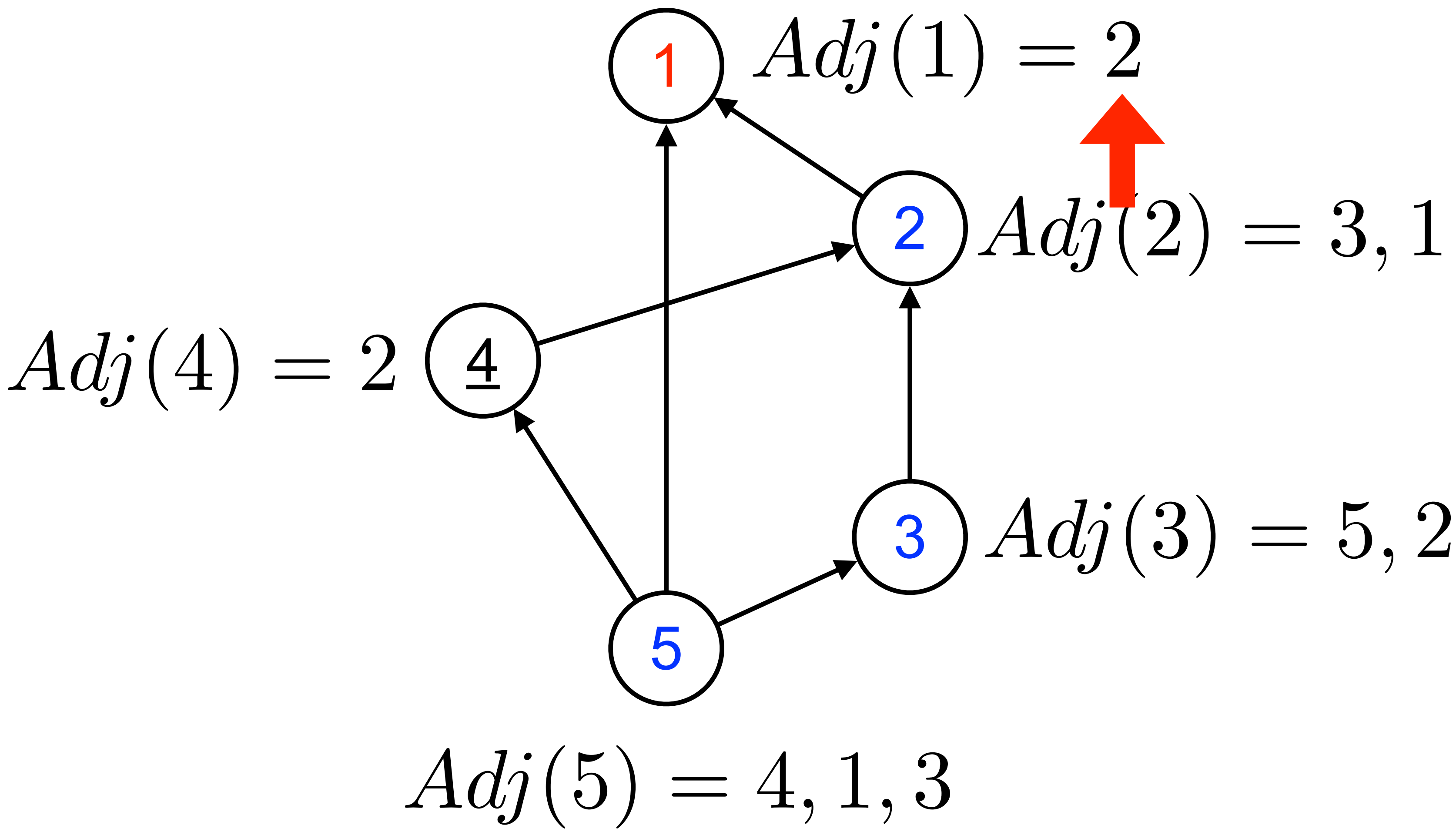
phase II - example

- mark all vertices „new“ ✔
- begin on t (**largest** st-number) ✔
- for each neighbor y
  - insert t in the beginning of $Adj(y)$ ✔
  - if y is „new“ proceed with it ✔

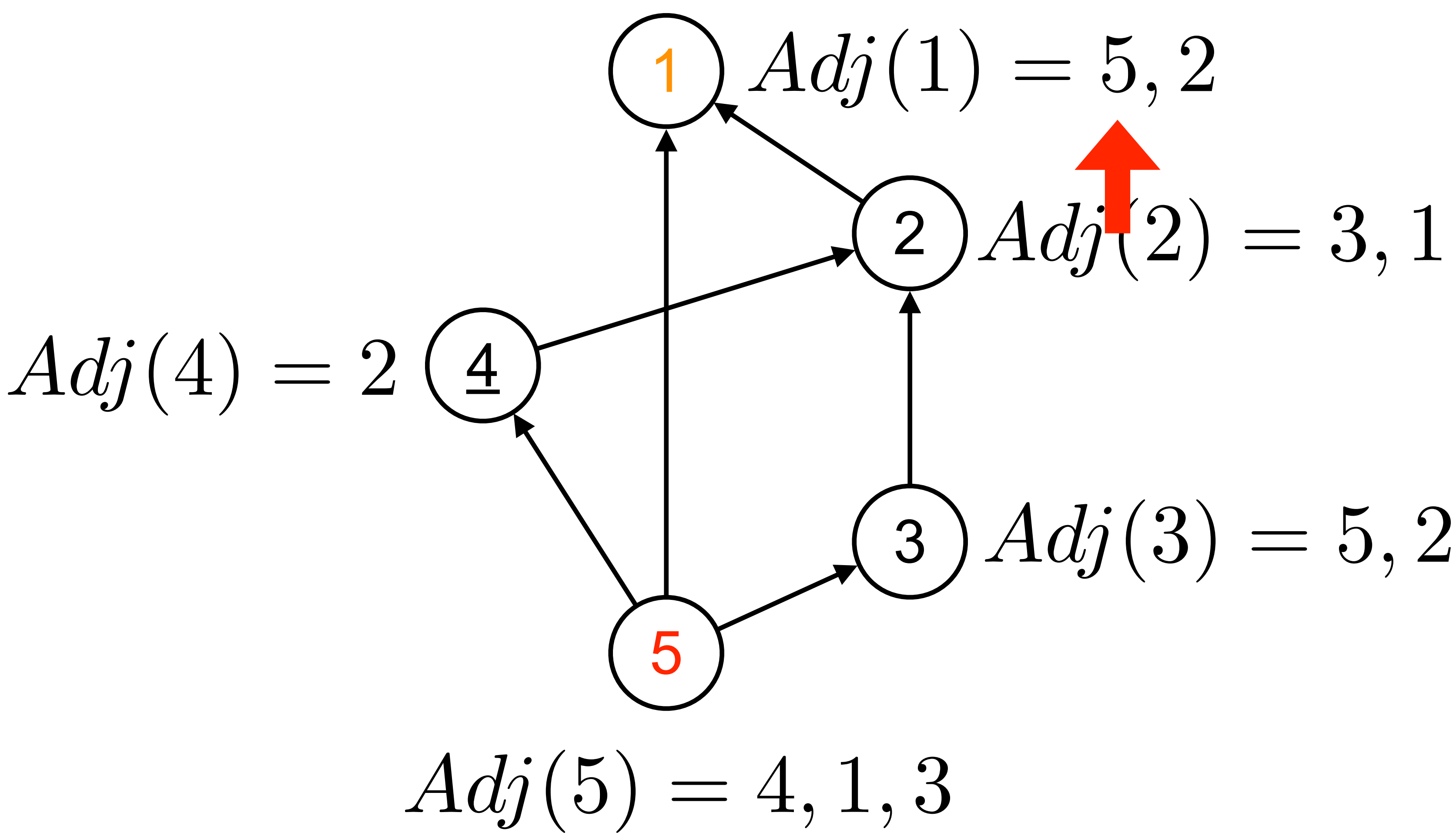
phase II - example



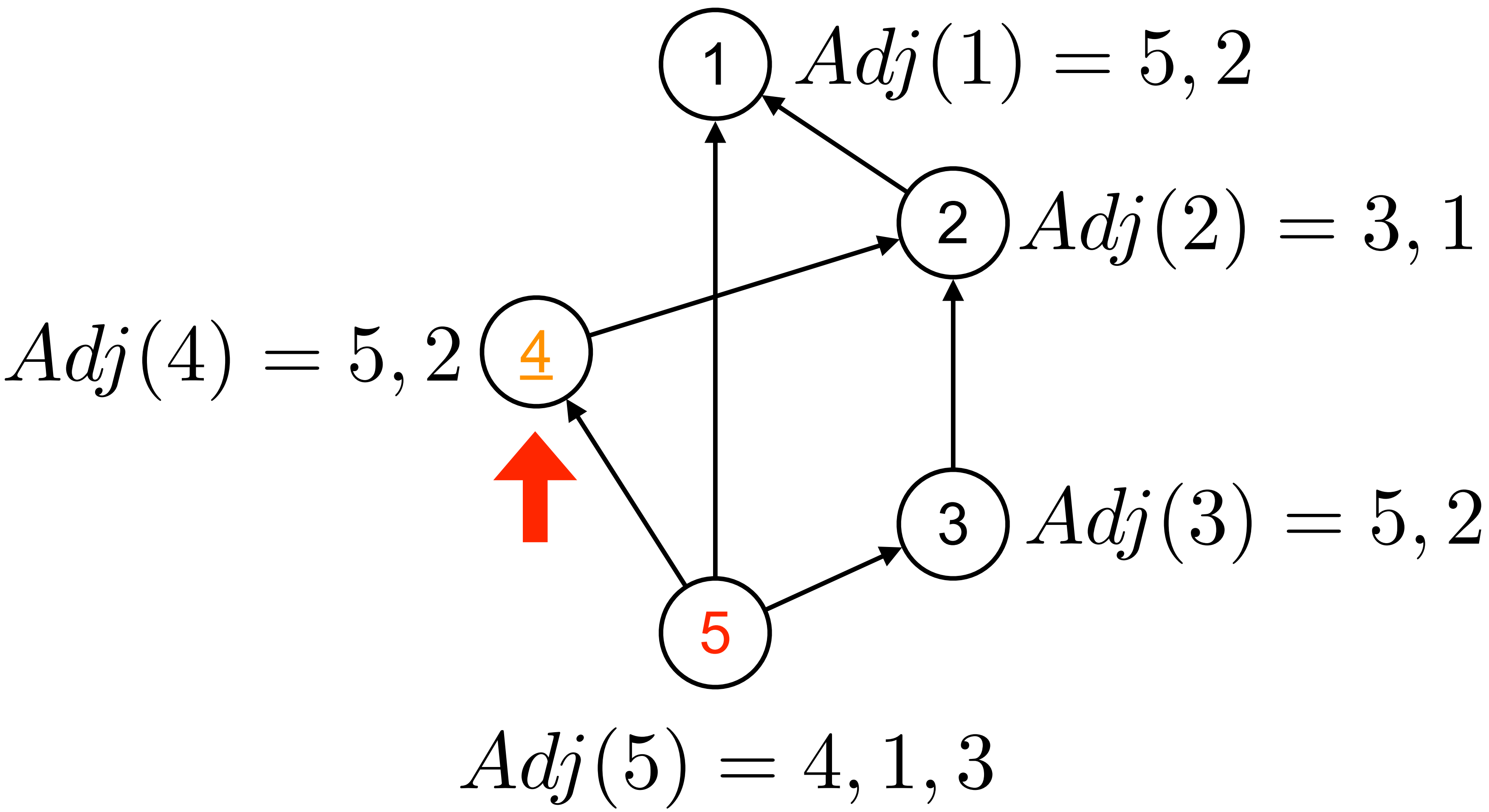
phase II - example



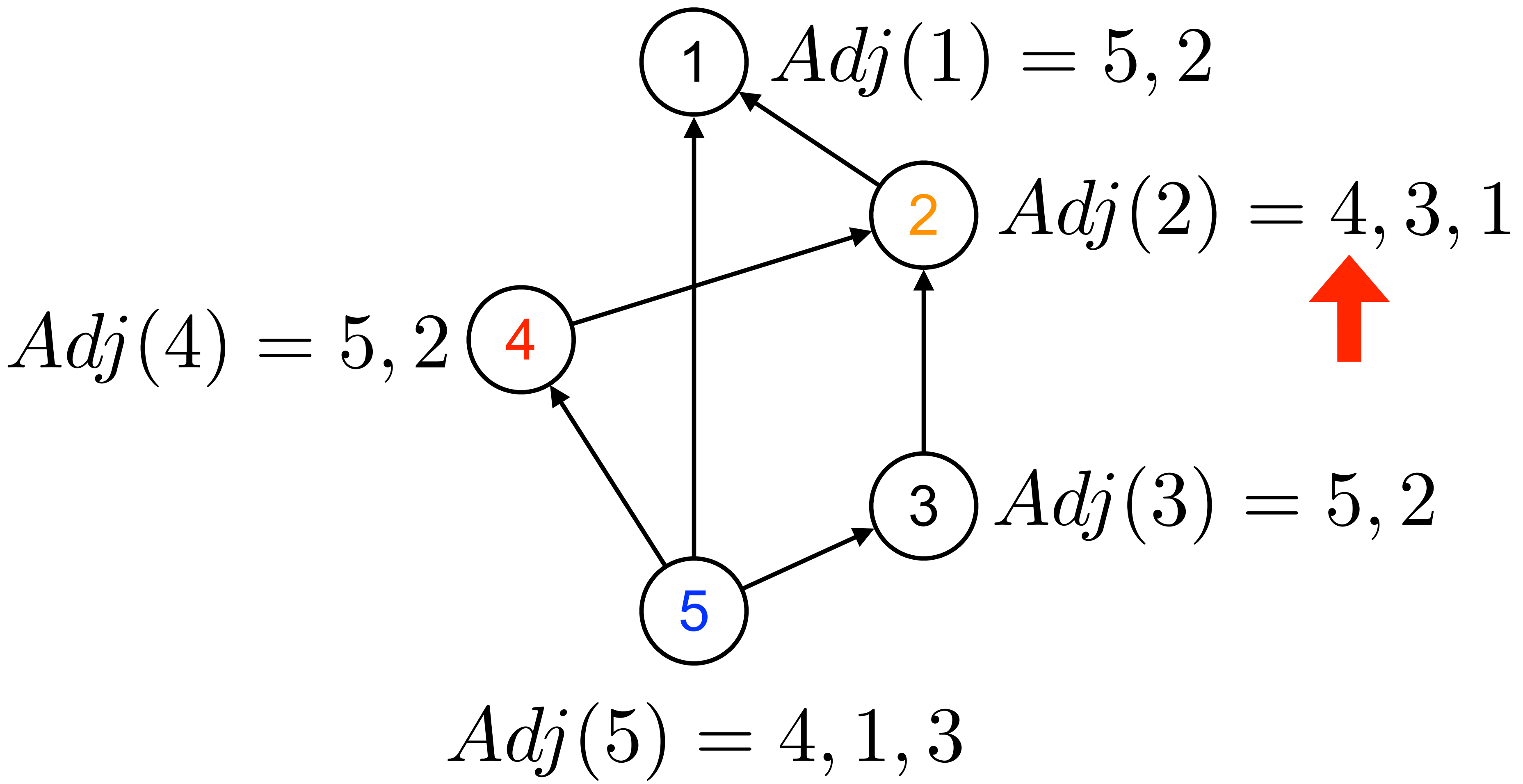
phase II - example



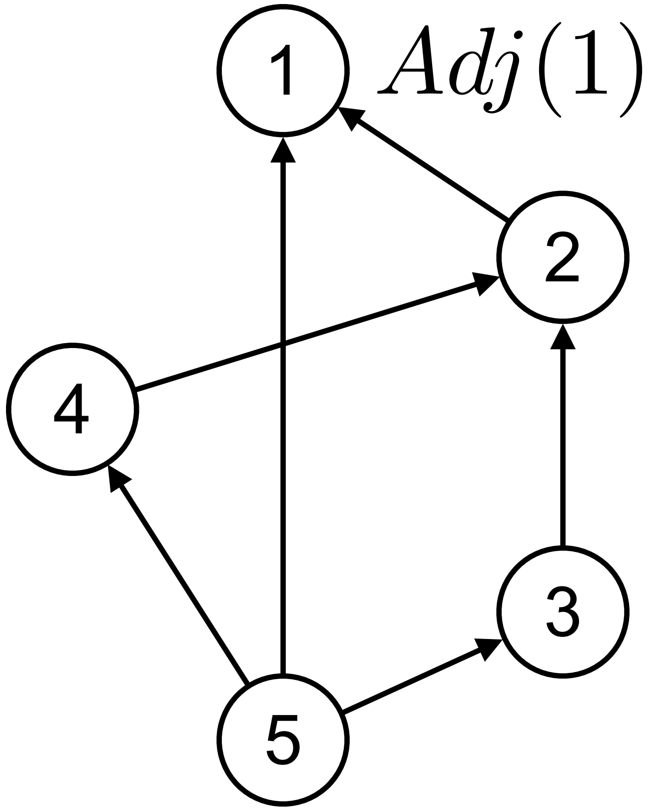
phase II - example



phase II - example



phase II - example



$Adj(1) = 5, 2$

$Adj(2) = 4, 3, 1$

$Adj(4) = 5, 2$

$Adj(3) = 5, 2$

$Adj(5) = 4, 1, 3$

intentional error: can somewhat spot it?
why did it happen?

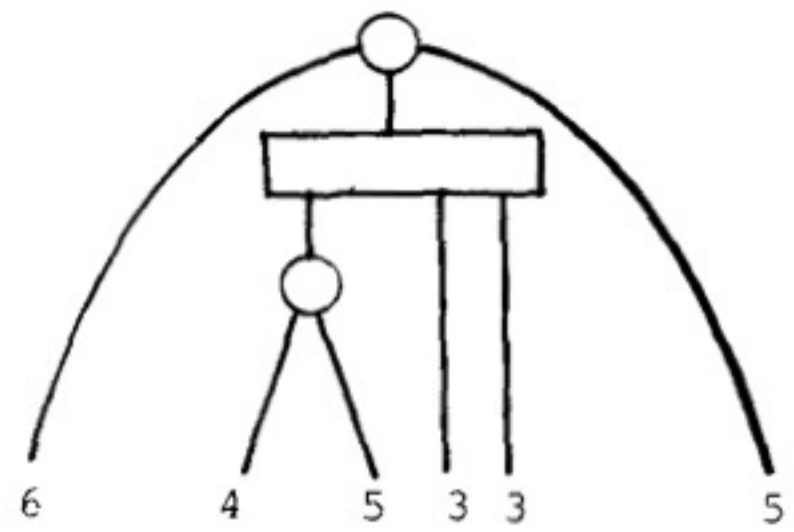
UPWARD-EMBED

- last thing you learn today
- core concept of EMBED
- uses **direction indicators** to determine direction of adjacency list
- cleverly inserts and removes indicators to yield $O(n)$

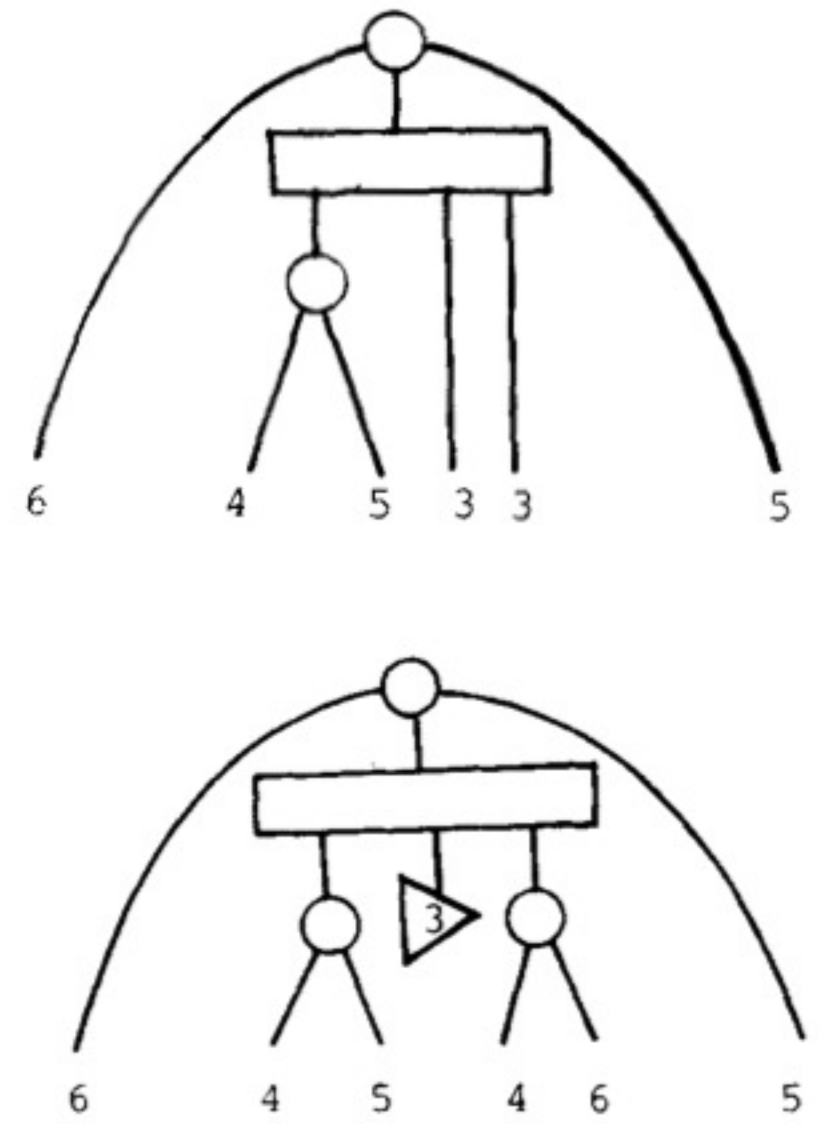
UPWARD-EMBED

- nearly the same as in PLANAR
- but now: use **direction indicators**
- correct **adjacency lists** in the end
- since **many errors** in paper: only an example to get the idea
- you can compose your own algorithm

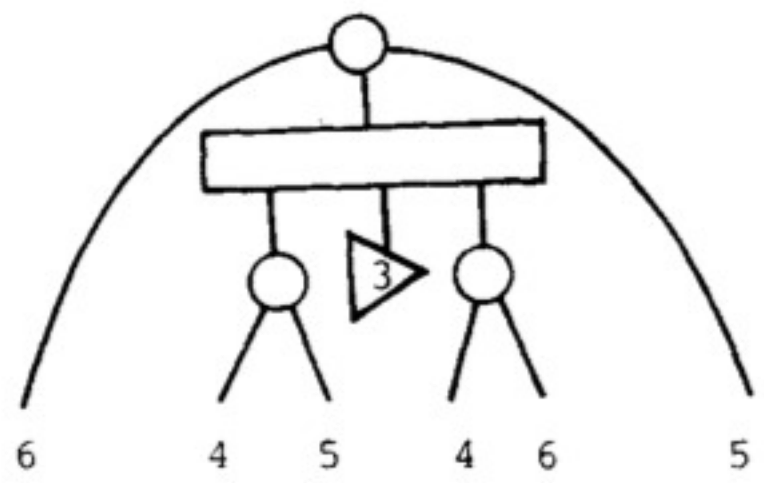
UPWARD-EMBED - example

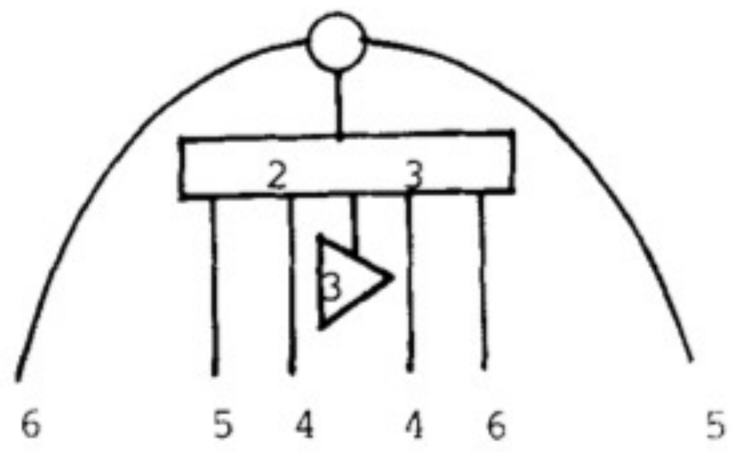


UPWARD-EMBED - example



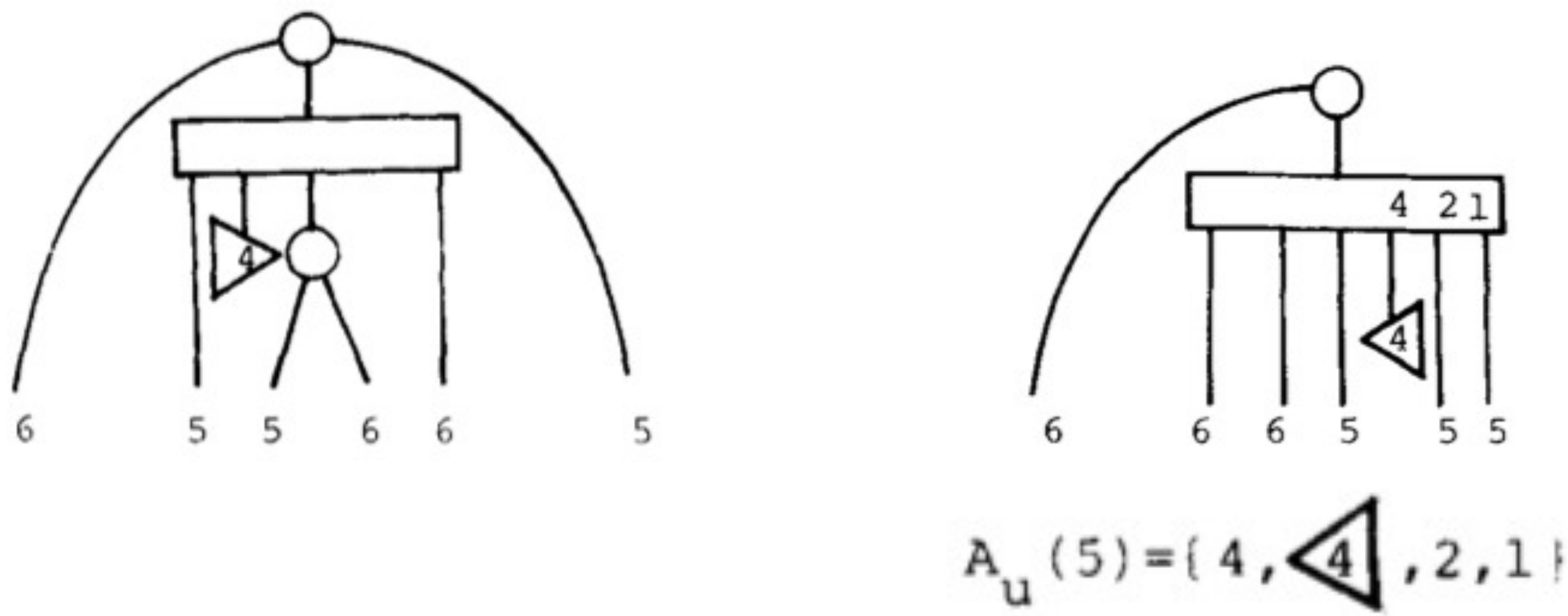
UPWARD-EMBED - example





UPWARD-EMBED - example

UPWARD-EMBED - example



UPWARD-EMBED - example

$A_u(5) = \{4, 4, 2, 1\}$

$A_u(4) = \{3, 2\}$

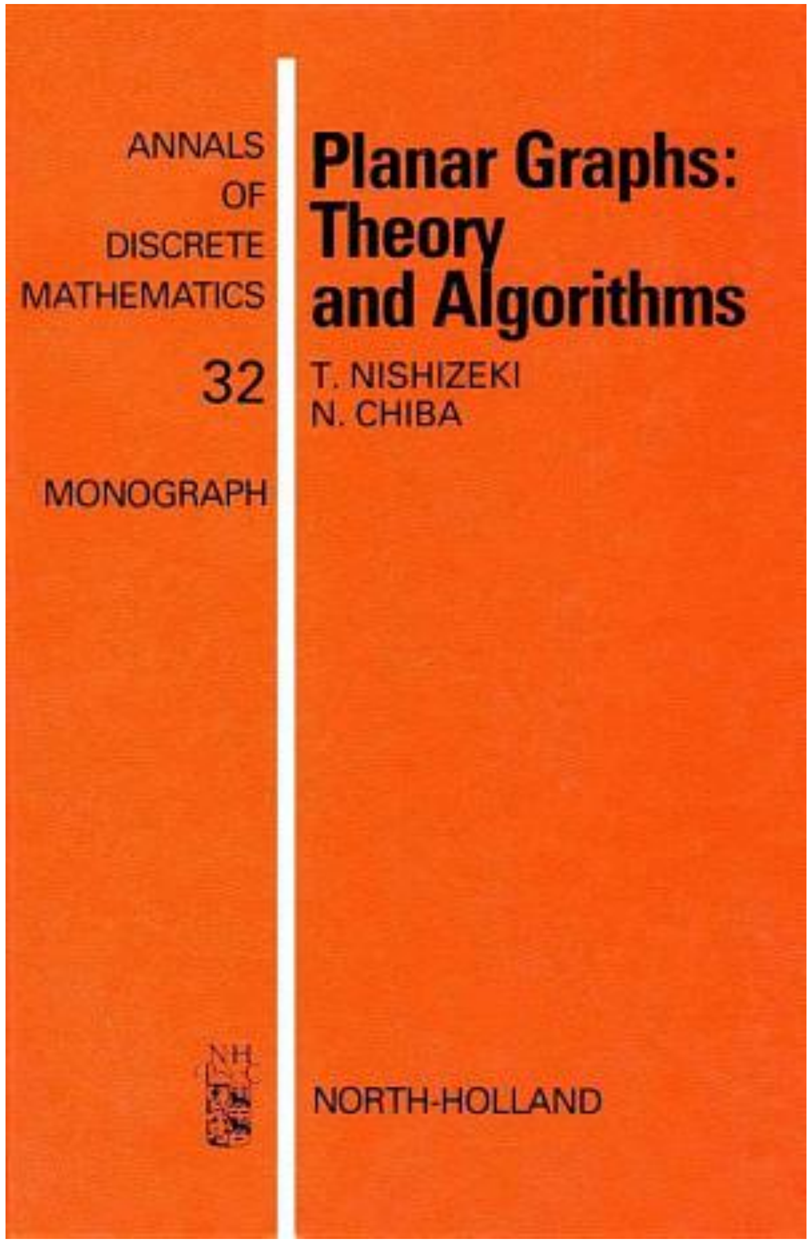
$A_u(4) = \{2, 3, 3\}$

UPWARD-EMBED

- PLANAR is linear time
- #edges is linear in #vertices (**planar graph as input**)
- processing of direction indicators is linear
- whole algorithm is linear (profit!)

good literature

- Nishizeki/Chiba



graphics source

- Despicable me 2 minions by Design Bolts
- tent icon by icons8
- example graphs from Nishizeki/Chiba

literature

- http://www.csd.uoc.gr/~hy583/reviewed_notes/st-orientations.pdf
- http://www.hausarbeiten.de/faecher/vorschau/213452.html (at least german, but also bugged, since only copy of initial paper by N/C)

thank you!

here is a photo of my cat