

# Decompositions of Graphs

Hengfeng Wei

hfwei@nju.edu.cn

May 20, 2017 – May 24, 2017



# Decompositions of Graphs

1 DFS and BFS

2 Cycles

3 DAG

4 SCC

5 Biconnectivity

# Turing Award



John Hopcroft



Robert Tarjan

*“For fundamental achievements in the design and analysis of algorithms and data structures.”*

*— Turing Award, 1986*

# Depth-first search

SIAM J. COMPUT.  
Vol. 1, No. 2, June 1972

## DEPTH-FIRST SEARCH AND LINEAR GRAPH ALGORITHMS\*

ROBERT TARJAN†

**Abstract.** The value of depth-first search or “backtracking” as a technique for solving problems is illustrated by two examples. An improved version of an algorithm for finding the strongly connected components of a directed graph and an algorithm for finding the biconnected components of an undirect graph are presented. The space and time requirements of both algorithms are bounded by  $k_1 V + k_2 E + k_3$  for some constants  $k_1, k_2$ , and  $k_3$ , where  $V$  is the number of vertices and  $E$  is the number of edges of the graph being examined.

**Key words.** Algorithm, backtracking, biconnectivity, connectivity, depth-first, graph, search, spanning tree, strong-connectivity.

## Reference

- ▶ “Depth-First Search And Linear Graph Algorithms” by Robert Tarjan.

# Depth-first search

SIAM J. COMPUT.  
Vol. 1, No. 2, June 1972

## DEPTH-FIRST SEARCH AND LINEAR GRAPH ALGORITHMS\*

ROBERT TARJAN†

**Abstract.** The value of depth-first search or “backtracking” as a technique for solving problems is illustrated by two examples. An improved version of an algorithm for finding the strongly connected components of a directed graph and an algorithm for finding the biconnected components of an undirect graph are presented. The space and time requirements of both algorithms are bounded by  $k_1 V + k_2 E + k_3$  for some constants  $k_1, k_2$ , and  $k_3$ , where  $V$  is the number of vertices and  $E$  is the number of edges of the graph being examined.

**Key words.** Algorithm, backtracking, biconnectivity, connectivity, depth-first, graph, search, spanning tree, strong-connectivity.

*“We **have seen** how the depth-first search method may be used in the construction of very efficient graph algorithms. . .*

*Depth-first search **is** a powerful technique with many applications.”*

## Reference

- ▶ “Depth-First Search And Linear Graph Algorithms” by Robert Tarjan.

# The POWER of DFS

Graph decomposition vs. Graph traversal

Structures!

# The POWER of DFS

Graph decomposition vs. Graph traversal

Structures!

1. states of vertices
2. types of edges
3. lifetime of vertices (DFS)
  - ▶  $v : d[v], f[v]$
  - ▶  $f[v]$ : DAG, SCC
  - ▶  $d[v]$ : biconnectivity

# Types of edges

## Definition (Classifying edges)

Given a DFS/BFS traversal  $\Rightarrow$  DFS/BFS tree:

Tree edge:  $\rightarrow$  child

Back edge:  $\rightarrow$  ancestor

Forward edge:  $\rightarrow$  *nonchild* descendant

Cross edge:  $\rightarrow$  neither ancestor nor descendant



# Types of edges

## Definition (Classifying edges)

Given a DFS/BFS traversal  $\Rightarrow$  DFS/BFS tree:

Tree edge:  $\rightarrow$  child

Back edge:  $\rightarrow$  ancestor

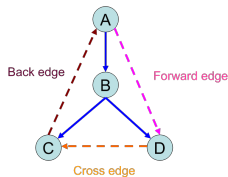
Forward edge:  $\rightarrow$  *nonchild* descendant

Cross edge:  $\rightarrow$  neither ancestor nor descendant

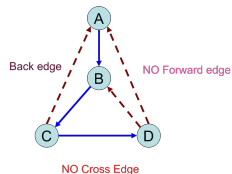
## Remarks

- ▶ applicable to both DFS and BFS
- ▶ w.r.t. DFS/BFS trees

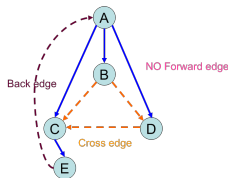
# Types of edges (Problem 5.18)



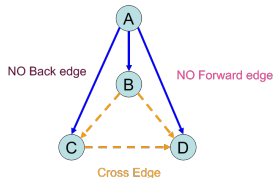
(a) DFS on directed graph.



(b) DFS on undirected graph.



(c) BFS on directed graph.



(d) BFS on undirected graph.

# Types of edges

## DFS tree and BFS tree coincide (Additional Problem)

- ▶ undirected connected graph  $G = (V, E), v \in V$
- ▶ DFS tree  $T$  from  $v \equiv$  BFS tree  $T'$  from  $v$
- ▶ prove:  $G = T$

# Types of edges

## DFS tree and BFS tree coincide (Additional Problem)

- ▶ undirected connected graph  $G = (V, E), v \in V$
- ▶ DFS tree  $T$  from  $v \equiv$  BFS tree  $T'$  from  $v$
- ▶ prove:  $G = T$

$G_{\text{DFS}}$ : tree + back vs.  $G_{\text{BFS}}$ : tree + cross

# Types of edges

## DFS tree and BFS tree coincide (Additional Problem)

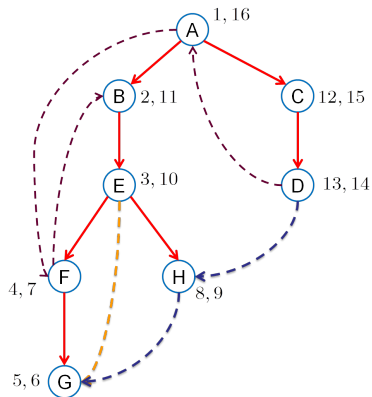
- ▶ undirected connected graph  $G = (V, E), v \in V$
- ▶ DFS tree  $T$  from  $v \equiv$  BFS tree  $T'$  from  $v$
- ▶ prove:  $G = T$

$G_{\text{DFS}}$ : tree + back vs.  $G_{\text{BFS}}$ : tree + cross

## Question

- ▶ DFS&BFS from different  $v'$ s?
- ▶ What if  $G$  is a digraph?

# Lifetime of vertices in DFS



# Lifetime of vertices in DFS

## Theorem (Disjoint or contained)

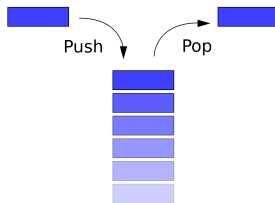
$$\begin{aligned} \forall u, v : \\ [u]_u \cap [v]_v = \emptyset \\ \vee \\ ([u]_u \subsetneq [v]_v \vee [v]_v \subsetneq [u]_u) \end{aligned}$$

# Lifetime of vertices in DFS

## Theorem (Disjoint or contained)

$$\forall u, v : \\ [u]_u \cap [v]_v = \emptyset \\ \vee \\ ([u]_u \subsetneq [v]_v \vee [v]_v \subsetneq [u]_u)$$

Proof.





# Ancestor/descendant relation

## Preprocessing for ancestor/descendant relation (Problem 5.23)

- ▶ binary tree  $T = (V, E)$
- ▶  $r \in V$

$$v : d[v], f[v]$$

# Ancestor/descendant relation

## Preprocessing for ancestor/descendant relation (Problem 5.23)

- ▶ binary tree  $T = (V, E)$
- ▶  $r \in V$

$$v : d[v], f[v]$$

## Question

$\forall v$ : how many descendants?

# Ancestor/descendant relation

## Preprocessing for ancestor/descendant relation (Problem 5.23)

- ▶ binary tree  $T = (V, E)$
- ▶  $r \in V$

$$v : d[v], f[v]$$

### Question

$\forall v$ : how many descendants?

### Remark

General (rooted) tree?

# Edge types and lifetime of vertices in DFS

## Edge types and lifetime of vertices in DFS (Problem 5.2)

$\forall u \rightarrow v$ :

- ▶ tree/forward edge:  $[u \ [v \ ]_v ]_u$
- ▶ back edge:  $[v \ [u \ ]_u ]_v$
- ▶ cross edge:  $[v \ ]_v \ [u \ ]_u$

# Edge types and lifetime of vertices in DFS

## Edge types and lifetime of vertices in DFS (Problem 5.2)

$\forall u \rightarrow v$ :

- ▶ tree/forward edge:  $[u \ [v \ ]_v ]_u$
- ▶ back edge:  $[v \ [u \ ]_u ]_v$
- ▶ cross edge:  $[v \ ]_v \ [u \ ]_u$

### Remark

- ▶  $f[v] < d[u]$ : cross edge
- ▶  $f[u] < f[v]$ : back edge

# Edge types and lifetime of vertices in DFS

## Edge types and lifetime of vertices in DFS (Problem 5.2)

$\forall u \rightarrow v$ :

- ▶ tree/forward edge:  $[u \ [v \ ]_v ]_u$
- ▶ back edge:  $[v \ [u \ ]_u ]_v$
- ▶ cross edge:  $[v \ ]_v \ [u \ ]_u$

### Remark

- ▶  $f[v] < d[u]$ : cross edge
- ▶  $f[u] < f[v]$ : back edge

$$u \rightarrow v \iff f[v] < f[u]$$

# Height and diameter of tree

## Height and diameter of tree (Problem 5.21)

Binary tree  $T = (V, E)$  with  $|V| = n$ :

- ▶ height ( $O(n)$ )
- ▶ diameter ( $O(n)$ )

# Height and diameter of tree

## Height and diameter of tree (Problem 5.21)

Binary tree  $T = (V, E)$  with  $|V| = n$ :

- ▶ height ( $O(n)$ )
- ▶ diameter ( $O(n)$ )

thought root or not?



# Height and diameter of tree

## Height and diameter of tree (Problem 5.21)

Binary tree  $T = (V, E)$  with  $|V| = n$ :

- ▶ height ( $O(n)$ )
- ▶ diameter ( $O(n)$ )

thought root or not?

## Question

Diameter of a tree *without* a designated root?

# Perfect subtree

## Perfect subtree (Problem 5.22)

- ▶ binary tree  $T = (V, E)$
- ▶ root  $r \in V$
- ▶ goal: find all perfect subtrees

# Counting shortest paths

## Counting shortest paths (Problem 5.26)

Counting # of shortest paths in (un)directed graphs using BFS.

# Counting shortest paths

## Counting shortest paths (Problem 5.26)

Counting # of shortest paths in (un)directed graphs using BFS.

Maybe in the next class...

# Decompositions of Graphs

1 DFS and BFS

2 Cycles

3 DAG

4 SCC

5 Biconnectivity

# Cycle detection

## Cycle detection (Problem 5.24–1)

	Digraph	Undirected graph
DFS		
BFS		

# Cycle detection

## Cycle detection (Problem 5.24–1)

	Digraph	Undirected graph
DFS	back edge $\iff$ cycle	
BFS		

# Cycle detection

## Cycle detection (Problem 5.24–1)

	Digraph	Undirected graph
DFS	back edge $\iff$ cycle	back edge $\iff$ cycle
BFS		



# Cycle detection

## Cycle detection (Problem 5.24–1)

	Digraph	Undirected graph
DFS	back edge $\iff$ cycle	back edge $\iff$ cycle
BFS		cross edge $\iff$ cycle

# Cycle detection

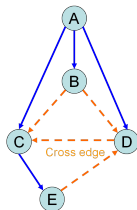
## Cycle detection (Problem 5.24–1)

	Digraph	Undirected graph
DFS	back edge $\iff$ cycle	back edge $\iff$ cycle
BFS	back edge $\implies$ cycle cycle $\not\Rightarrow$ back edge	cross edge $\iff$ cycle

# Cycle detection

## Cycle detection (Problem 5.24–1)

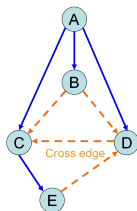
	Digraph	Undirected graph
DFS	back edge $\iff$ cycle	back edge $\iff$ cycle
BFS	back edge $\implies$ cycle cycle $\not\Rightarrow$ back edge	cross edge $\iff$ cycle



# Cycle detection

## Cycle detection (Problem 5.24–1)

	Digraph	Undirected graph
DFS	back edge $\iff$ cycle	back edge $\iff$ cycle
BFS	back edge $\implies$ cycle cycle $\not\Rightarrow$ back edge	cross edge $\iff$ cycle



### Remark

How to identify back edges?

# Evasiveness of acyclicity

## Evasiveness of acyclicity (Problem 5.24–2)

Evasiveness  $\triangleq$  check  $\binom{n}{2}$  edges (adjacency matrices)

# Evasiveness of acyclicity

## Evasiveness of acyclicity (Problem 5.24–2)

Evasiveness  $\triangleq$  check  $\binom{n}{2}$  edges (adjacency matrices)

Is acyclicity evasive?

# Evasiveness of acyclicity

## Evasiveness of acyclicity (Problem 5.24–2)

Evasiveness  $\triangleq$  check  $\binom{n}{2}$  edges (adjacency matrices)

Is acyclicity evasive?



# Evasiveness of acyclicity

## Evasiveness of acyclicity (Problem 5.24–2)

Evasiveness  $\triangleq$  check  $\binom{n}{2}$  edges (adjacency matrices)

Is acyclicity evasive?



Hint: Kruskal



# Evasiveness of connectivity

## Evasiveness of connectivity (Additional Problem)

Evasiveness  $\triangleq$  check  $\binom{n}{2}$  edges

Is connectivity evasive?

# Evasiveness of connectivity

## Evasiveness of connectivity (Additional Problem)

Evasiveness  $\triangleq$  check  $\binom{n}{2}$  edges

Is connectivity evasive?



Hint: Anti-Kruskal

# Edge deletion

## Edge deletion (Problem 5.20)

- ▶ connected, undirected graph  $G$
- ▶  $\exists e \in E : G \setminus e$  is connected?
- ▶  $O(|V|)$

# Edge deletion

## Edge deletion (Problem 5.20)

- ▶ connected, undirected graph  $G$
- ▶  $\exists e \in E : G \setminus e$  is connected?
- ▶  $O(|V|)$

$$\exists \text{ cycle} \iff \exists \text{ such } e$$

# Edge deletion

## Edge deletion (Problem 5.20)

- ▶ connected, undirected graph  $G$
- ▶  $\exists e \in E : G \setminus e$  is connected?
- ▶  $O(|V|)$

$$\exists \text{ cycle} \iff \exists \text{ such } e$$

$$O(m + n)$$

# Edge deletion

## Edge deletion (Problem 5.20)

- ▶ connected, undirected graph  $G$
- ▶  $\exists e \in E : G \setminus e$  is connected?
- ▶  $O(|V|)$

$$\exists \text{ cycle} \iff \exists \text{ such } e$$

$$O(m + n)$$

$$\text{tree: } |E| = |V| - 1 \implies \text{check } |E| \geq |V|$$

# Orientation of undirected graph

## Orientation of undirected graph (Problem 5.9)

- ▶ undirected (connected) graph  $G$
- ▶ edges oriented *s.t.*

$$\forall v, \text{in}[v] \geq 1$$

# Orientation of undirected graph

## Orientation of undirected graph (Problem 5.9)

- ▶ undirected (connected) graph  $G$
- ▶ edges oriented *s.t.*

$$\forall v, \text{in}[v] \geq 1$$

$$\text{orientation} \iff \exists \text{ cycle } C$$



# Orientation of undirected graph

## Orientation of undirected graph (Problem 5.9)

- ▶ undirected (connected) graph  $G$
- ▶ edges oriented *s.t.*

$$\forall v, \text{in}[v] \geq 1$$

orientation  $\iff \exists \text{ cycle } C$

BFS/DFS from  $v \in C$

# Shortest cycle of undirected graph

## Shortest cycle of undirected graph (Problem 5.8)

Shortest cycle of  $G$ :

- ▶ DFS on  $G$
- ▶  $\forall v : \text{level}[v]$
- ▶ back edge  $u \rightarrow v : \text{level}[u] - \text{level}[v] + 1$

# Shortest cycle of undirected graph

## Shortest cycle of undirected graph (Problem 5.8)

Shortest cycle of  $G$ :

- ▶ DFS on  $G$
- ▶  $\forall v : \text{level}[v]$
- ▶ back edge  $u \rightarrow v : \text{level}[u] - \text{level}[v] + 1$

## Question

What about digraphs?

# Decompositions of Graphs

1 DFS and BFS

2 Cycles

**3 DAG**

4 SCC

5 Biconnectivity

# DAG

no back edge  $\iff$  DAG

# DAG

no back edge  $\iff$  DAG  $\iff \exists$  topo. ordering

Toposort algorithm by Tarjan (probably), 1976

DFS on digraph,  $u \rightarrow v$ :

- ▶ ~~back edge~~:  $f[u] < f[v]$
- ▶ others:  $f[u] > f[v]$

## DAG

no back edge  $\iff$  DAG  $\iff \exists$  topo. ordering

Toposort algorithm by Tarjan (probably), 1976

DFS on digraph,  $u \rightarrow v$ :

- ▶ ~~back edge~~:  $f[u] < f[v]$
- ▶ others:  $f[u] > f[v]$

$$u \rightarrow v \implies f[u] > f[v]$$

## DAG

no back edge  $\iff$  DAG  $\iff \exists$  topo. ordering

Toposort algorithm by Tarjan (probably), 1976

DFS on digraph,  $u \rightarrow v$ :

- ▶ ~~back edge~~:  $f[u] < f[v]$
- ▶ others:  $f[u] > f[v]$

$$u \rightarrow v \implies f[u] > f[v]$$

Toposort: sort vertices in *decreasing* order of their *finish* times.



# Kahn's toposort algorithm

Kahn's toposort algorithm (1962; Problem 5.11)

- ▶ queue for source vertices ( $\text{in}[v] = 0$ )
- ▶ repeat: dequeue  $v$ , delete it, output it

# Kahn's toposort algorithm

## Kahn's toposort algorithm (1962; Problem 5.11)

- ▶ queue for source vertices ( $\text{in}[v] = 0$ )
- ▶ repeat: dequeue  $v$ , delete it, output it

## Lemma

*Every DAG has at least one source (and at least one sink vertex).*

# Kahn's toposort algorithm

## Kahn's toposort algorithm (1962; Problem 5.11)

- ▶ queue for source vertices ( $\text{in}[v] = 0$ )
- ▶ repeat: dequeue  $v$ , delete it, output it

## Lemma

*Every DAG has at least one source (and at least one sink vertex).*

## Question

What if  $G$  is not a DAG?

# Taking courses

## Taking courses in few semesters (Problem 5.14)

- ▶  $n$  courses
- ▶  $c_1 \rightarrow c_2$
- ▶ goal: taking courses in few semesters

# Taking courses

## Taking courses in few semesters (Problem 5.14)

- ▶  $n$  courses
- ▶  $c_1 \rightarrow c_2$
- ▶ goal: taking courses in few semesters

critical path *OR* longest path

# Taking courses

## Taking courses in few semesters (Problem 5.14)

- ▶  $n$  courses
- ▶  $c_1 \rightarrow c_2$
- ▶ goal: taking courses in few semesters

critical path *OR* longest path

## Remark

For general digraph, LONGEST-PATH is NP-hard.

# Line up

## Line up (Problem 5.16)

1.  $i$  hates  $j$ :  $i \prec j$
2.  $i$  hates  $j$ :  $\#i < \#j$

BFS

# Hamiltonian path in DAG

## Hamiltonian path in DAG (Problem 5.10)

- ▶ DAG  $G$
- ▶ HP: path visiting each vertex once



# Hamiltonian path in DAG

## Hamiltonian path in DAG (Problem 5.10)

- ▶ DAG  $G$
- ▶ HP: path visiting each vertex once

### Remark

For general (di)graph, HP is NP-hard.

# Hamiltonian path in DAG

## Hamiltonian path in DAG (Problem 5.10)

- ▶ DAG  $G$
- ▶ HP: path visiting each vertex once

### Remark

For general (di)graph, HP is NP-hard.

DAG:  $\exists$  HP  $\iff \exists!$  topo. ordering

# Hamiltonian path in DAG

## Hamiltonian path in DAG (Problem 5.10)

- ▶ DAG  $G$
- ▶ HP: path visiting each vertex once

### Remark

For general (di)graph, HP is NP-hard.

DAG:  $\exists$  HP  $\iff \exists!$  topo. ordering

### Proof.

$\Leftarrow$ : By construction. □

# Hamiltonian path in DAG

## Hamiltonian path in DAG (Problem 5.10)

- ▶ DAG  $G$
- ▶ HP: path visiting each vertex once

### Remark

For general (di)graph, HP is NP-hard.

DAG:  $\exists$  HP  $\iff \exists!$  topo. ordering

Algorithms:

1. toposort, check edges

Proof.

$\Leftarrow$ : By construction. □

# Hamiltonian path in DAG

## Hamiltonian path in DAG (Problem 5.10)

- ▶ DAG  $G$
- ▶ HP: path visiting each vertex once

### Remark

For general (di)graph, HP is NP-hard.

DAG:  $\exists$  HP  $\iff \exists!$  topo. ordering

Algorithms:

1. toposort, check edges
2. the Kahn toposort algorithm

Proof.

$\iff$ : By construction. □

# Decompositions of Graphs

1 DFS and BFS

2 Cycles

3 DAG

4 SCC

5 Biconnectivity

# Digraph as DAG

## Digraph as DAG (Problem 5.3)

Every digraph is a dag of its SCCs.

### Remark

Two tiered structure of digraphs:

- ▶ digraph  $\equiv$  a dag of SCCs
- ▶ SCC: equivalence class over reachability

# SCC

## Kosaraju SCC algorithm, 1978

*"SCCs can be topo-sorted in decreasing order of their highest finish time."*



# SCC

## Kosaraju SCC algorithm, 1978

*"SCCs can be topo-sorted in decreasing order of their highest finish time."*

The vertex with the highest finish time is in a source SCC.

# SCC

## Kosaraju SCC algorithm, 1978

*"SCCs can be topo-sorted in decreasing order of their highest finish time."*

The vertex with the highest finish time is in a source SCC.

### Remark

- ▶ DFS on  $G$ ; DFS/BFS on  $G^T$
- ▶ DFS on  $G^T$ ; DFS/BFS on  $G$

## SCC

Kosaraju SCC algorithm, 1978 (Problem 5.4)

- ▶ 1st DFS  $\xRightarrow{?}$  BFS
- ▶ 2nd DFS  $\xRightarrow{?}$  BFS

# One-to-all reachability

## One-to-all reachability (Problem 5.12)

Digraph  $G = (V, E)$ :

- ▶ given  $v : v \rightsquigarrow^? \forall u$
- ▶  $\exists? v : v \rightsquigarrow \forall u$

# One-to-all reachability

## One-to-all reachability (Problem 5.12)

Digraph  $G = (V, E)$ :

- ▶ given  $v : v \rightsquigarrow^? \forall u$
- ▶  $\exists? v : v \rightsquigarrow \forall u$

SCC;  $\exists!$ source vertex  $v \iff v \rightsquigarrow \forall u$

# One-to-all reachability

## One-to-all reachability (Problem 5.12)

Digraph  $G = (V, E)$ :

- ▶ given  $v : v \rightsquigarrow^? \forall u$
- ▶  $\exists? v : v \rightsquigarrow \forall u$

SCC;  $\exists!$ source vertex  $v \iff v \rightsquigarrow \forall u$

Proof.

- ▶  $\iff$ : (1) source (2)  $\exists!$

# One-to-all reachability

## One-to-all reachability (Problem 5.12)

Digraph  $G = (V, E)$ :

- ▶ given  $v : v \rightsquigarrow^? \forall u$
- ▶  $\exists? v : v \rightsquigarrow \forall u$

SCC;  $\exists!$ source vertex  $v \iff v \rightsquigarrow \forall u$

Proof.

- ▶  $\Leftarrow$ : (1) source (2)  $\exists!$
- ▶  $\Rightarrow$ : By contradiction.



# Impacts of vertices

## Impacts of vertices (Problem 5.13)

Digraph  $G$ :

$$\text{impact}(v) = |\{w : v \rightsquigarrow w\}|$$

- ▶  $\arg \min_v \text{impact}(v)$
- ▶  $\arg \max_v \text{impact}(v)$



# Impacts of vertices

## Impacts of vertices (Problem 5.13)

Digraph  $G$ :

$$\text{impact}(v) = |\{w : v \rightsquigarrow w\}|$$

- ▶  $\arg \min_v \text{impact}(v)$
- ▶  $\arg \max_v \text{impact}(v)$

$\arg \min_v \text{impact}(v) \in \text{SCC of smallest cardinality}$

# Impacts of vertices

## Impacts of vertices (Problem 5.13)

Digraph  $G$ :

$$\text{impact}(v) = |\{w : v \rightsquigarrow w\}|$$

- ▶  $\arg \min_v \text{impact}(v)$
- ▶  $\arg \max_v \text{impact}(v)$

$\arg \min_v \text{impact}(v) \in \text{SCC of smallest cardinality}$

## Question

$\forall v$  : computing  $\text{impact}(v)$ .

# One-way streets

## One-way streets (Problem 5.15)

Digraph  $G$  for city:

1.  $\forall u, v : u \rightsquigarrow v$
2.  $s : s \rightsquigarrow v \rightsquigarrow s$

# One-way streets

## One-way streets (Problem 5.15)

Digraph  $G$  for city:

1.  $\forall u, v : u \rightsquigarrow v$
2.  $s : s \rightsquigarrow v \rightsquigarrow s$

(2)  $\{v \mid s \rightsquigarrow v\}$  is an SCC

# Connectivity

## Connectivity (Problem 5.7)

**Prove:** connected undirected graph  $G$ :

$$\exists v : G \setminus v \text{ is still connected}$$

**Example:** strongly connected digraph  $G$ :

$$\exists v : G \setminus v \text{ is not strongly connected}$$

**Example:** digraph  $G$  with 2 SCCs:

$$(G + e) \text{ is not strongly connected}$$

# 2SAT

## 2SAT (Problem 5.17)

$$I : (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_2) \wedge (\overline{x_3} \vee x_4) \wedge (\overline{x_1} \vee x_4)$$

# 2SAT

## 2SAT (Problem 5.17)

$$I : (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_2) \wedge (\overline{x_3} \vee x_4) \wedge (\overline{x_1} \vee x_4)$$

$$\alpha \vee \beta \equiv \overline{\alpha} \rightarrow \beta \equiv \overline{\beta} \rightarrow \alpha$$

# 2SAT

## 2SAT (Problem 5.17)

$$I : (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_2) \wedge (\overline{x_3} \vee x_4) \wedge (\overline{x_1} \vee x_4)$$

$$\alpha \vee \beta \equiv \overline{\alpha} \rightarrow \beta \equiv \overline{\beta} \rightarrow \alpha$$

Implication graph  $G_I$ .



# 2SAT

## 2SAT (Problem 5.17)

$$I : (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_2) \wedge (\overline{x_3} \vee x_4) \wedge (\overline{x_1} \vee x_4)$$

$$\alpha \vee \beta \equiv \overline{\alpha} \rightarrow \beta \equiv \overline{\beta} \rightarrow \alpha$$

Implication graph  $G_I$ .

## Theorem

$$\exists \text{ SCC } \exists x : v_x \in \text{SCC} \wedge v_{\overline{x}} \in \text{SCC} \iff I \text{ is not satisfiable.}$$

# 2SAT

## 2SAT (Problem 5.17)

$$I : (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_2) \wedge (\overline{x_3} \vee x_4) \wedge (\overline{x_1} \vee x_4)$$

$$\alpha \vee \beta \equiv \overline{\alpha} \rightarrow \beta \equiv \overline{\beta} \rightarrow \alpha$$

Implication graph  $G_I$ .

## Theorem

$$\exists \text{ SCC } \exists x : v_x \in \text{SCC} \wedge v_{\overline{x}} \in \text{SCC} \iff I \text{ is not satisfiable.}$$

## Reference

# 2SAT

## 2SAT (Problem 5.17)

$$I : (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_2) \wedge (\overline{x_3} \vee x_4) \wedge (\overline{x_1} \vee x_4)$$

$$\alpha \vee \beta \equiv \overline{\alpha} \rightarrow \beta \equiv \overline{\beta} \rightarrow \alpha$$

Implication graph  $G_I$ .

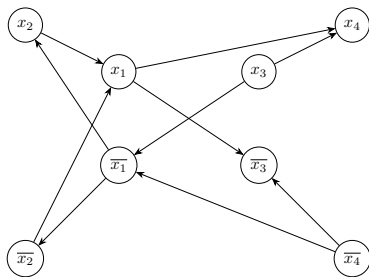
## Theorem

$$\exists \text{ SCC } \exists x : v_x \in \text{SCC} \wedge v_{\overline{x}} \in \text{SCC} \iff I \text{ is not satisfiable.}$$

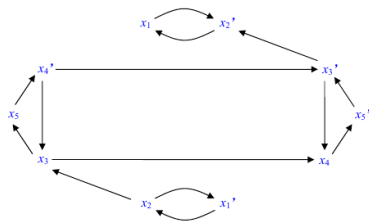
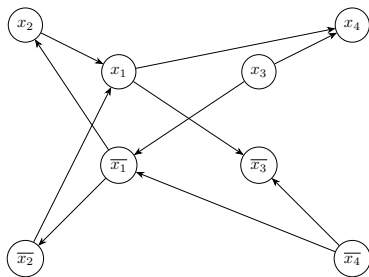
## Reference

- ▶ “A Linear-time Algorithm for Testing the Truth of Certain Quantified Boolean Formulas” by Bengt Aspvall, Michael Plass, and Robert Tarjan, 1979.

## 2SAT



## 2SAT



# Decompositions of Graphs

- 1 DFS and BFS
- 2 Cycles
- 3 DAG
- 4 SCC
- 5 Biconnectivity**

# Biconnectivity algorithm in one word

Back!

# Biconnectivity algorithm in two questions

(1) When and how to update  $\text{back}[v]$ ?



# Biconnectivity algorithm in two questions

(1) When and how to update  $\text{back}[v]$ ?

(2) When and how to identify a bicomponent?

# Biconnectivity algorithm

Initialization of  $\text{back}[v]$  (Problem 5.6)

$$\text{back}[v] = d[v] \text{ vs. } \text{back}[v] = \infty, 2(n+1)$$

# Biconnectivity algorithm

## Initialization of $\text{back}[v]$ (Problem 5.6)

$$\text{back}[v] = d[v] \text{ vs. } \text{back}[v] = \infty, 2(n+1)$$

- ▶ if updated

# Biconnectivity algorithm

## Initialization of $\text{back}[v]$ (Problem 5.6)

$$\text{back}[v] = d[v] \text{ vs. } \text{back}[v] = \infty, 2(n+1)$$

- ▶ if updated
- ▶ if never updated:

$$\text{wBack} = \infty > d[v] \text{ vs.}$$

# Biconnectivity algorithm

## Initialization of $\text{back}[v]$ (Problem 5.6)

$$\text{back}[v] = d[v] \text{ vs. } \text{back}[v] = \infty, 2(n+1)$$

- ▶ if updated
- ▶ if never updated:

$$\text{wBack} = \infty > d[v] \text{ vs. } \text{wBack} = d[w] > d[v]$$

# Root cutnode

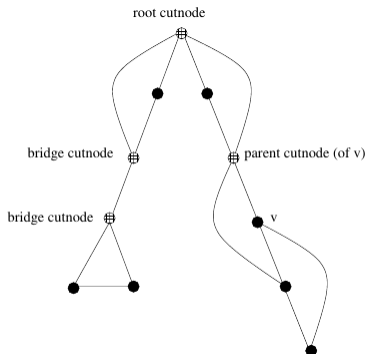
## Root cutnode (Problem 5.5)

$v$  is a cutnode  $\iff \text{OutDegree}[v] \geq 2$

# Root cutnode

## Root cutnode (Problem 5.5)

$v$  is a cutnode  $\iff \text{OutDegree}[v] \geq 2$



# $K$ -core of a graph

## Planning a party (Problem 5.27)

- ▶ undirected graph  $G$
- ▶ subgraph  $G' = (V', E')$ :

$$\forall v' \in V : K(v') \geq 5 \wedge D(v') \geq 5$$



# $K$ -core of a graph

## Planning a party (Problem 5.27)

- ▶ undirected graph  $G$
- ▶ subgraph  $G' = (V', E')$ :

$$\forall v' \in V : K(v') \geq 5 \wedge D(v') \geq 5$$

$K$ -core of a graph:

$$\forall v' \in V' : \deg[v'] \geq k$$

# $K$ -core of a graph

## Planning a party (Problem 5.27)

- ▶ undirected graph  $G$
- ▶ subgraph  $G' = (V', E')$ :

$$\forall v' \in V : K(v') \geq 5 \wedge D(v') \geq 5$$

$K$ -core of a graph:

$$\forall v' \in V' : \deg[v'] \geq k$$

Iteratively delete nodes  $v$  of  $K(v) < 5 \vee D(v) < 5$

