Datastructuren

Data Structures

Hendrik Jan Hoogeboom

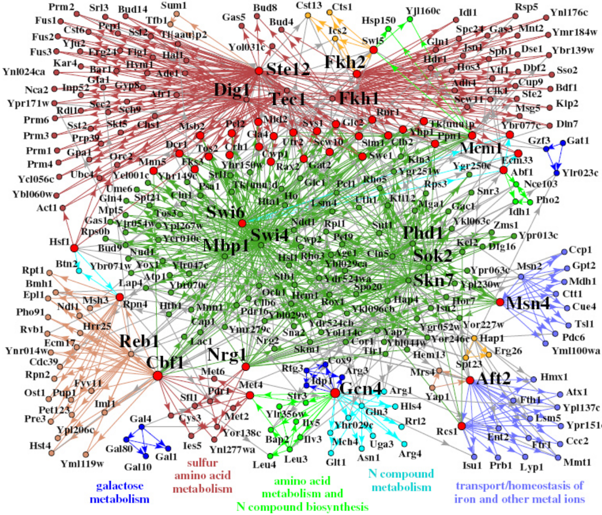
Informatica – LIACS
Universiteit Leiden

najaar 2023

Table of Contents I

Contents

transcription regulatory interactions



Directed network modules, Palla etal. New Journal of Physics, 2007.
zie ook college SNACS

graph definition

zie FoCS en Algoritmiek!

Definition

A *graph* is a pair $G = (V, E)$ where:

- V is a set of *vertices*, or *nodes*
- $E \subseteq V \times V$ is a set *edges*, or *arcs*, *lines*

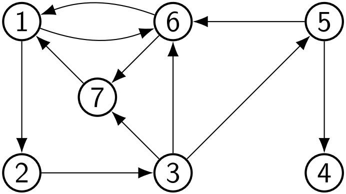
directed / undirected

vertices / edges can have labels (string, number)

complexity in $|V|$ and $|E|$ $\quad |E| \leqslant |V|^2$

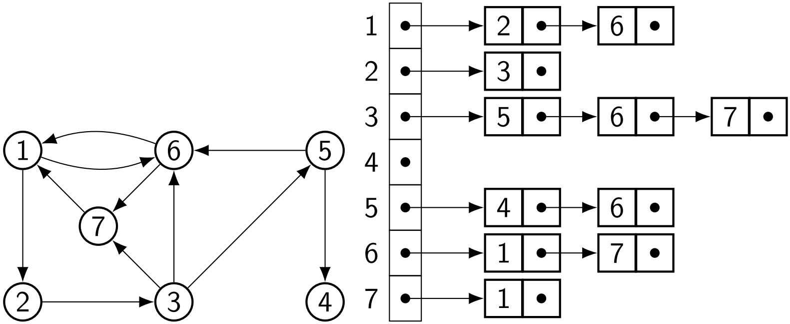
Contents

adjacency matrix



	1	2	3	4	5	6	7
1	·	1	·	·	·	1	·
2	·	·	1	·	·	·	·
3	·	·	·	·	1	1	1
4	·	·	·	·	·	·	·
5	·	·	·	1	·	1	·
6	1	·	·	·	·	·	1
7	1	·	·	·	·	·	·

adjacency lists



representation

space matrix $O(|V|^2)$
lists $O(|V| + |E|)$ $|E| \leqslant |V|^2$

data science / network analysis
huge graphs, few bits per node
sparse graphs

operations 'abstract'

- $(u, v) \in E$
- all outgoing edges
- all incoming edges

Contents

graph traversal

DepthFS	pre-order	stack
BreadthFS	level-order	queue

– tree-traversal + marking nodes

DFS nodes can be twice on stack

structure *spanning tree*

tree, forward, back, cross

depth first search

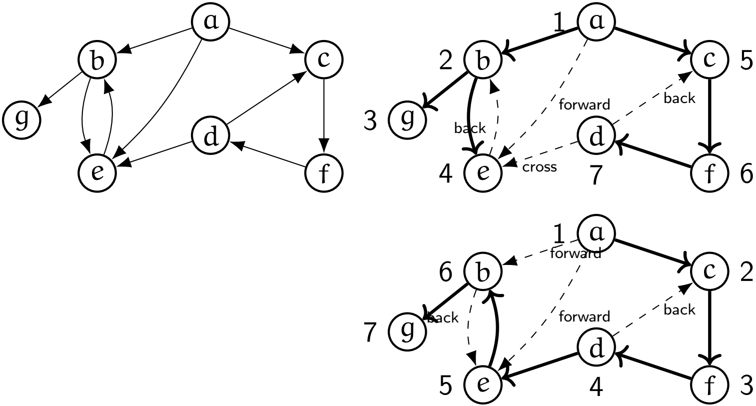
Recursive DFS

```
DFS(v)
    visit(v)
    mark(v)
    for each w adjacent to v
    do  if w is not marked
        then DFS(w)
        fi
    od
end // DFS
```

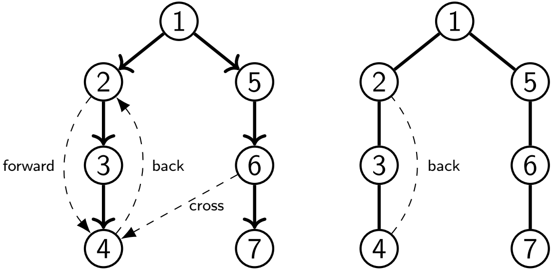
Iterative DFS

```
// start with unmarked nodes
S.push(init)
while S is not empty
do  v = S.pop()
    if v is not marked
    then mark v
        for each edge from v to w
        do  if w is unmarked
            then S.push(w)
            fi
        od
    fi
od
```

dfs tree (directed)



dfs edges

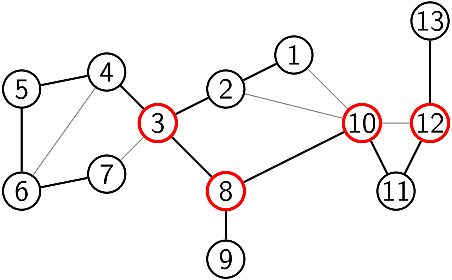


applications of DFS

A DFS traversal itself and the forest-like representation of the graph it provides have proved to be extremely helpful for the development of efficient algorithms for checking many important properties of graphs. Note that the DFS yields two orderings of vertices: the order in which the vertices are reached for the first time (pushed onto the stack) and the order in which the vertices become dead ends (popped off the stack). These orders are qualitatively different, and various applications can take advantage of either of them. [Levitin, Design & Analysis of Algorithms]

- articulation points
- topological sorting

articulation points



articulation points

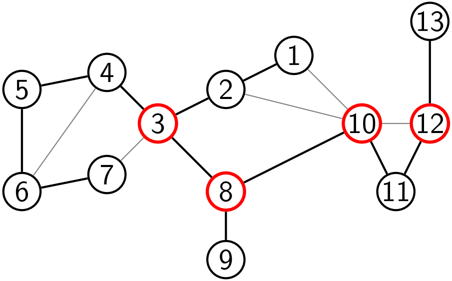
construct DFS tree for graph

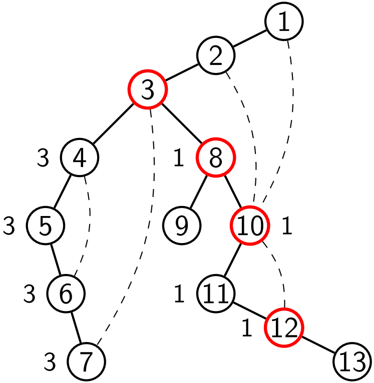
Lemma

A vertex v is an articulation point if either

- *v is the root, and has two or more children, or*
- *v has a (strict) subtree, and no node in the subtree has a back edge that reaches above v.*

dfs and articulation points



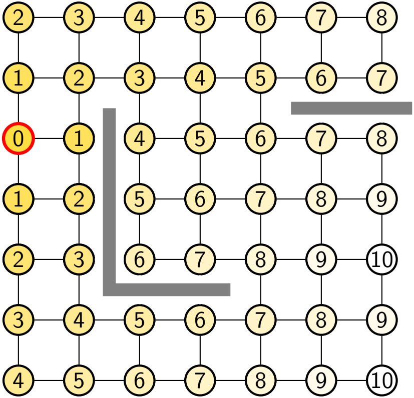


breadth-first search

Iterative BFS

```
    // Q is a queue of vertices
    // start with unmarked nodes
Q.enqueue(init)
dist[init] = 0
while Q is not empty
do  v = Q.dequeue()
    newdist = dist[v] + 1
    for all edges from v to w
    do  if w is not marked
        then Q.enqueue(w)
             mark w
             dist[w] = newdist
        fi
    od
od
```

bfs: 'floodfill'



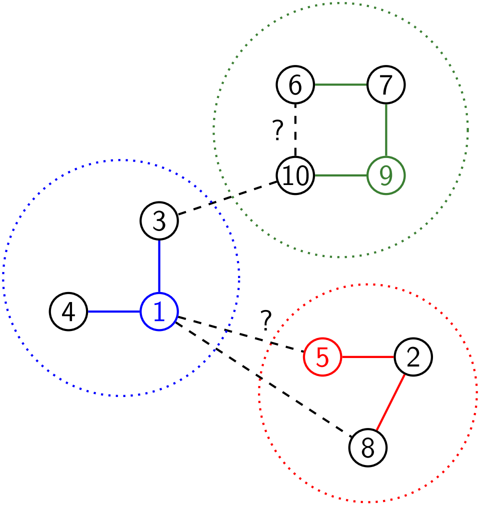
Contents

Algorithms from the Book ⊠

124	union-find	Galler and Fischer
116	Knuth-Morris-Pratt	pattern matching
99	Blum,Floyd,Pratt,Rivest,Tarjan	median
94	binary search	
88	Floyd-Warshall	all-pairs shortest path
84	Euclidean algorithm	greatest common divisor (GCD)
75	quicksort	Tony Hoare
63	Huffman coding	data compression
55	Schwartz-Zippel lemma	polynomial identity
54	Miller-Rabin	primality test
48	depth first search	
45	sieve of Eratosthenes	primes
45	Dijkstra	shortest path
44	Gentry	homomorphic encryption
43	Cooley–Tukey	fast Fourier transform

accessed 22.10'23

application Union-Find



Union-Find

- $\textsc{Initialize}$: construct the initial partition; each component consists of a singleton set $\{d\}$, with $d \in D$.
- $\textsc{Find}$: retrieves the name of the component, i.e, $\textsc{Find}(u) = \textsc{Find}(v)$ iff u and v belong to the same set in the partition.
- $\textsc{Union}$: given two elements u and v the sets they belong to are merged. Has no effects when u and v already belong to the same set.
  Usually it is assumed that u, v are representatives, i.e., names of components, not arbitrary elements.

name array

1	2	3	4	5	6	7	8	9	10	
1	2	3	4	5	6	7	8	9	10	find

1	2	3	4	5	6	7	8	9	10	
1	2	1	4	5	6	6	5	9	9	find

$\textsc{Union}(9,6)$

1	2	3	4	5	6	7	8	9	10	
1	2	1	4	5	9	9	5	9	9	find

lists

1	2	3	4	5	6	7	8	9	10	
1	2	3	4	5	6	7	8	9	10	find
1	2	3	4	5	6	7	8	9	10	next
1	1	1	1	1	1	1	1	1	1	size

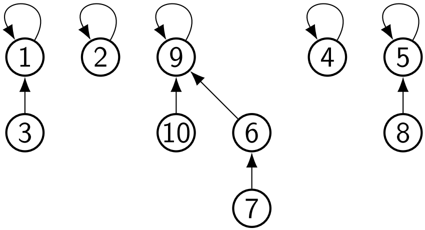
1	2	3	4	5	6	7	8	9	10	
1	2	1	4	5	6	6	5	9	9	find
3	2	1	4	8	7	6	5	10	9	next
2	1	.	1	2	2	.	.	2	.	size

UNION(9, 6)

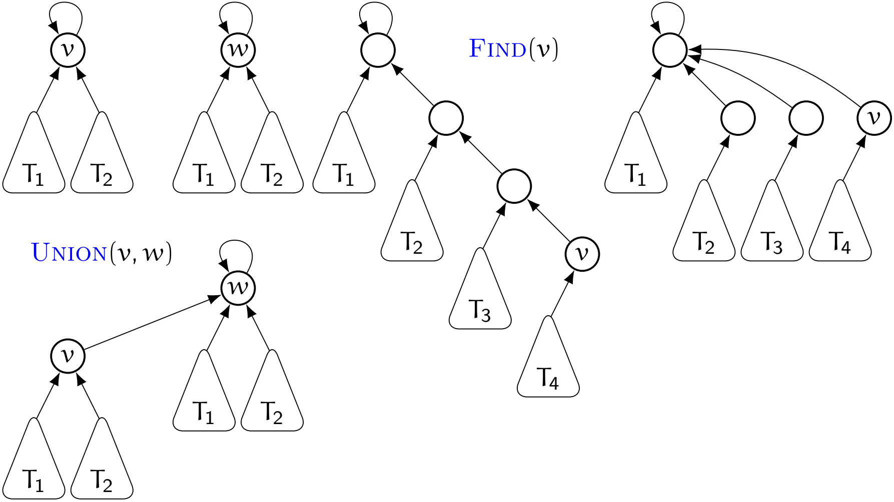
1	2	3	4	5	6	7	8	9	10	
1	2	1	4	5	9	9	5	9	9	find
3	2	1	4	8	10	6	5	7	9	next
2	1	.	1	2	.	.	.	4	.	size

union-find with path-compression

1	2	3	4	5	6	7	8	9	10	
1	2	1	4	5	9	6	5	9	9	parent
2	1	.	1	2	.	.	.	3	.	height

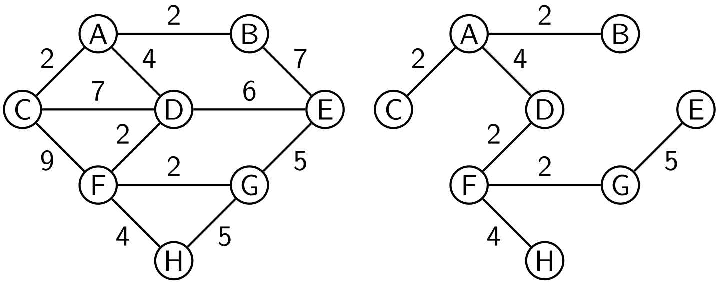


path compression



Contents

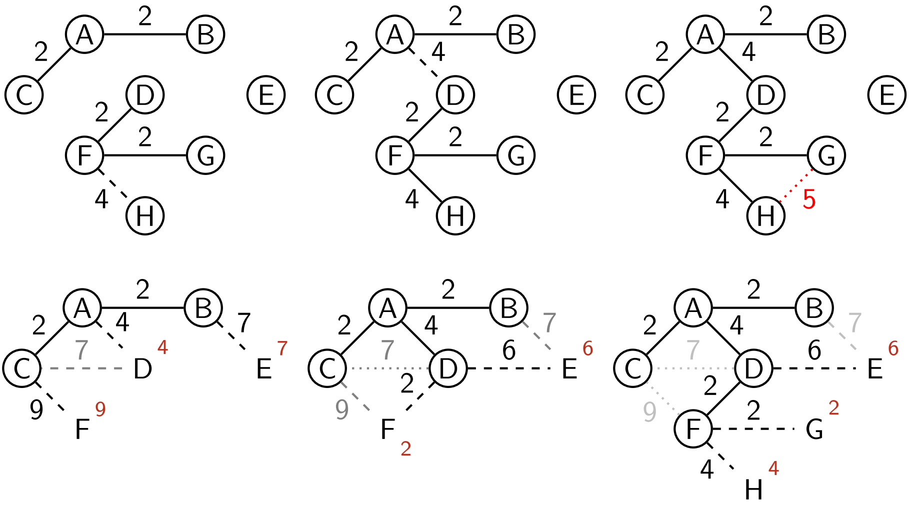
minimal spanning tree



Definition (Minimal spanning tree of weighted graph)

A tree containing all nodes of the graph, with minimal total sum of edge weights

minimal spanning tree Kruskal vs. Prim



minimal spanning tree - Kruskal

Kruskal (high level)

```
repeat
    consider edge with smallest weight
    if it does not yield a cycle
    then add it to the tree
    else discard the edge
    fi
until no edges left
```

minimal spanning tree - Prim

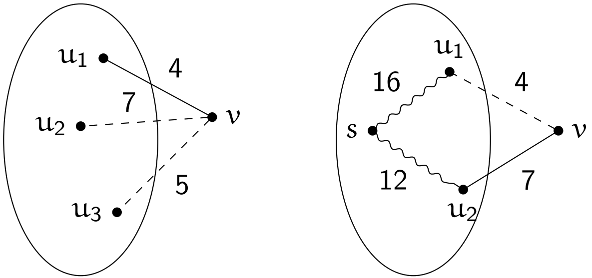
Prim (high level)

```
start with single node
repeat
    consider edge with smallest weight
            that connects node in tree with one outside
    add new node+edge to the tree
until all nodes in tree
```

optimization: for each node outside tree select minimal weight connection to tree

Prim vs Dijkstra

growing tree : keep best candidate connection

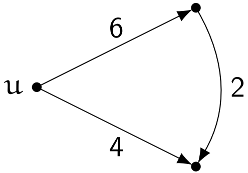


Prim

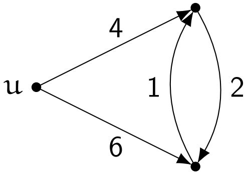
Prim

```
cost[source] = 0        // infinite for other vertices
parent[source] = 0      // code for the root
PQ = V                  // all vertices
while PQ is not empty
do  u is vertex in PQ with minimal cost[u]
    remove u from PQ
    for each edge (u,v)
    do  if length(u,v) < cost[v]
        then cost[v] = length(u,v)
             parent[v] = u
        fi
    od
od
```

directed graphs not supported



Prim fails



Kruskal fails

Contents

Dijkstra

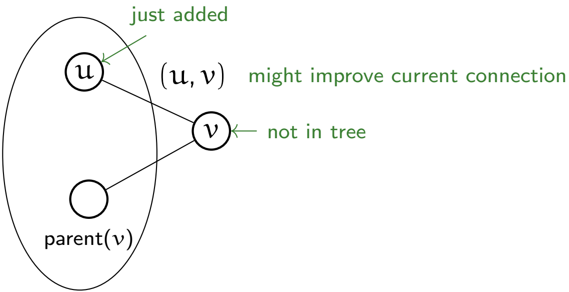
Dijkstra

```
dist[source] = 0     // infinite for other vertices
parent[source] = 0  // code for the root
Q = V               // start with all vertices
while Q is not empty
do  u is vertex in Q with minimal dist[u]
    remove u from Q
    for each edge (u,v)
    do  if dist[u] + length(u,v) < dist[v]
        then dist[v] = dist[u] + length(u,v)
             parent[v] = u
        fi
    od
od
```

shortest path from *fixed source* node to all other nodes

Prim vs Dijkstra

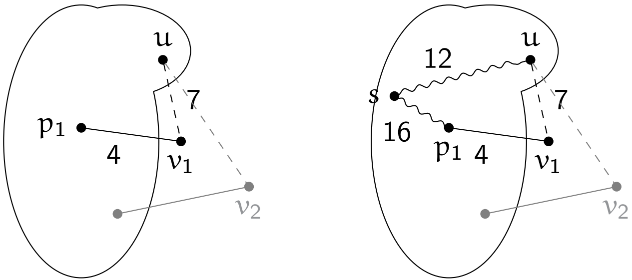
growing tree



Prim	Dijkstra
single edge	length complete path
undirected	(un)directed
negative OK	non-negative
random start	specific source (to all other)

Prim vs Dijkstra update

new node added u: update nodes v_i not in tree



complexity

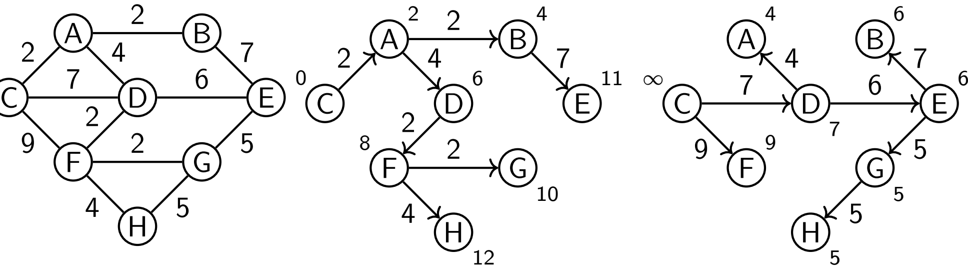
adjacency lists

$|V| \leqslant |E| \leqslant |V|^2$

		heap	better	
minimal	$\lvert V\rvert^2$	$\lvert V\rvert \cdot \lg \lvert V\rvert$	$\lvert V\rvert \cdot \lg \lvert V\rvert$	findmin
each edge	$\lvert E\rvert$	$\lvert E\rvert \cdot \lg \lvert V\rvert$	$\lvert E\rvert$	decreasekey
	$\lvert V\rvert^2$	$\lvert E\rvert \cdot \lg \lvert V\rvert$	$\lvert E\rvert + \lvert V\rvert \cdot \lg \lvert V\rvert$	

heap: better for small number of edges
(or use better priority queue)

distance vs. bottleneck



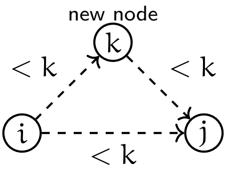
all pairs

$$L^k(i,j) = \min(\ L^{k-1}(i,j),\ L^{k-1}(i,k) + L^{k-1}(k,j)\ )$$

nodes $1, 2, \ldots, n$

L^k path via nodes $\leqslant k$

L^0 only single edges $\sim$ adjacency matrix



all pairs shortest distance

$$L^k(i,j) = \min(\ L^{k-1}(i,j),\ L^{k-1}(i,k) + L^{k-1}(k,j)\ )$$

Floyd-Warshall

```
// initially dist equals the adjacency matrix
for each edge (i,j)
do  next[i,j] = i
od
for k from 1 to n
do  for i from 1 to n
    do  for j from 1 to n
        do  if dist[i,k] + dist[k,j] < dist[i,j]
            then dist[i,j] = dist[i,k] + dist[k,j]
                 next[i,j] = next[i,k]
            fi
        od
    od
od
```

Floyd example

partial result A^3, and distances via node 4

$$A^3 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 2 & 1 & 6 \\ 3 & 0 & 1 & 4 \\ \boxed{4} & 1 & 0 & 5 \\ -2 & 0 & -1 & . \end{pmatrix} \end{array}$$

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} . & 6+0 & 6-1 & 6 \\ 4-2 & . & 4-1 & 4 \\ \boxed{5-2} & 5+0 & . & \boxed{5} \\ \boxed{-2} & 0 & -1 & . \end{pmatrix} \end{array}$$

path reconstruction

Path-reconstruction

```
Path(u, v)
    if next[u][v] = null
    thenreturn []
    fi
    path = [v]
    while u != v
    do  v = next[u][v]
        path.insert_at_end(v)
    od
    return path
```

transitive closure

Warshall

```
    // initially conn equals the adjacency matrix
    // with additionally 1=true on the diagonal
for k from 1 to n
do  for i from 1 to n
    do  for j from 1 to n
        do  conn[i,j] = conn[i,j] or ( conn[i,k] and conn[k,j] )
        od
    od
od
```

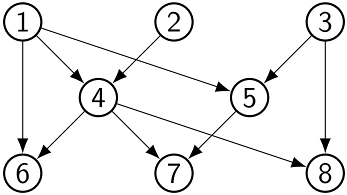
Contents

topological sorting

Let $G = (V, E)$ be a directed graph.

Definition

A *topological ordering* [or *sort*] of G is an ordering $(v_1, \ldots, v_n)$ of V, such that if $(v_i, v_j) \in E$ then $i < j$.

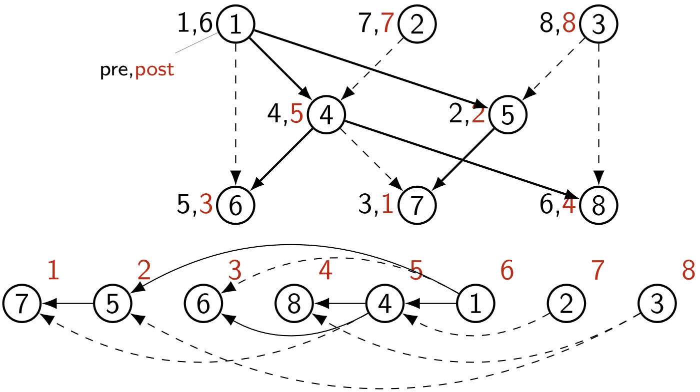


topological sorting

finding a topological sort:

- depth-first search   post-order
- source removal   Kahn's algorithm
  1. pick node without incoming edges
  2. remove that node with outgoing edges. go to step 1.

DFS application: topological sort



end.