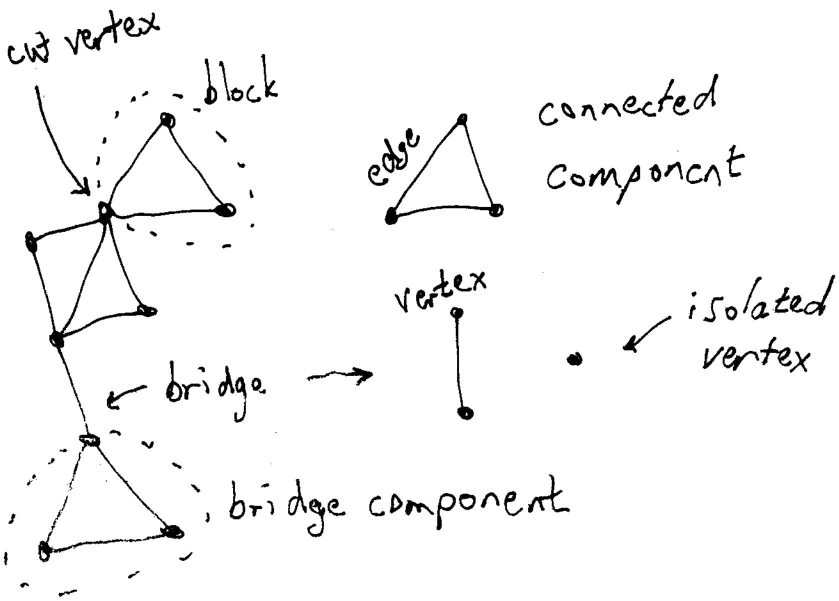
Graphs, Search, Components



Undirected Graph

Representations:

Adjacency Matrix

	a	b	c	d
a	0	1	0	1
b	1	0	1	0
c	0	1	0	0
d	1	0	0	0

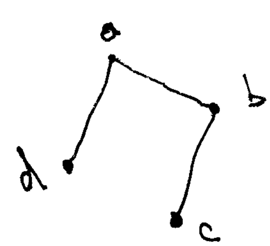
Adjacency Lists

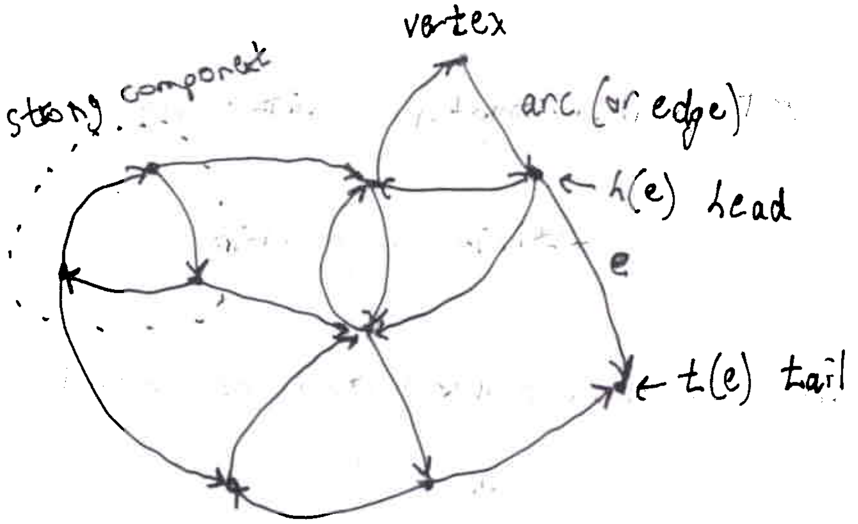
a: b, d

b: a, c

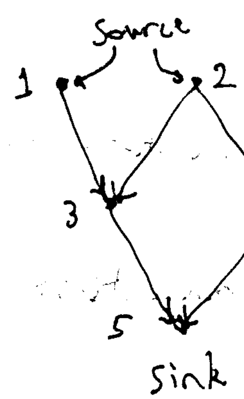
c: b

d: a



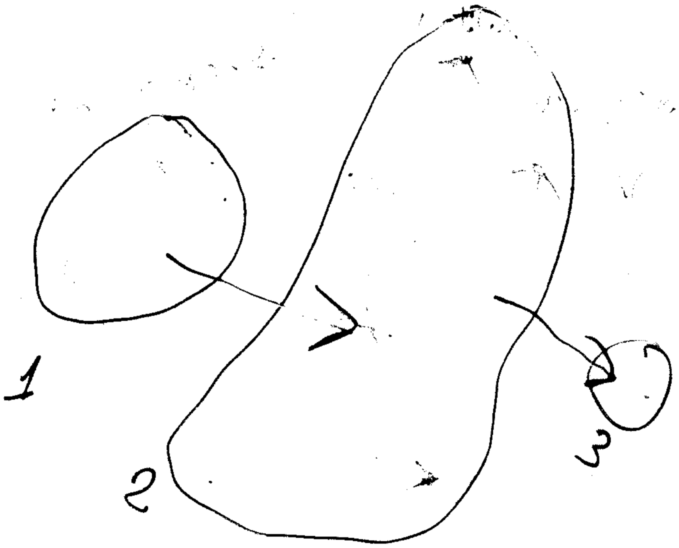


Di(rected) Graph



acyclic digraph

topological order: $i \to j \Rightarrow n(i) < n(j)$


1
2
3

Search

Explore vertices systematically by traversing edges

Mark vertices when visited

Traverse an untraversed edge from a visited vertex

or else

Start a new search from an unvisited vertex

Breadth-first search: queue of visited vertices

Depth-first search: stack of visited vertices

dfs(v): { previsit(v); scan(v); postvisit(v) }

scan(v): for $e \in$ arcs out(v) $\rightarrow$ traverse(e)

traverse(e): { advance(e);
  if not previsited(t(e)) $\rightarrow$ dfs(t(e));
  retreat(e) }

explore(G): for $v \in G \rightarrow$ if not previsited(v) $\rightarrow$ dfs(v)

Strong Components by DFS (Gabow)

Maintain stack of tentative components

Add each new vertex to stack as a singleton component

When advancing along an edge, if it leads from a component to a lower component on stack, combine all components down to the lower component

When postvisiting the last vertex in a component, output the component

Needs disjoint set union to maintain components:

$$O((m+n)\alpha(n))$$

Components output in reverse topological order

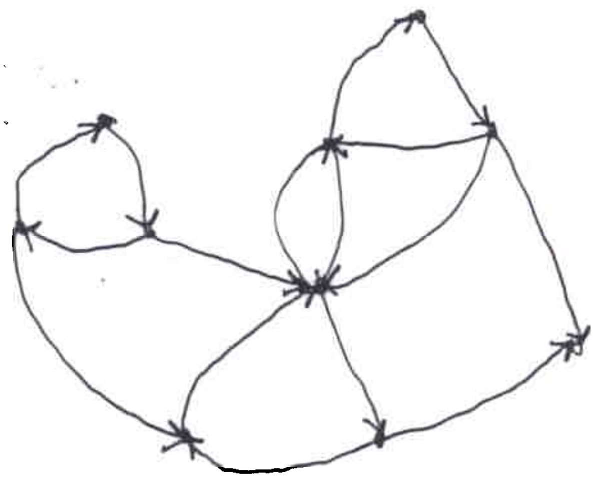
Linear-Time Version

Maintain stack of vertices not in permanent
components in preorder

Observe: tentative components are intervals on this stack

Maintain separate stack of (indices of) bottom vertices
in tentative components

vertex number : 0 if not previsited
stack position if positive
− component number if negative



Blocks

Number vertices in pre(visit) order

$low(v) = \min \{pre(v)\} \cup \{pre(w) \; \exists (x,w) \text{ with } x \text{ a descendant of } v\}$

v a cut vertex iff start vertex with degree ≥ 2

or $\exists$ child w of v with $pre(v) \leq low(w)$

Algorithm

Initialize $low(v) = pre(v)$ $\forall v$, stack empty

advance (v,w): add (v,w) to stack

retreat (v,w): if (v,w) not a tree edge $\rightarrow$
$low(v) = \min\{low(v), pre(w)\}$
else$\{$ $low(v) = \min\{low(v), low(w)\}$;
if $low(w) \geq pre(v)$, pop edges
down to and including (v,w)
from stack to form a block$\}$