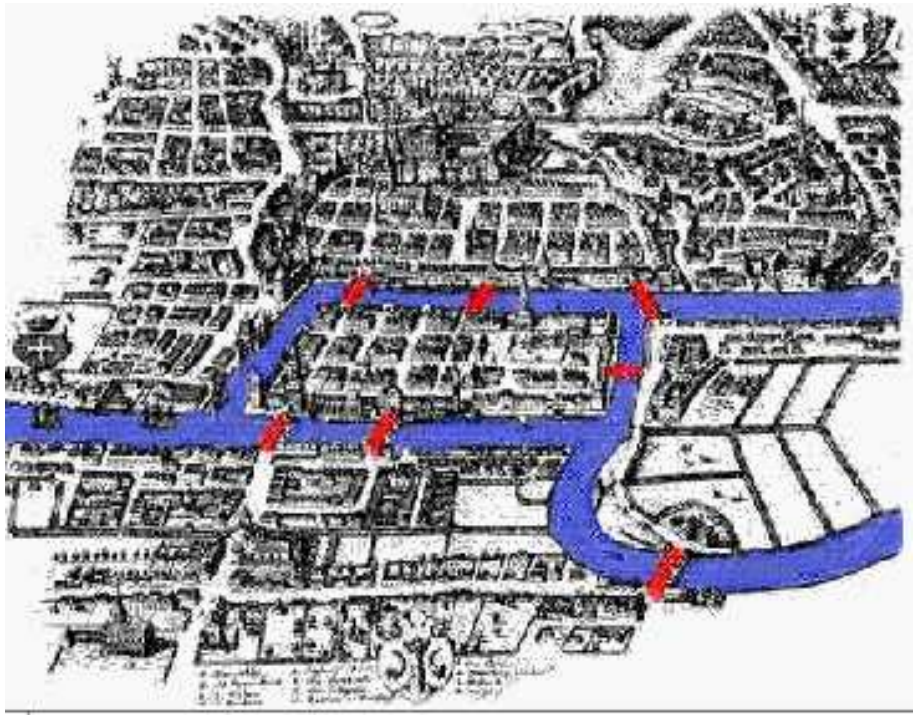
# Excursus: Graph traversal

- Notions
- Graph representation
- Reachability by means of depth-first search
- Searching for cycles





# Directed Graphs $G = (V, E)$

The set of nodes $V$

The set of edges $E \subseteq V \times V$

( "wedges“)



# The Path

A path $p = \langle v_0, \ldots, v_k \rangle$

with a length $k$ connects the nodes $v_0$ and $v_k$ if

in $p$ the successive nodes are connected by edges from $E$, i.e.,

$e_1 = (v_0, v_1) \in E, e_2 = (v_1, v_2) \in E, \ldots, e_k = (v_{k-1}, v_k) \in E.$

Simple path: $v_0, \ldots, v_k$ are different.



# A cycle

Path with the first and the last node coincided.

Simple cycle: all other nodes are different.



# Reachability

$u \in V$ is from $v$ reachable if there is a path from $v$ to $u$.



# Graph representation

Overview:

- What give us the basic operations?
- Adjacency arrays



# Operations

Here only navigation in time $\mathcal{O}(1)$ per node:

Given $v$, find the outgoing nodes

Access: to associated information.

Here: Marking the nodes.

Solution: $V = 1..n$, using additional arrays.

Navigation: Given $v$, find the outgoing nodes

(time $\mathcal{O}(1)$ per node)

# Adjacency arrays

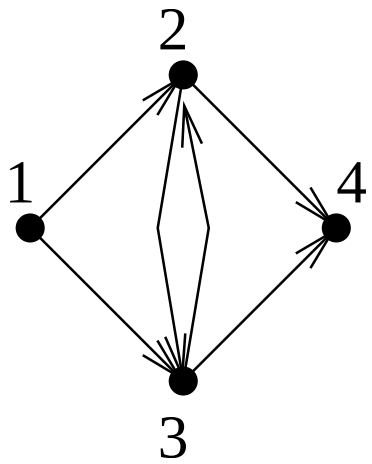
$V = 1..n$

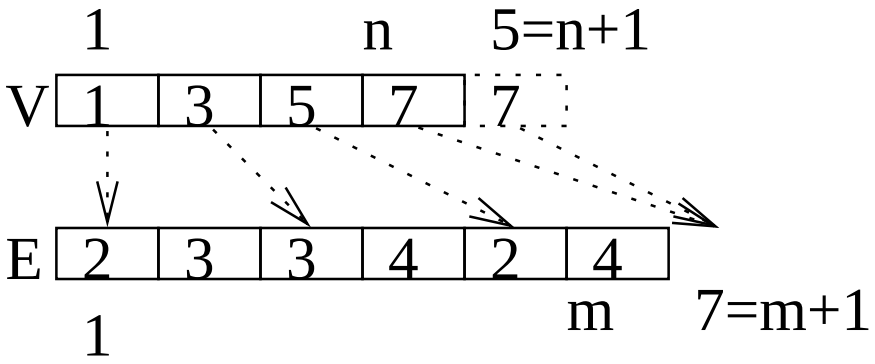
Node array E stores all target

edges of each node (will be grouped)

Edge array V stores the index of the first outgoing edges

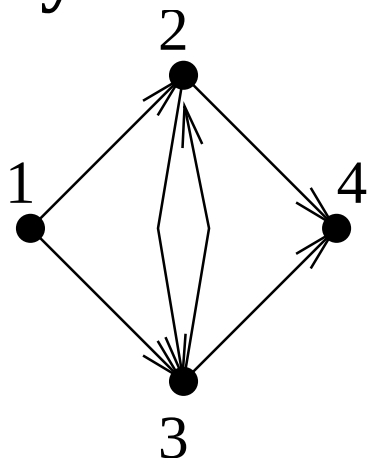
Pseudoentry $V[n+1]$ stores $m+1$

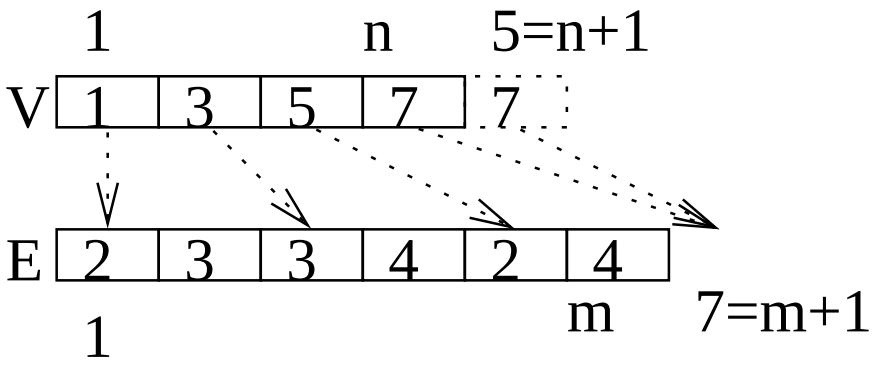



Example: Outgoing grad $(v) = \mathtt{V}[v+1] - \mathtt{V}[v]$

# Adjacency arrays




- Space $\mathcal{O}(m+n)$
- Simple
- Fast navigation. $\mathcal{O}(1)$ time.

  Cache efficient access to edges from the nodes.
- Associate information $\rightsquigarrow$ Records/additional arrays
- Extensible for another operations for the static graphs

Only edges insertion etc. is a problem.



# Reachability by means of Depth First Search

// mark all reachable from $s$ nodes

**Procedure** reachable$(s)$

    mark $s$

    **foreach** $(s,v) \in E$ **do**

        **if** $v$ is not marked **then** reachable$(v)$



# Implementing by means of adjacency arrays

$V[1..n+1]$ // Input

$E[1..m]$ // Graph

$\text{mark}[1..n] = \langle 0, \ldots, 0 \rangle$

**Function** $\text{reachable}(s)$

    $\text{mark}[s] := 1$

    **for** $e := V[s]$ **to** $V[s+1]-1$ **do** // for each $e = (s,v) \in E$

        $v := E[e]$

        **if** $\text{mark}[v] = 0$ **then** $\text{reachable}(v)$



# Correctness

follows from:

**Lemma:** reachable nodes are exactly marked

$\{v \in V : \exists \text{ path } P = \langle s = v_0, \ldots, v = v_k \rangle\}$

**Proof „all reachable will be marked“:**

Induction on the length of the path $k$

$k = 0$: $s$ will be immediately marked.

$k \rightsquigarrow k+1$: by IH $v_k$ will be marked.

(At start calling reachable$(v_k)$.)

$\underbrace{\longrightarrow}_{(v_k, v_{k+1}) \in E}$ "‘**if** mark$[E[v]] = 0$ **then** reachable$(v)$"’ will be accomplished.

$\longrightarrow$ so or so $v_{k+1}$ will be marked



**Lemma:** The reachable nodes are exactly marked

$\{v \in V : \exists \text{ path } P = \langle s = v_0, \ldots, v = v_k \rangle\}$

**Proof „no unreachable nodes will be marked“:**

Induction. Exercise.



# Complexity analysis

**Proposition:** reachable runs in time $\mathcal{O}(m+n)$

**Proof:**

$\leq 1$ recursive call per edge

$\leq 1$ perform statement per node



# Cycle reachable from $s$?

$V[1..n+1]$ // Input

$E[1..m]$ // Graph

$\text{mark}[1..n] = \langle 0,\ldots,0\rangle$

$\text{onPath}[1..n] = \langle 0,\ldots,0\rangle$// flags all nodes on currently explored path

**Function** reachesCycle$(s)$

    mark$[s]$:= 1

    onPath$[s]$:= 1

    **for** $e := V[s]$ **to** $V[s+1]-1$ **do** // foreach $e = (s,v) \in E$

        $v$:= $E[e]$

        **if** onPath$[v]$ **then return** 1 // "backward edge"

        **if** mark$[v] = 0$ **then**

            **if** reachesCycle$(v)$ **then return** 1

        onPath$[s]$:= 0 // backtrack

    **return** 0



# Correctness

**Proposition:** The nodes with onPath$= 1$ define at time a path $P = \langle s = v_0, \ldots, v_k \rangle$.

$\longrightarrow$ notified path corresponds to one from $\langle s, \ldots, v_i \rangle$ reachable cycle $\langle v_i, \ldots, v_k, v_i \rangle$.

# Correctness

**Lemma M:** reachesCycle$(v) = 0 \longrightarrow$ all nodes will be marked,

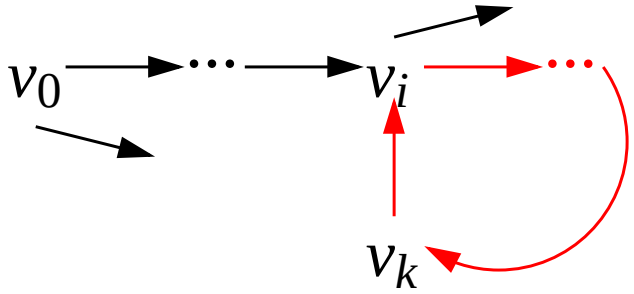
which from $v$ by before not marked nodes reachable are.

**Proof:** analog of the correctness proof of reachable.

**Lemma:** No cycle will be missed.

**Proof:** Assume, reachesCycle$(s) = 0$ but

$\exists$ a path $P = \langle s = v_0, \ldots, v_i, \ldots, v_k, v_i \rangle$




All from $s$ reachable nodes will be marked.

Let $v_i$ be the first marked node from the cycle.

$\longrightarrow$ before reachesCycle$(v_i)$ ends, $v_i, \ldots, v_k$ will be marked.

$\longrightarrow$ by call from reachesCycle$(v_k)$ guild onPath$(v_i)$

$\longrightarrow$ there is a cycle notified !



# Complexity analysis

x **Proposition:** reachesCycle runs in time $\mathcal{O}(m+n)$

**Proof:** analog reachable



# Is there a cycle ?

Initialization for reachesCycle

**return** $\bigvee_{s \in V}$ reachesCycle$(s)$

- the marking will not taking back

  $\rightsquigarrow$ runtime$\mathcal{O}(n+m)$

- correctness proof: analog before



# Strongly connected components

$C \subseteq V$ is a connected component $\Leftrightarrow$

$\forall u, v \in C : \exists$path $u \rightsquigarrow v$.

$C \subseteq V$ is strongly connected component $\Leftrightarrow$

$C$ is a connected component and maximal

(i.e. $\nexists w \in V \setminus C : C \cup \{w\}$ is connected component)



# Strongly connected components

**Proposition:** All strongly connected components we can find in time $\mathcal{O}(m+n)$.

**Proof:** not here.

Idea: depth-first.

Invariant: strongly connected components with reference to all processed nodes and edges which are known.



# Topological sorting

Numerate the nodes $V = \langle v_1, \ldots, v_n \rangle$, so that $\forall (v_i, v_j) \in E : i < j$

This holds iff when

$G = (V, E)$ has no direct cycle —

so $G$ is a directed acyclic graph (Directed Acyclic Graph).



# Topological sorting

**Dept-first search algorithm:** (without proof)

Sequence: order$= \langle\rangle$

**foreach** $s \in V$ **do**

    explore$(s)$

**Procedure** explore$(s)$

    mark $s$

    **foreach** $(s,v) \in E$ **do**

        **if** $v$ is not marked **then** explore$(v)$

    order.pushFront$(s)$



# Topological sorting

**Lemma:** Every DAG contains a node with output 0.

**Proof:** Assuming $\forall v \in V$ : outdegree$(v) > 0$.

$u$:= any node

$S$:= $\{u\}$

**loop**

choose any edge $(u,v) \in E$ // outdegree$(u) > 0$ !

**if** $v \in S$ **then** cycle found // a contradiction (DAG !)

$S$:= $S \cup \{v\}$

$u$:= $v$

assuming $\longrightarrow$ infinite run $\longrightarrow$ $V$ is infinite $\longrightarrow$ a contradiction.



# Topological sorting – another algorithm

Start by an arbitrary node.

Search at a time

Sequence: order$= \langle\rangle$

**while** $\exists s \in V$ : outdegree$(s) = 0$ **do**

order.pushFront$(s)$

remove $s$ and all its incoming edges from $G$



# An important model for graph algorithms:

- Find strongly connected components
- Strongly connected components are simple to process
- Contract the strongly connected components
- It remains a DAG
- DAGs are easily by topological sorting to process.

Example transitive hull $G = (V, E^+)$, $E^+ := \{(u,v) : \exists \text{path } u \rightsquigarrow v\}$