Some Graph Algorithms

1. Depth-First Search (DFS)
2. Topological sort
3. Strongly connected components
4. Shortest paths in graphs
5. Graph colorings

1. Representation of graphs

Let $G = (V, E)$ be a graph and let $u, v \in V$.

- Adjacency list.

  $Adj[u]$ is a list of nodes adjacent to u
  Memory space: $O(|V| + |E|)$
  Disadvantage: there is no quick way to check if $(u, v) \in E$.

- Adjacency matrix $A(G) = a_{ij}$.

  $$a_{ij} = \begin{cases} 1, & \text{if } (v_i, v_j) \in E \\ 0, & \text{otherwise} \end{cases}$$
  Memory space: $O(|V|^2)$.
  Advantage: one can save space by using bit encoding of a_{ij}

One can also use both representations for oriented graphs.

2. Depth-First Search (DFS)

This procedure visits all vertices and edges of $G = (V_G, E_G)$ and colors the vertices in white, gray, and black.

Initially all vertices of G are white. As soon as a vertex v is visited for the first time, we color it gray. As soon as all adjacent to v vertices have been visited, the color of v becomes black.

We assign to each vertex $v \in V_G$ three labels: $d[v]$, $f[v]$ and $\pi[v]$:

$d[v]$: the time interval when v becomes gray

$f[v]$: the time interval when v becomes black

$\pi[v]$: the predecessor of v in DFS.

It holds:

$$\begin{aligned} d[v], f[v] &\in \{1, \ldots, 2|V_G|\}, \\ d[v] &< f[v] \end{aligned}$$

for any $v \in V_G$.

The method constructs a spanning forest W of G, defined by

$$E_W = \{(\pi[v], v) \mid v \in V, \text{ and } \pi[v] \neq \text{nil}\}.$$

The edges of E_W are called <u>tree edges</u>.

Algorithm 1 DFS(G, s);

```
for all u ∈ V_G do
    color[u] := white
    π[u] := nil
time := 0
for all u ∈ V_G do        /* let s be the first vertex */
    if (color[u] = white) then
        DFS-VISIT(u)
```

```
DFS-VISIT(u);
color[u] := gray
time := time + 1
d[u] := time
for all v ∈ Adj[u] do
    if (color[v] = white) then
        π[v] := u
        DFS-VISIT(v)
color[u] := black
time := time + 1
f[u] := time
```

The running time of DFS is $\Theta(|V_G| + |E_G|)$.

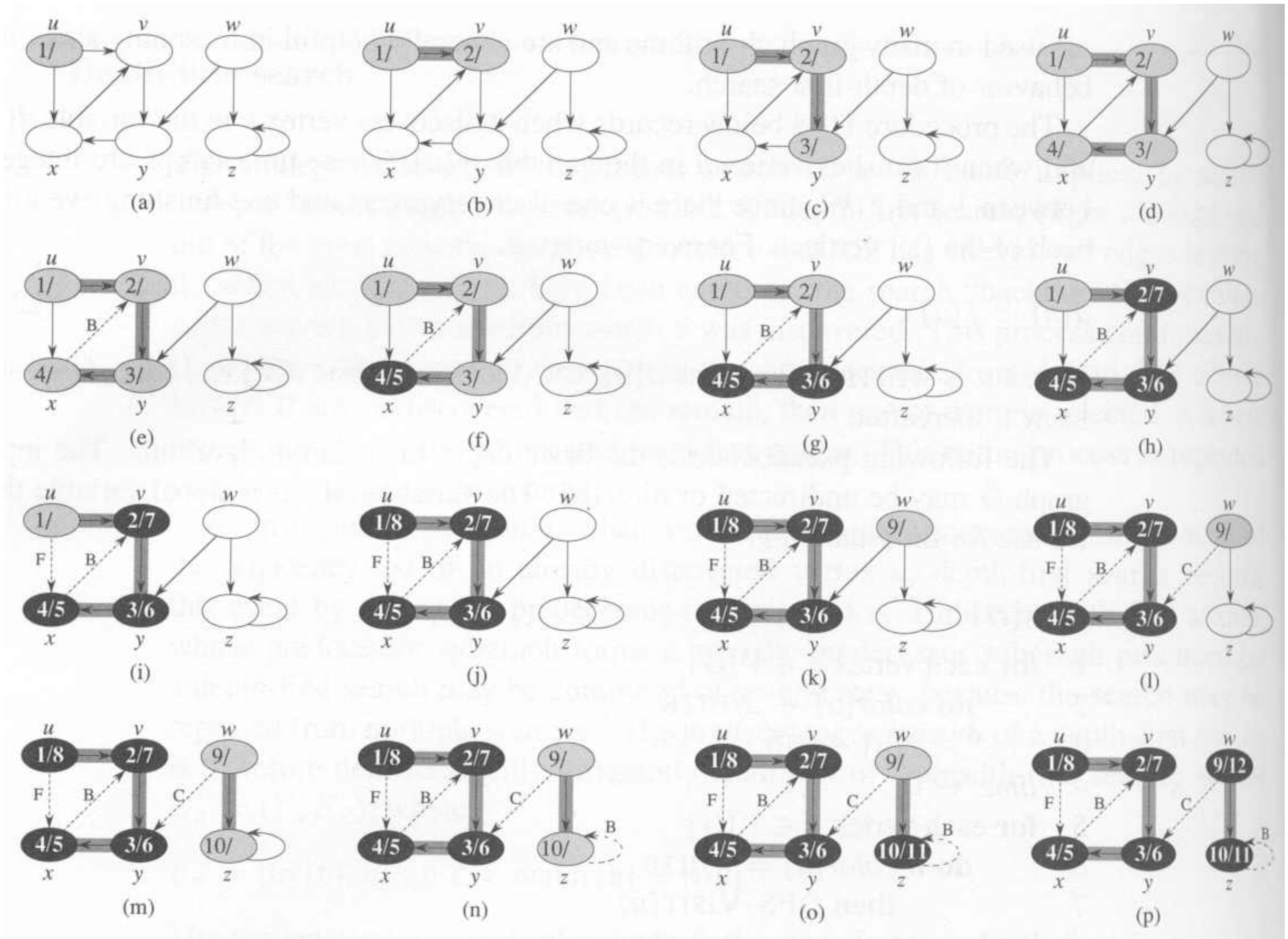

Figure 1: DFS on a directed graph

3. Properties of Depth-First Search

Theorem 1 *Let* $G = (V_G, E_G)$ *be an (oriented or non-oriented) graph and* $u, v \in V_G$*,* $u \neq v$*. Then either:*

- *the intervals* $[d[u], f[u]]$ *and* $[d[v], f[v]]$ *are disjoint. or*
- *the interval* $[d[u], f[u]]$ *is a subinterval of* $[d[v], f[v]]$ *and* u *is the descendant of* v *in the DFS tree, or*
- *the interval* $[d[v], f[v]]$ *is a subinterval of* $[d[u], f[u]]$ *and* v *is the descendant of* u *in the DFS tree.*

Proof. Assume $d[u] < d[v]$.

Case 1. Assume $d[v] < f[u]$. $\Rightarrow v$ was discovered when u was gray $\Rightarrow v$ is a descendant of u.

Since v was discovered after u, if became black before u did.
$\Rightarrow [d[v], f[v]] \subset [d[u], f[u]]$.

Case 2. Assume $f[u] < d[v]$. Since $d[u] < f[u]$ and $d[v] < f[v]$,

$$d[u] < f[u] < d[v] < f[v]$$

$\Rightarrow$ intervals $[d[u], f[u]]$ and $[d[v], f[v]]$ are disjoint. □

Corollary 1 *A vertex* v *is a descendant of* u *(in DFS forest) if and only if*

$$d[u] < d[v] < f[v] < f[u]$$

Theorem 2 *Let $G = (V_G, E_G)$ be an (oriented or non-oriented) graph, and W be its DFS forest and $u, v \in V_G$. Then v is the descendant of u if and only if in time $d[u]$ there exists a path from u to v, consisting of white vertices only.*

Proof.
"⇒"-part: assume v is a descendant of u and $w \in u \rightsquigarrow v$. So, w is a descendant of $u \Rightarrow d[u] < d[w] \Rightarrow$ (C. 1) w is white at time $d[u]$.

"⇐"-part: assume any vertex on the path $u \rightsquigarrow v$ is white at time $d[u]$, but v is not a descendant of u in the DFS tree.

WLOG assume any other vertex on $u \rightsquigarrow v$ is a descendant of u and let w be the predec. of v in $u \rightsquigarrow v$ ($u \rightsquigarrow w \rightarrow v$).

$\Rightarrow$ (C. 1) $f[w] \le f[u]$. v must be discovered after u but before w turns black. Hence,

$$d[u] < d[v] < f[w] \le f[u].$$

$\Rightarrow$ (T. 1) $[d[v], f[v]] \subset [d[u], f[u]]$.
$\Rightarrow$ (C. 1) v is a descendant of u. □

4. Topological Sort

Let $G = (V_G, E_G)$ be a DAG (Directed Acyclic Graph), that is a graph without oriented cycles or loops.

The Problem:
Construct a numbering $\psi : V \mapsto \{1, \ldots, |V|\}$, such that:

$$(u \to v) \in E \qquad \Rightarrow \qquad \psi(u) < \psi(v)$$

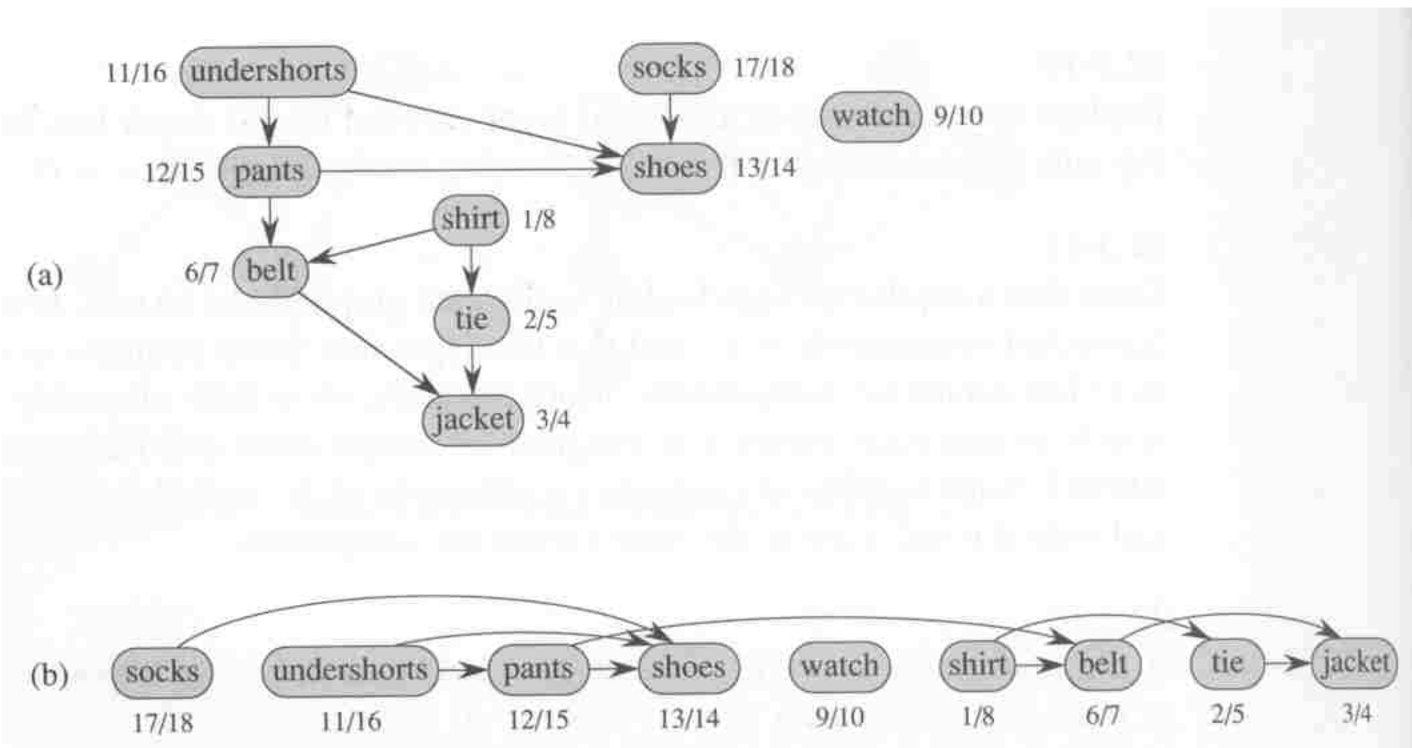

Figure 2: DAG and topological sort

Algorithm 2 TOP-SORT(G);

1. Call $\mathrm{DFS}(G)$ to compute $\{f[u]\}$.
2. Place u to the head of the list as soon as $f[u]$ is computed;
return the list

Proof of of the correctness of this algorithm is based on the following notion and lemma:

Definition 1 *An edge $(u, v) \in V_G$ is called back edge if v is an ancestor of u in the DFS tree (or u is a descendant of v).*

Lemma 1 [1]. *An oriented graph G is a DAG if and only if the DFS forest has no back edges.*

Proof:

"$\Rightarrow$" Assume $\exists$back edge (u, v)
$\Rightarrow u$ is a descendant of v
$\Rightarrow \exists$path $v \rightsquigarrow u$ in G and G contains a cycle, a contradiction.

"$\Leftarrow$" Assume G contains a cycle C.
Let v be the first vertex of C in DFS and $(u, v) \in E_C$.
$\Rightarrow \exists$"white" path $v \rightsquigarrow u$ at time $d[v]$. (T. 2)
$\Rightarrow u$ is a descendant of v and (u, v) is a back edge, a contradiction. $\square$

Theorem 3 $[1]$. *Let G be a DAG. Then $\textsc{Top-Sort}(G)$ algorithm constructs a topological sorting of G.*

Proof:

We show that $\forall u, v \in V \ (u \neq v)$
$(u, v) \in E_G \Rightarrow f[u] > f[v]$.

Consider an edge (u, v) explored by DFS.
$\Rightarrow color(u) = gray$ and $color(v) \neq gray$.

(*$color(v) = gray \Rightarrow v$* is an ancestor of u
$\Rightarrow (u, v)$ is a back edge.)

$\Rightarrow color(v) \in \{white, black\}$.

a. $[color(v) = white] \Rightarrow v$ is a descendant of u
$\Rightarrow f[v] < f[u]$.

b. $[color(v) = black] \Rightarrow f[v] < d[u] < f[u]$. □

The running time of $\textsc{Top-Sort}(G)$ is $\Theta(|V_G| + |E_G|)$.

5. Strongly connected components (SCC)

Definition 2 *Let $G = (V_G, E_G)$ be an oriented graph.*
A strongly connected component is a maximal (by inclusion) vertex set $K \subseteq V_G$, such that for any $u, v \in K$ there exist oriented paths $u \rightsquigarrow v$ and $v \rightsquigarrow u$.

The Problem:
Given $G = (V_G, E_G)$, partition V_G into SCC.

Denote $G^T = (V_G, E_G^T)$, where

$$E_G^T = \{(u \to v) \mid (v \to u) \in E_G\}.$$

Algorithm 3 $\mathrm{SCC}(G)$;

1. Call $\mathrm{DFS}(G)$ to compute the numbers $\{f[u]\}$
2. Construct G^T
3. Call $\mathrm{DFS}(G^T)$, where the vertices are ordered according to $f[u]$ taken in decreasing order
4. Output the vertices of the DFS trees of G^T as strongly connected components

The running time of $\mathrm{SCC(G)}$ is $\Theta(|V_G| + |E_G|)$.

Theorem 4 [1]. $\mathrm{SCC(G)}$ *partitions V_G into SCCs.*

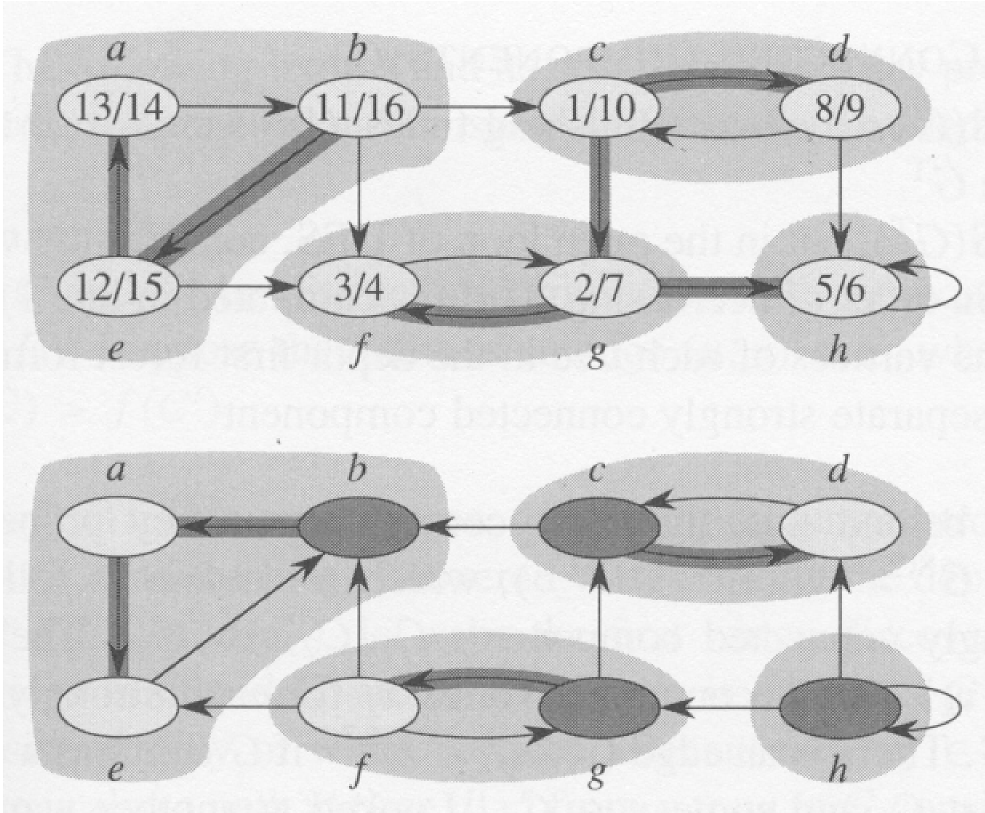

Figure 3: DFS labeling and Strongly Connected Components

Lemma 2 *Let C and C' be distinct SCC in directed graph $G = (V_G, E_G)$. Let $u, v \in C$, $u', v' \in C'$ and there $\exists$path $u \rightsquigarrow u'$. Then there is no path $v' \rightsquigarrow v$.*

Proof:
If $v' \rightsquigarrow v$ then there are paths $u \rightsquigarrow u' \rightsquigarrow v' \rightsquigarrow v \rightsquigarrow u$, so u and v' belong to the same component, a contradiction. □

For a set $U \subseteq V_G$ denote

$$d(U) = \min_{u \in U}\{d[u]\}$$
$$f(U) = \max_{u \in U}\{f[u]\}$$

Lemma 3 *Let C and C' be distinct SCCs in directed graph $G = (V_G, E_G)$. If $\exists$edge $(u, v) \in E_G$ with $u \in C$ and $v \in C'$ then $f(C) > f(C')$.*

Proof:
<u>Case 1.</u> Assume $d(C) < d(C')$, and let $x \in C$, s.t. $d(C) = d[x]$.
$\Rightarrow$ at time $d[x]$ all vertices of C and C' are white
$\Rightarrow \forall w \in C'$ $\exists$path $x \leadsto u \to v \leadsto w$
$\Rightarrow$ all vertices of C and C' are descendants of x in DFS tree
$\Rightarrow f(x) = f(C) > f(C')$.

<u>Case 2.</u> Assume $d(C) > d(C')$, and let $y \in C'$, s.t. $d(C') = d[y]$.
$\Rightarrow$ at time $d[y]$ all vertices of C' are white
$\Rightarrow$ all vertices of C' are descendants of y, so $f[y] = f(C')$

Since $\exists u \to v$, there is no path $v \leadsto u$ (Lemma 2)
$\Rightarrow$ all vertices of C are white at time $f[y]$
$\Rightarrow \forall w \in C$, $f[w] > f[y] \Rightarrow f(C) > f[y] = f(C')$ $\square$

Corollary 2 *Let C and C' be distinct SCCs in directed graph $G = (V_G, E_G)$. If $\exists$edge $(u, v) \in E^T$ with $u \in C$ and $v \in C'$ then $f(C) < f(C')$.*

Proof: $(u, v) \in E^T \Rightarrow (v, u) \in E$. Since the SCCs in G and G^T are the same, Lemma 3 implies $f(C) < f(C')$. $\square$

Proof of Theorem 4:

Claim: the vertices of each tree constructed in step 3 form a SCC. Induction on the number k of DFS trees. Trivial for $k = 0$.

Assume each of the first k trees is a SCC and consider the $(k+1)$-th tree $T = (V_T, E_T)$. Let u be its root and $u \in C$ for some SCC C. We show $C = V_T$.

At time the DFS on G^T visits u, all vertices of C are white
$\Rightarrow$ all vertices of C are descendants of u in DFS tree
$\Rightarrow \forall w \in C$, $w \in V_T$, i.e.

$$C \subseteq V_T$$

To show the equality, assume $C \subset V_T$ and let v be the first vertex of $V_T - C$ visited by the DFS on G^T. Let $v \in C'$ for some SCC C'.
$\Rightarrow f(C) < f(C')$ (Cor. 2)
$\Rightarrow$ all vertices of C' have already been visited, a contradiction. $\square$

6. Shortest Path Algorithms

- Generalities
- Part I. Single source shortest paths
  - The Bellman-Ford algorithm
  - Dijkstra's algorithm
- Part II. All pairs shortest paths
  - The Floyd-Warshall algorithm

6a. Generalities

Let $G = (V_G, E_G)$ be an oriented graph and let $w : E_G \mapsto \mathbf{R}$ be a weight function. Let $P = (v_0, \ldots, v_k)$ be a (oriented) path in G. We define

$$w(P) = \sum_{i=1}^{k} w(v_{i-1}, v_i).$$

For $u, v \in V$ put

$$\delta(u,v) = \begin{cases} \min\limits_{P=(u \rightsquigarrow v)} w(P), & \text{if } \exists \text{path } u \rightsquigarrow v \\ \infty, & \text{otherwise.} \end{cases}$$

Definition 3 *A path $P = (u \rightsquigarrow v)$ is called* <u>*shortest path*</u>*, if*

$$w(P) = \delta(u, v).$$

<u>**Problems:**</u>

Given $G = (V_G, E_G)$ and a weight function w.

- Let $s \in V_G$. Find a shortest path from s to any vertex of G.
- Find a shortest path between any pair of vertices of G.

We represent a path P by the set of predecessors $\{\pi[v]\}$ for $v \in P$, and define the predecessor graph by
$G_\pi = (V_\pi, E_\pi)$, where

$$\begin{aligned} V_\pi &= \{v \in V \mid \pi[v] \neq \text{NIL}\} \cup \{s\} \\ E_\pi &= \{(\pi[v], v) \in E \mid v \in V_\pi - \{s\}\}. \end{aligned}$$

Part I. Single-source shortest paths

Assume G contains no cycle of negative weight.
We construct a shortest paths tree $G' = (V', E')$:

- $V' = \{v \in V \mid \exists \text{path } s \rightsquigarrow v\}$.
- G' is a tree rooted in s.
- $\forall v \in V'$ the path $s \overset{G'}{\rightsquigarrow} v$ is also a shortest path $s \overset{G}{\rightsquigarrow} v$.

Lemma 4 *Let $(v_1, \ldots, v_k)$ be a shortest path $v_1 \rightsquigarrow v_k$. Then $(v_i, \ldots, v_j)$ is a shortest path $v_i \rightsquigarrow v_j$, for all $1 \le i \le j \le k$.*

Lemma 5
a. Let $(v_1, \ldots, v_k, u)$ be a path $v_1 \rightsquigarrow u$ and $(v_k, u) \in E$. Then

$$\delta(v_1, u) \le \delta(v_1, v_k) + w(v_k, u).$$

b. Let $(v_1, \ldots, v_k, u)$ be a shortest path $v_1 \rightsquigarrow u$ and $(v_k, u) \in E$. Then

$$\delta(v_1, u) = \delta(v_1, v_k) + w(v_k, u).$$

Algorithm 4 INITIALIZE-SS(G, w);

for each $v \in V$ **do**
  $d[v] := \infty$
  $\pi[v] :=$ NIL
$d[s] := 0$

6b. Relaxation

Let $(u, v) \in E_G$.

Algorithm 5 $\textsc{Relax}(u, v, w)$;

if $(d[v] > d[u] + w(u, v))$ **then**
  $d[v] := d[u] + w(u, v)$
  $\pi[v] := u$

Assume the procedure $\textsc{Initialize-SS}$ has been applied to a graph $G = (V_G, E_G)$. The relaxation satisfies the following properties:

Lemma 6 *Let $(u, v) \in E_G$. Then right after calling $\textsc{Relax}(u, v, w)$ one has*

$$d[v] \leq d[u] + w(u, v).$$

Lemma 7
a. $d[v] \geq \delta(s, v)$ for all $v \in V_G$.
b. If $d[v] = \delta(s, v)$ then no further call of $\textsc{Relax}$ modifies $d[v]$.

Proof:
a. The inequality is valid right after the initialization, since $d[s] = 0 \geq \delta(s, s)$ and $d[v] = \infty \geq \delta(s, v)$ for $v \neq s$.

Let v be the first vertex for which $\textsc{Relax}$ provides $d[v] < \delta(s,v)$. Then for $(u,v) \in E_G$ right after calling $\textsc{Relax}(u,v,w)$ one has:

$$\begin{aligned} d[u] + w(u,v) &= d[v] \\ &< \delta(s,v) \\ &\leq \delta(s,u) + w(u,v) \quad \text{(by L. 5a)} \end{aligned}$$

Hence, $d[u] < \delta(s,u)$, contradicting the choice of v.

b. Since $d[v] \geq \delta(s,v)$ and $\textsc{Relax}$ does not increase the values of $d[\cdot]$, the assertion is true. □

Lemma 8 *Let $s \rightsquigarrow u \to v$ be a shortest path $s \overset{G}{\rightsquigarrow} v$ and $(u,v) \in E_G$. If prior to the call of $\textsc{Relax}(u,v,w)$ one has $d[u] = \delta(s,u)$, then $d[v] = \delta(s,v)$ for all times afterwards.*

Proof:
$d[u] = \delta(s,u)$ prior to the call of $\textsc{Relax}(u,v,w)$
$\Rightarrow d[u] = \delta(s,u)$ after the call (L. 7b).

One has:

$$\begin{aligned} d[v] &\leq d[u] + w(u,v) \quad \text{(L. 6)} \\ &= \delta(s,u) + w(u,v) \\ &= \delta(s,v) \qquad\qquad\quad \text{(L. 5b).} \end{aligned}$$

On the other hand, by L. 7a one has $d[v] \geq \delta(s,v)$. □

Lemma 9 *Assume G contains no negative-weight loop reachable from s and $d[v] = \delta(s, v)$ holds for any $v \in V$. Then the graph G_π is a shortest paths tree.*

Proof. We follow the definition of the shortest paths tree.

- $\delta(s, v) < \infty$ only for vertices v reachable from s.
  $d[v] < \infty \Leftrightarrow \pi[v] \neq \text{NIL}$.
- Assume there exist 2 different paths from s to v:

$$\begin{aligned} P_1 &= s \rightsquigarrow u \rightsquigarrow x \rightarrow z \rightsquigarrow v \\ P_2 &= s \rightsquigarrow u \rightsquigarrow y \rightarrow z \rightsquigarrow v, \end{aligned}$$

  where $(x, z), (y, z) \in E'$. Then: $x = \pi[z]$ and $y = \pi[z]$
  $\Rightarrow x = y$, a contradiction.
- Let $P = (s \rightsquigarrow v)$ be a path (in G') and $P = v_0, \ldots, v_k$, where $s = v_0$ and $v = v_k$. For $i = 1, \ldots, k$ one has:

$$\begin{aligned} d[v_i] &= \delta(s, v_i) \\ d[v_i] &= d[v_{i-1}] + w(v_{i-1}, v_i). \end{aligned}$$

  $\Rightarrow w(v_{i-1}, v_i) = \delta(s, v_i) - \delta(s, v_{i-1})$ and

$$\begin{aligned} w(P) &= \sum_{i=1}^{k} w(v_{i-1}, v_i) \\ &= \sum_{i=1}^{k} (\delta(s, v_i) - \delta(s, v_{i-1})) \\ &= \delta(s, v_k) - \delta(s, v_0) \\ &= \delta(s, v_k). \end{aligned}$$

  Hence: $w(P) = \delta(s, v_k)$, so P is a shortest path. □

6c. The Bellman-Ford-Algorithm

The Algorithm returns TRUE iff G does not contain a negative-weight cycle that is reachable from s, and runs in $O(|V| \cdot |E|)$ time.

Algorithm 6 BELLMAN-FORD(G, w, s);
INITIALIZE-SS(G, w, s)
for $i := 1$ **to** $|V| - 1$ **do**
  for every $(u, v) \in E$ **do**
    RELAX(u, v, w)
for every $(u, v) \in E$ **do**
  if $(d[v] > d[u] + w(u, v))$ **then**
    return FALSE
return TRUE

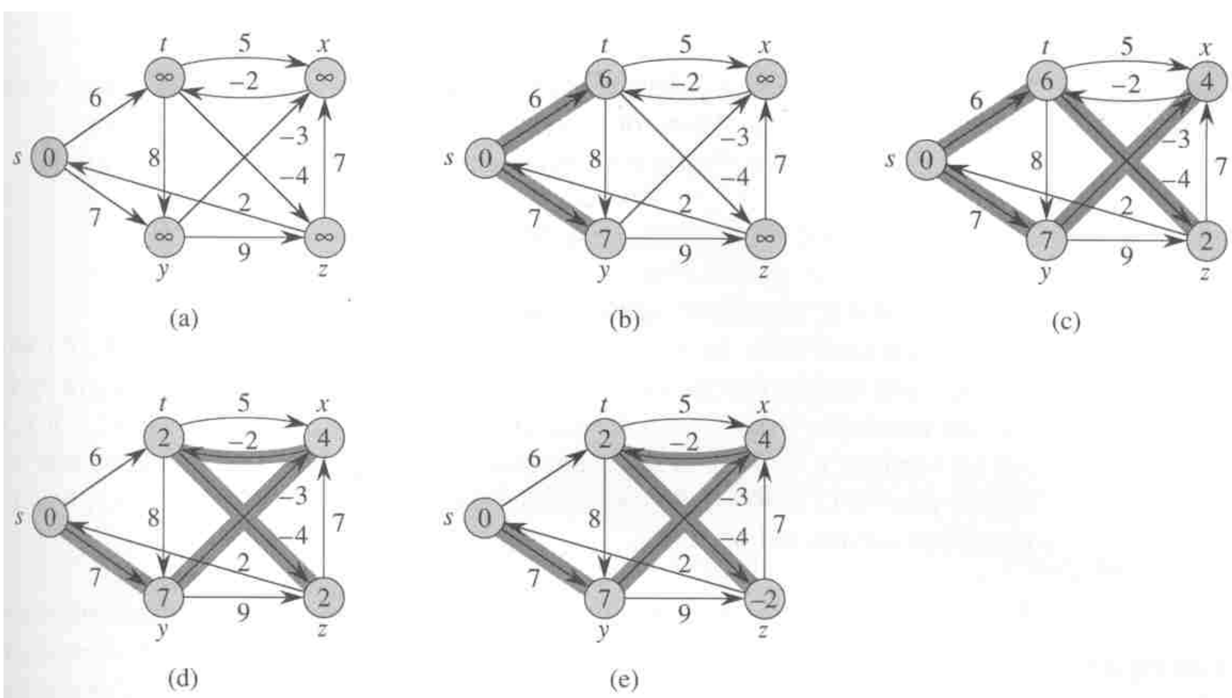

Figure 4: Bellman-Ford Algorithm

Lemma 10 *Assume G does not contain a negative-weight cycle that is reachable from s. Then after $|V|-1$ iterations of the first loop one has: $d[v] = \delta(u,v)$ for any $v \in V$ that is reachable from s.*

Proof. Let $v \in V$ be reachable from s and $P = (v_0, \ldots, v_k)$ be a shortest path $v_0 = s \rightsquigarrow v = v_k$. Then $k \le |V| - 1$.

We show by induction on i that $d[v_i] = \delta(s, v_i)$ after first i iterations of the **for** -loop.

Induction basis: $i = 0$: $d[v_0] = \delta(s, v_0) = 0$.
Induction step: assume $d[v_{i-1}] = \delta(s, v_{i-1})$. Since the edge (v_{i-1}, v_i) is relaxed on the i-th iteration of the loop, the assertion follows from L. 8. □

Corollary 3 *Vertex v is reachable from s iff the* Bellman-Ford *algorithm terminates with $d[v] < \infty$.*

Theorem 5 *If G contains no negative-weight loop that is reachable from s, then the algorithm returns* TRUE *and the shortest paths from s are provided by the pred. subgraph G_π. If G contains such a loop, then the algorithm returns* FALSE.

If G does not contain a negative-weight loop reachable from s, then $d[v] = \delta(s, v)$ follows from L. 10 and its corollary.

The predecessor subgraph G_π is a shortest-path tree (L. 9).

We show that the algorithm returns TRUE. For $(u, v) \in E_G$ one has:

$$\begin{aligned} d[v] &= \delta(s,v) \leq \delta(s,u) + w(u,v) \quad \text{(L. 5a)} \\ &= d[u] + w(u,v). \end{aligned}$$

Hence, the **if** -condition in RELAX is not satisfied for every edge, so the algorithm returns TRUE.

Assume G contains a negative-weight loop $C = (v_0, \ldots, v_k)$ (with $v_0 = v_k$) that is reachable from s. So,

$$\sum_{i=1}^{k} w(v_{i-1}, v_i) < 0.$$

If the algorithm returns TRUE , then

$$d[v_i] \leq d[v_{i-1}] + w(v_{i-1}, v_i), \qquad \text{for } i = 1, \ldots, k.$$

Summing up these inequalities results in:

$$\sum_{i=1}^{k} d[v_i] \leq \sum_{i=1}^{k} d[v_{i-1}] + \sum_{i=1}^{k} w(v_{i-1}, v_i).$$

Since $v_0 = v_k$, all d-values are finite, and each vertex appears in the sums exactly once,

$$\sum_{i=1}^{k} d[v_i] = \sum_{i=1}^{k} d[v_{i-1}]$$

which implies

$$0 \leq \sum_{i=1}^{k} w(v_{i-1}, v_i).$$

This contradiction implies that the algorithm returns FALSE. $\square$

6d. Dijkstra's Algorithm

Assume $w(u, v) \geq 0$ for all $(u, v) \in E_G$.

Algorithm 7 $\textsc{Dijkstra}(G, w, s)$;

$\textsc{Initialize-SS}(G, s)$
$S := \emptyset$; $\quad Q := V(G)$
while $(Q \neq \emptyset)$ **do**
$\quad u := \textsc{Extract-Min}(Q)$
$\quad S := S \cup \{u\}$
$\quad$ **for** each $v \in Adj[u]$
$\quad\quad \textsc{Relax}(u, v, w)$

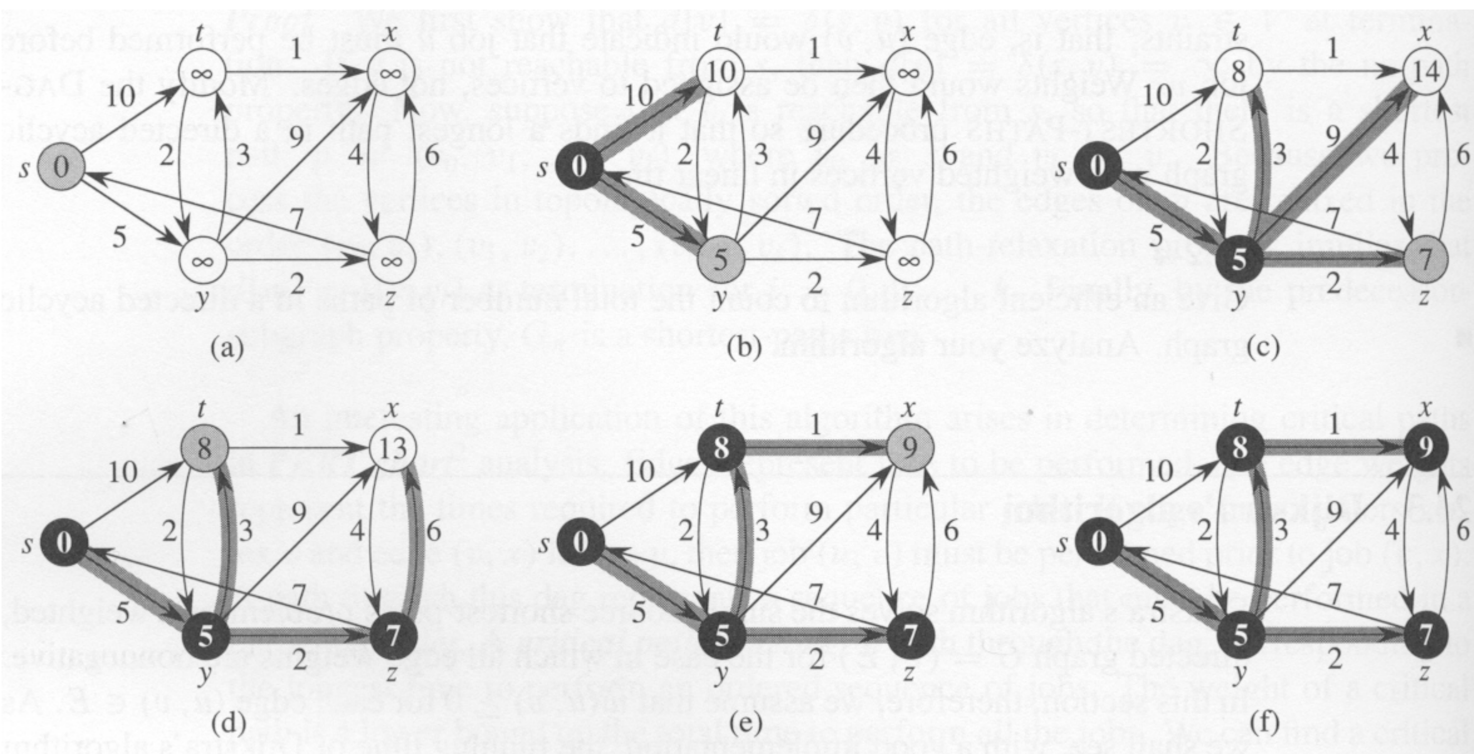

Figure 5: Dijkstra's Algorithm

The running time of $\textsc{Dijkstra}$ Algorithm is $O(|V_G|^2)$. With a careful implementation it can run in $O(|V|\log|V| + |E|)$ time.

Theorem 6 *Let $G = (V_G, E_G)$ be a graph with non-negative edge weights w. Then the* DIJKSTRA *Algorithm provides $d[u] = \delta(s,u)$ $\forall u \in V$.*

Proof:
We show that at time when u is included in S it holds: $d[u] = \delta(s,u)$.

Assume $\exists u$ such that $d[u] > \delta(s,u)$ and let u be the first such vertex. Then u is reachable from s and $u \neq s$.

Let $P = (s \overset{G}{\rightsquigarrow} u)$ be a shortest path and $(x,y) \in E_G$ be the first edge of P with $x \in S$, $y \notin S$. Then $P = (s \rightsquigarrow x \rightarrow y \rightsquigarrow u)$, $d[x] = \delta(s,x)$ and $d[y] = \delta(s,y)$ (by L. 8). Therefore,

$$d[y] = \delta(s,y) \leq \delta(s,u) \leq d[u].$$

However, since $y \in V - S$ when u was chosen: $d[u] \leq d[y]$.
$\Rightarrow d[u] = \delta(s,u)$, a contradiction. □

Since predecessor subgraph G_π is a shortest-path tree (L. 9), the last theorem proves the correctness of Dijkstra's algorithm.

Part II. All pairs shortest paths

Since the running time of DIJKSTRA algorithm is $O(|V|^2)$, one can construct all shortest paths in $O(|V|^3)$ time if the weight function is non-negative.

If G contains no cycle of negative weight, the BELLMAN-FORD algorithm constructs a solution in $O(|V|^4)$ time.

We will develop a better algorithm.

We represent a graph $G = (V_G, E_G)$ with the vertex set $V = \{v_1, \ldots, v_n\}$ by its ajacency matrix w_{ij}, where

$$w_{ij} = \begin{cases} 0, & \text{if } i = j \\ w(v_i, v_j), & \text{if } i \neq j \text{ and } (v_i, v_j) \in E \\ \infty, & \text{if } i \neq j \text{ and } (v_i, v_j) \notin E \end{cases}$$

The shortest paths will be defined by the matrix of predecessors

$$\Pi = \{\pi_{ij}\}.$$

We assume that G contains no cycle of negative weight.

7. The Floyd-Warshall algorithm

Let $P = (v_1, \ldots, v_l)$ be a shortest path $v_1 \rightsquigarrow v_l$. We call the vertices $v_2, \ldots, v_{l-1}$ (if they exist) <u>inner nodes</u> of the path P.

Denote the vertices of G by $\{1, 2, \ldots, n\}$. For $i, j \in V$ and given k consider shortest paths $i \rightsquigarrow j$ with the inner nodes belonging to the set $\{1, \ldots, k\}$. Let P be such a path (if it exists).

- If $k \notin P$ then all the inner nodes of $i \rightsquigarrow j$ are taken from the set $\{1, \ldots, k-1\}$. So P is also a shortest path with the inner nodes of the set $\{1, \ldots, k-1\}$.
- If $k \in P$ then split P into two paths: $P_1 = (i \to k)$ and $P_2 = (k \to j)$. Then P_1 is a shortest path $i \rightsquigarrow k$ with all inner nodes of the set $\{1, \ldots, k-1\}$, and the same holds for P_2.

Denote by d_{ij}^k the weight of the shortest path $i \rightsquigarrow j$ with all inner nodes of the set $\{1, \ldots, k\}$. One has:

$$d_{ij}^k = \begin{cases} w_{ij}, & \text{if } k = 0 \\ \min\left\{d_{ij}^{k-1},\ d_{ik}^{k-1} + d_{kj}^{k-1}\right\}, & \text{if } k \geq 1. \end{cases}$$

We put these numbers into a matrix $D^k = \{d_{ij}^k\}$, where $d_{ij}^n = \delta(i, j)$ for $1 \leq i \leq j \leq n$.

Algorithm 8 FLOYD-WARSHALL(W)

$n := \#rows(W)$
$D^0 := W$
for $k := 1$ **to** n **do**
  for $i := 1$ **to** n **do**
    for $j := 1$ **to** n **do**
      $d_{ij}^k := \min\{d_{ij}^{k-1},\, d_{ik}^{k-1} + d_{kj}^{k-1}\}$
return D^n

The running time of FLOYD-WARSHALL algorithm is $O(n^3)$.

Construction of shortest paths

We construct a series of matrices: $\Pi^0, \ldots, \Pi^n$ with $\Pi^k = \{\pi_{ij}^k\}$, where π_{ij}^k is the predecessor of j on a shortest path $i \rightsquigarrow j$ with all inner nodes of the set $\{1, \ldots, k\}$.

$$\pi_{ij}^0 = \begin{cases} \text{NIL}, & \text{if } i = j \text{ or } w_{ij} = \infty \\ i, & \text{if } i \neq j \text{ and } w_{ij} < \infty. \end{cases}$$

For $k \geq 1$ define:

$$\pi_{ij}^k = \begin{cases} \pi_{ij}^{k-1}, & \text{if } d_{ij}^{k-1} \leq d_{ik}^{k-1} + d_{kj}^{k-1} \\ \pi_{kj}^{k-1}, & \text{if } d_{ij}^{k-1} > d_{ik}^{k-1} + d_{kj}^{k-1}. \end{cases}$$

The elements of Π^n provide for each vertex j its predecessor π_{ij}^n on a shortest path $i \rightsquigarrow j$.

$$D^{(0)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(0)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & \text{NIL} & 4 & \text{NIL} & \text{NIL} \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(1)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(2)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(3)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(3)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(4)} = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad \Pi^{(4)} = \begin{pmatrix} \text{NIL} & 1 & 4 & 2 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(5)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad \Pi^{(5)} = \begin{pmatrix} \text{NIL} & 3 & 4 & 5 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

Figure 6: Floyd-Warshall algorithm

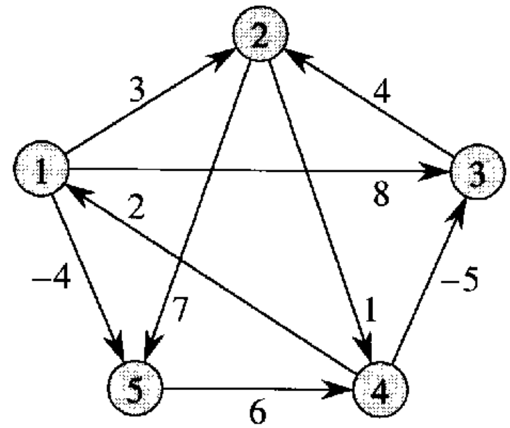

Figure 7: Example graph for the Floyd-Warshall algorithm

$$D^{(0)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(0)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & \text{NIL} & 4 & \text{NIL} & \text{NIL} \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(1)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(2)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(3)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(3)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(4)} = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad \Pi^{(4)} = \begin{pmatrix} \text{NIL} & 1 & 4 & 2 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(5)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad \Pi^{(5)} = \begin{pmatrix} \text{NIL} & 3 & 4 & 5 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

Figure 8: The Floyd-Warshall algorithm again

8. Graph Colorings

Definition 4 *A* coloring *an assignment of colors to vertices such that no two adjacent nodes carry the same color.*

A k-coloring is a coloring that uses k different colors $\{1, 2, \ldots, k\}$.

The chromatic number $\chi(G)$ *of a graph G is the smaller k for which G admits a k-coloring.*

A coloring that uses exactly $\chi(G)$ colors is called minimal.

It holds:

$$\chi(K_n) = n, \qquad \chi(C_{2n}) = 2, \qquad \chi(C_{2n+1}) = 3.$$

Theorem 7 *A graph G is 2-colorable iff it has no loop of an odd length.*

Sketch of proof:
"$\Longrightarrow$" Obvious.

"$\Longleftarrow$" The following algorithm devilers a 2-coloring for G if one exists and returns FALSE otherwise.

Let Q be (a FIFO)-Queue.

Algorithm 9 BIPARTITE(G);

Choose any node $v \in V$ and color it with 1
$Q := \{v\}$
repeat **while** $Q \neq \emptyset$
  $u := head(Q)$
  $S := Adj[u]$
   for all $w \in S$
    do if $color[w] = color[u]$
      then Graph is not bipartite. FALSE
  Color every uncolored node w in S with color $3 - color[w]$
   and add it to Q.
  $Q := Q - \{u\}$
return 2-coloring

Let $\omega(G)$ be the size of a maximum clique in G.

Theorem 8 *It holds:*

$$\omega(G) \leq \chi(G) \leq \Delta(G) + 1.$$

The lower bound is obvious. The following algorithm constructs a coloring satisfying the upper bound.

Algorithm 10 COLORING(G);

Choose $v \in V$ and color it with color 1
$V' := V - \{v\}$
repeat **while** $V' \neq \emptyset$
    Choose $u \in V'$
    $S := Adj[u]$
    Color u with the smallest unused color number in S
    $V' := V' - \{u\}$
return largest used color number

Remark 1 *For any two numbers Δ, k with $2 \leq k \leq \Delta$ there exists a G with maximum degree Δ and $\chi(G) = k$.*

A general method for computing $\chi(G)$:

Definition 5 *Let $G = (V, E)$ be a graph and $a, b \in V$, $(a, b) \notin E$. Define*
$G : ab = (V', E')$, where

$$\begin{aligned} V' &= (V - \{a, b\}) \cup \{z\}, \quad (z \notin V) \\ E' &= (E - \{(x, y) \mid x \in V, y \in \{a, b\}\}) \cup \{(x, z) \mid x \notin \{a, b\}, \\ &\qquad \textit{and either } (x, a) \in E \textit{ or } (x, b) \in E\}. \end{aligned}$$

$G/ab = (V, E'')$, where $E'' = E \cup (a, b)$.

A coloring of G satisfying $color(a) = color(b)$ also provides a coloring of $G : ab$. Similarly, a coloring of G satisfying $color(a) \neq color(b)$ provides a correct coloring of G/ab.

Repeat the above operations until the resulting graph is a clique. If the smallest-size clique consists of k nodes, then $\chi(G) = k$.

This method leads to an exponential running time, in general.

Example:

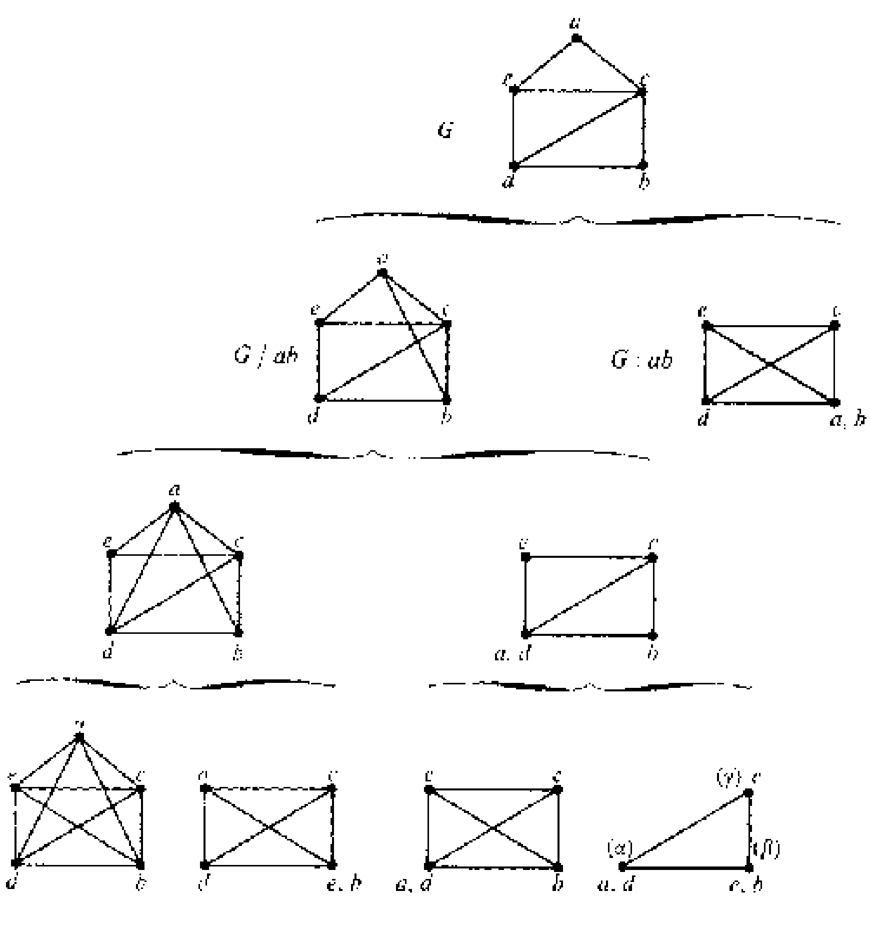

Figure 9: Graph coloring with a DP algorithm

Coloring of special graph classes

Definition 6 *A graph $G = (V, E)$ is called* $\mathrm{interval\ graph}$ *if it can be represented by a set of intervals on a line as the set of nodes. An edge of G only exists between overlapping intervals.*

Theorem 9 *Let G be an interval graph. Then $\chi(G) = \omega(G)$ and a greedy algorithm returns a coloring consisting of $\omega(G)$ colors.*

Definition 7 *A graph $G = (V, E)$ is called* planar *if it can be drawn on a plane so that no two edges have a proper intersection.*

Theorem 10 (Euler)
Let $G = (V, E)$ be a planar connected graph with $|V| = n$, $|E| = e$ and f be the number of its faces. Then:

$$n - e + f = 2.$$

Corollary 4

1. *Let $G = (V, E)$ be a planar graph with $|V| = n$, $|E| = e$. Then:*

$$e \leq 3 \cdot n - 6.$$

2. *Let $G = (V, E)$ be a planar graph with $|V| \geq 4$. Then G has a node of degree ≤ 5.*

Proof:
1. Every face consists of ≥ 3 edges, and every edge belongs to 2 faces. Therefore, counting the number of edges by different ways one has $3f \leq 2e$. Furthermore, using the Euler identity,

$$e = n + f - 2 \leq n + 2e/3 - 2,$$

which implies $e \leq 3 \cdot n - 6$.

2. If the degree of each vertex is at least 6, then: $2e \geq 6n$, which is equivalent to $e \geq 3n$. $\square$

Theorem 11 *Every planar graph is 6-colorable.*

With a more deep analysis one can also prove

Theorem 12 *Every planar graph is 5-colorable.*

A very difficult prove that involves many days of non-stop computing shows

Theorem 13 *Every planar graph is 4-colorable.*

However, not all planar graphs are 3-colorable (e.g. K_4). As we will see later, a problem to determine if G admits a 3-coloring is intractable!