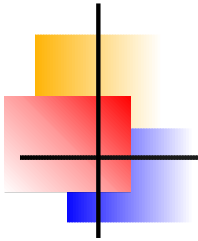
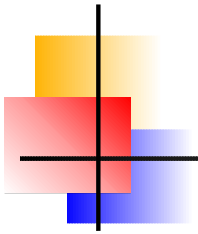


Paths in Graphs

Stanislav Palúch

Fakulta riadenia a informatiky, Žilinská univerzita

18. mája 2016



Walk, trail, path

Definition

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

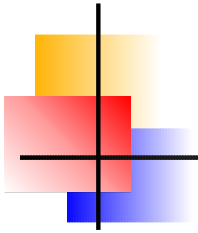
A walk (a v_1–v_k walk) *in graph G is arbitrary alternating sequence of vertices and edges in the form:*

$$\mu(v_1, v_k) = (v_1, \{v_1, v_2\}, v_2, \{v_2, v_3\}, v_3, \ldots, \{v_{k-1}, v_k\}, v_k). \qquad (1)$$

A trail (a v_1–v_k trail) *in graph G is a v_1–v_k walk in graph G with no repeated edges.*

A path (a v_1–v_k path) *in graph G is a v_1–v_k walk in graph G with no repeated vertices.*

We allow so called **trivial walk**, *having only one vertex i.e. $k = 1$, i.e. walk in form (v_1).*



Walk, trail, path

Definition

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

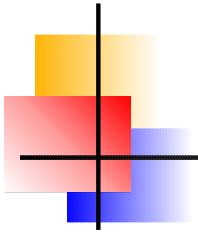
A walk (a v_1–v_k walk) *in graph G is arbitrary alternating sequence of vertices and edges in the form:*

$$\mu(v_1, v_k) = (v_1, \{v_1, v_2\}, v_2, \{v_2, v_3\}, v_3, \ldots, \{v_{k-1}, v_k\}, v_k). \qquad (1)$$

A trail (a v_1–v_k trail) *in graph G is a v_1–v_k walk in graph G with no repeated edges.*

A path (a v_1–v_k path) *in graph G is a v_1–v_k walk in graph G with no repeated vertices.*

We allow so called **trivial walk**, *having only one vertex i.e. $k = 1$, i.e. walk in form (v_1).*



Walk, trail, path

Definition

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

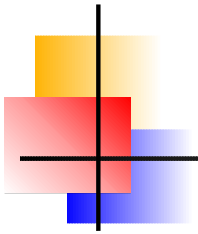
A walk (a v_1–v_k walk) *in graph G is arbitrary alternating sequence of vertices and edges in the form:*

$$\mu(v_1, v_k) = (v_1, \{v_1, v_2\}, v_2, \{v_2, v_3\}, v_3, \ldots, \{v_{k-1}, v_k\}, v_k). \qquad (1)$$

A trail (a v_1–v_k trail) *in graph G is a v_1–v_k walk in graph G with no repeated edges.*

A path (a v_1–v_k path) *in graph G is a v_1–v_k walk in graph G with no repeated vertices.*

We allow so called **trivial walk**, *having only one vertex i.e. $k = 1$, i.e. walk in form (v_1).*



Walk, trail, path

Definition

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

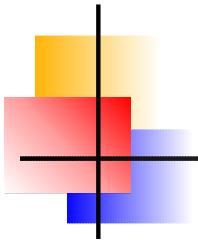
A walk (a v_1–v_k walk) *in graph G is arbitrary alternating sequence of vertices and edges in the form:*

$$\mu(v_1, v_k) = (v_1, \{v_1, v_2\}, v_2, \{v_2, v_3\}, v_3, \ldots, \{v_{k-1}, v_k\}, v_k). \qquad (1)$$

A trail (a v_1–v_k trail) *in graph G is a v_1–v_k walk in graph G with no repeated edges.*

A path (a v_1–v_k path) *in graph G is a v_1–v_k walk in graph G with no repeated vertices.*

We allow so called **trivial walk**, *having only one vertex i.e. $k = 1$, i.e. walk in form (v_1).*



Directed walk, directed trail, directed path

Definition

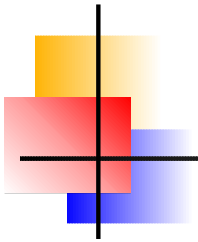
Let $\overrightarrow{G} = (V, H)$ *be a digraph.*

A directed walk (a directed v_1–v_k *walk)** *in digraph* $\overrightarrow{G}$ *is arbitrary alternating sequence of vertices and arcs in the form:*

$$\mu(v_1, v_k) = (v_1, (v_1, v_2), v_2, (v_2, v_3), v_3, \ldots, (v_{k-1}, v_k), v_k). \qquad (2)$$

A directed trail *in digraph* $\overrightarrow{G}$ *is a directed* v_1–v_k *walk in* $\overrightarrow{G}$ *with no repeated arcs.*

A directed path *in digraph* $\overrightarrow{G}$ *is a directed* v_1–v_k *swalk in* $\overrightarrow{G}$*, with no repeated vertices.*



Directed walk, directed trail, directed path

Definition

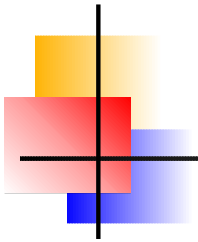
Let $\overrightarrow{G} = (V, H)$ *be a digraph.*

A directed walk (a directed v_1–v_k *walk)** *in digraph* $\overrightarrow{G}$ *is arbitrary alternating sequence of vertices and arcs in the form:*

$$\mu(v_1, v_k) = (v_1, (v_1, v_2), v_2, (v_2, v_3), v_3, \ldots, (v_{k-1}, v_k), v_k). \qquad (2)$$

A directed trail *in digraph* $\overrightarrow{G}$ *is a directed* v_1–v_k *walk in* $\overrightarrow{G}$ *with no repeated arcs.*

A directed path *in digraph* $\overrightarrow{G}$ *is a directed* v_1–v_k *swalk in* $\overrightarrow{G}$*, with no repeated vertices.*



Directed walk, directed trail, directed path

Definition

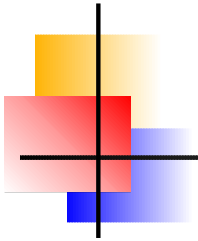
Let $\overrightarrow{G} = (V, H)$ *be a digraph.*

A directed walk (a directed v_1–v_k *walk***)** *in digraph* $\overrightarrow{G}$ *is arbitrary alternating sequence of vertices and arcs in the form:*

$$\mu(v_1, v_k) = (v_1, (v_1, v_2), v_2, (v_2, v_3), v_3, \ldots, (v_{k-1}, v_k), v_k). \qquad (2)$$

A directed trail *in digraph* $\overrightarrow{G}$ *is a directed* v_1–v_k *walk in* $\overrightarrow{G}$ *with no repeated arcs.*

A directed path *in digraph* $\overrightarrow{G}$ *is a directed* v_1–v_k *swalk in* $\overrightarrow{G}$*, with no repeated vertices.*



Quasi-walk, quasi-trail, quasi-path in a digraph

Definition

Let $\overrightarrow{G} = (V, H)$ be a digraph.

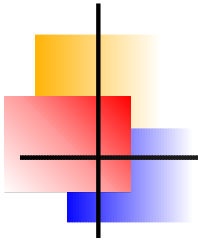
A quasi-walk (a v_1–v_k quasi-walk) *in digraph $\overrightarrow{G}$ is arbitrary alternating sequence of vertices and arcs in the form:*

$$\mu(v_1, v_k) = (v_1, h_1, v_2, h_2, \ldots, v_{k-1}, h_{k-1}, v_k),$$

in which every arc h_i is incident with both vertices v_i, v_{i+1} such that one of them is a head and one of the is the tail of the arc h_i.

A quasi-trail (a v_1–v_k quasi-trail) *in digraph $\overrightarrow{G}$ is a v_1–v_k quasi-walk in $\overrightarrow{G}$ with no repeated arcs.*

A quasi-path (a v_1–v_k quasi-path) *in digraph $\overrightarrow{G}$ is a v_1–v_k quasi-walk in $\overrightarrow{G}$ with no repeated vertices.*



Quasi-walk, quasi-trail, quasi-path in a digraph

Definition

Let $\overrightarrow{G} = (V, H)$ be a digraph.

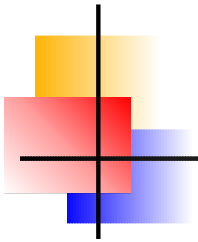
A quasi-walk (a v_1–v_k quasi-walk) *in digraph $\overrightarrow{G}$ is arbitrary alternating sequence of vertices and arcs in the form:*

$$\mu(v_1, v_k) = (v_1, h_1, v_2, h_2, \ldots, v_{k-1}, h_{k-1}, v_k),$$

in which every arc h_i is incident with both vertices v_i, v_{i+1} such that one of them is a head and one of the is the tail of the arc h_i.

A quasi-trail (a v_1–v_k quasi-trail) *in digraph $\overrightarrow{G}$ is a v_1–v_k quasi-walk in $\overrightarrow{G}$ with no repeated arcs.*

A quasi-path (a v_1–v_k quasi-path) *in digraph $\overrightarrow{G}$ is a v_1–v_k quasi-walk in $\overrightarrow{G}$ with no repeated vertices.*



Quasi-walk, quasi-trail, quasi-path in a digraph

Definition

Let $\overrightarrow{G} = (V, H)$ be a digraph.

A quasi-walk (a v_1–v_k quasi-walk) *in digraph $\overrightarrow{G}$ is arbitrary alternating sequence of vertices and arcs in the form:*

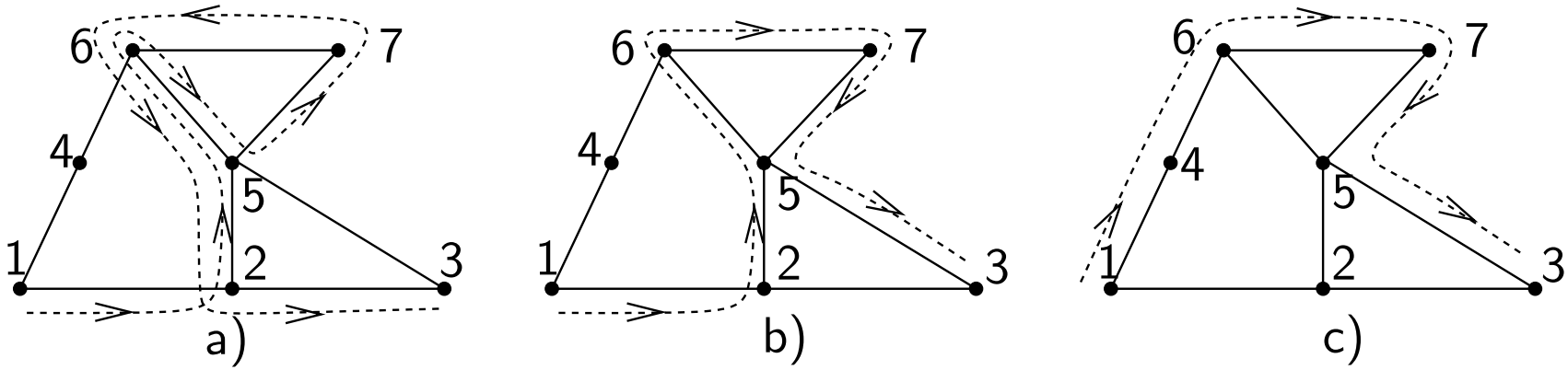
$$\mu(v_1, v_k) = (v_1, h_1, v_2, h_2, \ldots, v_{k-1}, h_{k-1}, v_k),$$

in which every arc h_i is incident with both vertices v_i, v_{i+1} such that one of them is a head and one of the is the tail of the arc h_i.

A quasi-trail (a v_1–v_k quasi-trail) *in digraph $\overrightarrow{G}$ is a v_1–v_k quasi-walk in $\overrightarrow{G}$ with no repeated arcs.*

A quasi-path (a v_1–v_k quasi-path) *in digraph $\overrightarrow{G}$ is a v_1–v_k quasi-walk in $\overrightarrow{G}$ with no repeated vertices.*

Examples



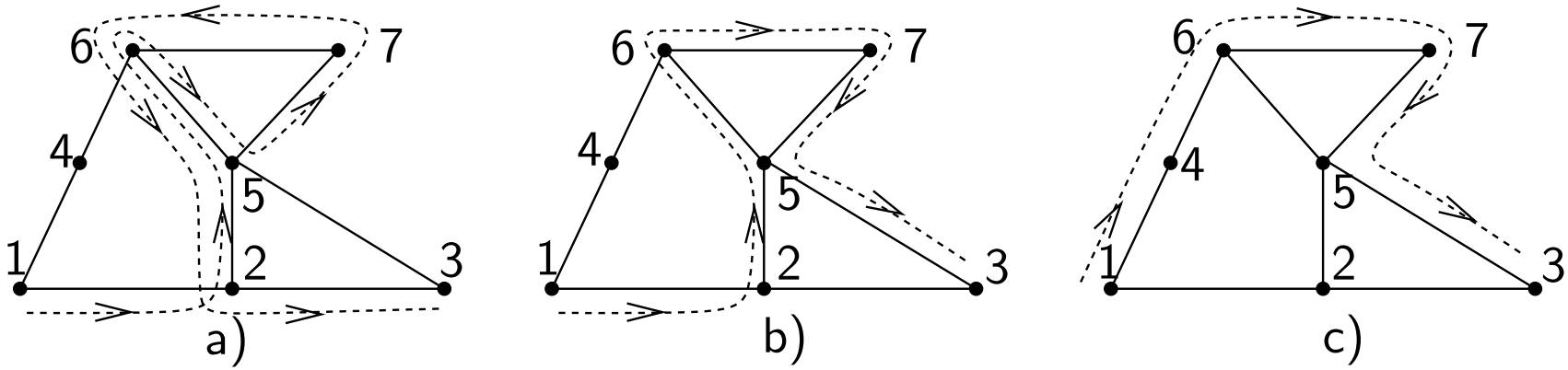
Obr.: Walk, trail an path in a graph.

a) 1–3 walk: $(1, \{1,2\}, 2, \{2,5\}, 5,$
$\{5,6\}, 6, \{6,5\}, 5, \{5,7\}, 7, \{7,6\}, 6, \{6,5\}, 5, \{5,2\}, 2, \{2,3\}, 3)$.

b) 1–3 trail: $(1, \{1,2\}, 2, \{2,5\}, 5, \{5,6\}, 6, \{6,7\}, 7, \{7,5\}, 5, \{5,3\}, 3)$.

c) 1–3 path: $(1, \{1,4\}, 4, \{4,6\}, 6, \{6,7\}, 7, \{7,5\}, 5, \{5,3\}, 3)$.

Examples



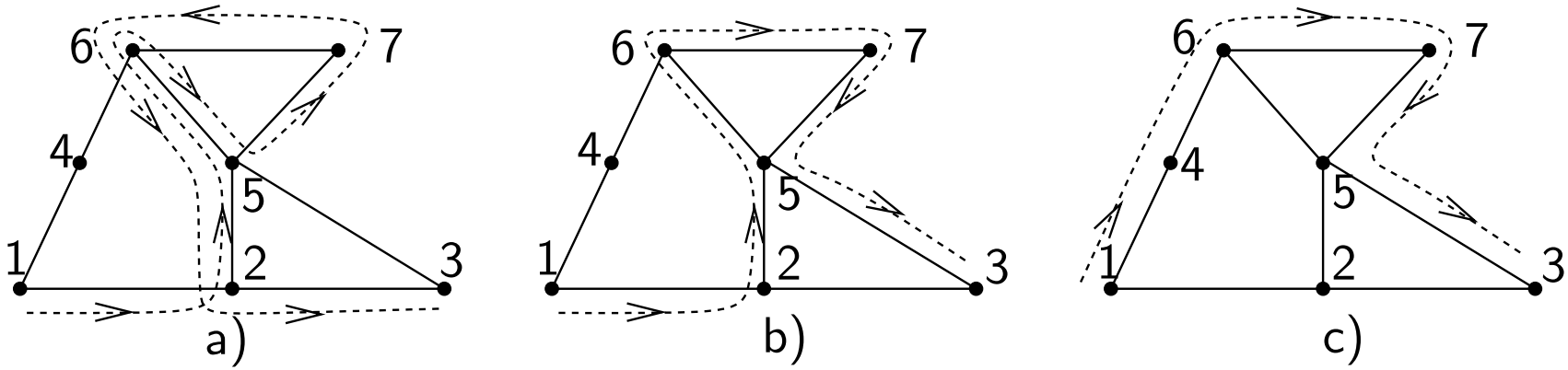
Obr.: Walk, trail an path in a graph.

a) 1–3 walk: $(1, \{1,2\}, 2, \{2,5\}, 5, \{5,6\}, 6, \{6,5\}, 5, \{5,7\}, 7, \{7,6\}, 6, \{6,5\}, 5, \{5,2\}, 2, \{2,3\}, 3)$.

b) 1–3 trail: $(1, \{1,2\}, 2, \{2,5\}, 5, \{5,6\}, 6, \{6,7\}, 7, \{7,5\}, 5, \{5,3\}, 3)$.

c) 1–3 path: $(1, \{1,4\}, 4, \{4,6\}, 6, \{6,7\}, 7, \{7,5\}, 5, \{5,3\}, 3)$.

Examples



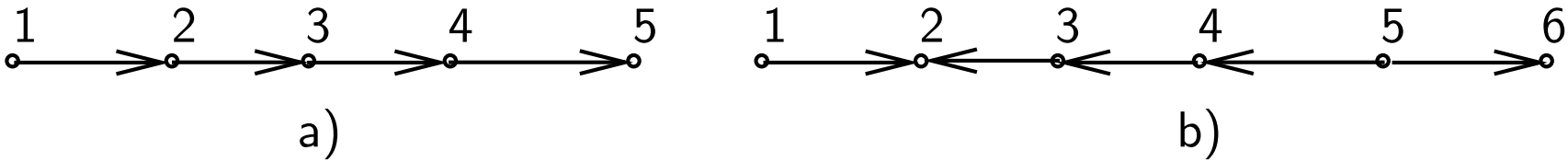
Obr.: Walk, trail an path in a graph.

a) 1–3 walk: $(1, \{1,2\}, 2, \{2,5\}, 5, \{5,6\}, 6, \{6,5\}, 5, \{5,7\}, 7, \{7,6\}, 6, \{6,5\}, 5, \{5,2\}, 2, \{2,3\}, 3)$.

b) 1–3 trail: $(1, \{1,2\}, 2, \{2,5\}, 5, \{5,6\}, 6, \{6,7\}, 7, \{7,5\}, 5, \{5,3\}, 3)$.

c) 1–3 path: $(1, \{1,4\}, 4, \{4,6\}, 6, \{6,7\}, 7, \{7,5\}, 5, \{5,3\}, 3)$.

Examples



Obr.: Directed path and quasi-path in a digraph.

a) 1–5 directed paths: $(1, (1,2), 2, (2,3), 3, (3,4), 4, (4,5), 5)$.
b) 1–6 quasi-path: $(1, (1,2), 2, (3,2), 3, (4,3), 4, (4,5), 5, (5,6), 6)$.

We allow to write instead of

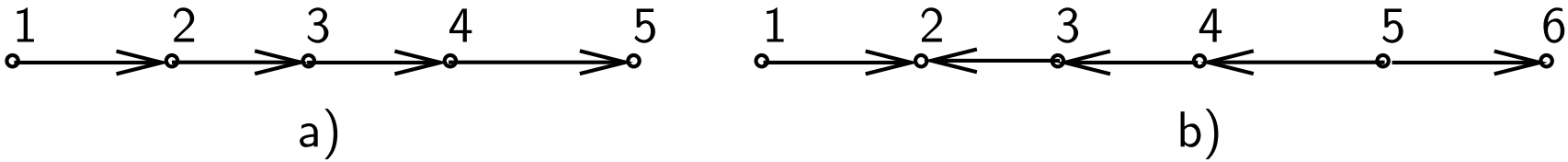
$$\mu(v_1, v_k) = (v_1, h_1, v_2, h_2, \ldots, v_{k-1}, h_{k-1}, v_k),$$

a shortened notation in the form:

$$\mu(v_1, v_k) = (v_1, v_2, \ldots, v_k),$$

namely in cases when it can not come to misunderstanding.

Examples



Obr.: Directed path and quasi-path in a digraph.

a) 1–5 directed paths: $(1, (1,2), 2, (2,3), 3, (3,4), 4, (4,5), 5)$.

b) 1–6 quasi-path: $(1, (1,2), 2, (3,2), 3, (4,3), 4, (4,5), 5, (5,6), 6)$.

We allow to write instead of

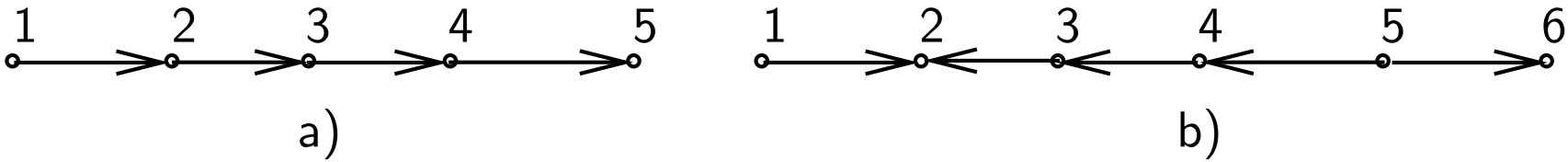
$$\mu(v_1, v_k) = (v_1, h_1, v_2, h_2, \ldots, v_{k-1}, h_{k-1}, v_k),$$

a shortened notation in the form:

$$\mu(v_1, v_k) = (v_1, v_2, \ldots, v_k),$$

namely in cases when it can not come to misunderstanding.

Examples



Obr.: Directed path and quasi-path in a digraph.

a) 1–5 directed paths: $(1, (1,2), 2, (2,3), 3, (3,4), 4, (4,5), 5)$.

b) 1–6 quasi-path: $(1, (1,2), 2, (3,2), 3, (4,3), 4, (4,5), 5, (5,6), 6)$.

We allow to write instead of

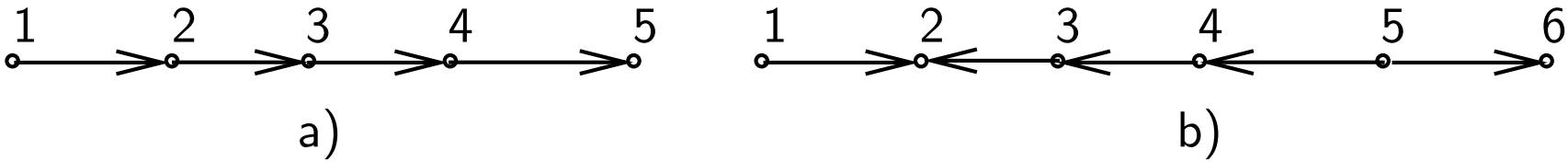
$$\mu(v_1, v_k) = (v_1, h_1, v_2, h_2, \ldots, v_{k-1}, h_{k-1}, v_k),$$

a shortened notation in the form:

$$\mu(v_1, v_k) = (v_1, v_2, \ldots, v_k),$$

namely in cases when it can not come to misunderstanding.

Examples



Obr.: Directed path and quasi-path in a digraph.

a) 1–5 directed paths: $(1, (1,2), 2, (2,3), 3, (3,4), 4, (4,5), 5)$.
b) 1–6 quasi-path: $(1, (1,2), 2, (3,2), 3, (4,3), 4, (4,5), 5, (5,6), 6)$.

We allow to write instead of

$$\boldsymbol{\mu}(v_1, v_k) = (v_1, h_1, v_2, h_2, \ldots, v_{k-1}, h_{k-1}, v_k),$$

a shortened notation in the form:

$$\boldsymbol{\mu}(v_1, v_k) = (v_1, v_2, \ldots, v_k),$$

namely in cases when it can not come to misunderstanding.

Closed walk

Definition

A walk (quasi-walk, trail, quasi-walk)

$$\mu(v_1, v_k) = (v_1, h_1, v_2, h_2, \ldots, v_{k-1}, h_{k-1}, v_k)$$

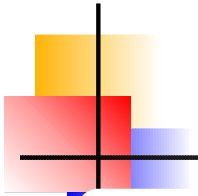
is **closed**, *if* $v_1 = v_k$.
Otherwise, a walk (quasi-walk, trail, quasi-walk) $\mu(v_1, v_k)$ *is* **open**.

Remark

Closed path can not be defined similarly since a path must not contain repeated vertices.
We have a cycle instead of closed path.

Definition

Cycle (directed cycle, quasi-cycle) *is non trivial closed walk (directed walk, quasi-walk) in which every vertex but the first and the last appears at most once.*



Closed walk

Definition

A walk (quasi-walk, trail, quasi-walk)

$$\boldsymbol{\mu}(v_1, v_k) = (v_1, h_1, v_2, h_2, \ldots, v_{k-1}, h_{k-1}, v_k)$$

is **closed**, *if* $v_1 = v_k$.
Otherwise, a walk (quasi-walk, trail, quasi-walk) $\boldsymbol{\mu}(v_1, v_k)$ *is* **open**.

Remark

Closed path can not be defined similarly since a path must not contain repeated vertices.
We have a cycle instead of closed path.

Definition

Cycle (directed cycle, quasi-cycle) *is non trivial closed walk (directed walk, quasi-walk) in which every vertex but the first and the last appears at most once.*

Concatenation of walks

Definition

Let

$$\mu(v_1, v_r) = (v_1, \{v_1, v_2\}, v_2, \{v_2, v_3\}, v_3, \ldots, \{v_{r-1}, v_r\}, v_r),$$

$$\mu(w_1, w_s) = (w_1, \{w_1, w_2\}, w_2, \{w_2, w_3\}, w_3, \ldots, \{w_{s-1}, w_s\}, w_s),$$

let $v_r = w_1$.

Concatenation of walks $\mu(v_1, v_r)$, $\mu(w_1, w_s)$ *is the walk*

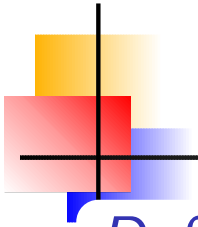
$$\mu(v_1, v_r) \oplus \mu(w_1, w_s) =$$

$$= (v_1, \{v_1, v_2\}, v_2, \ldots, \{v_{r-1}, v_r\}, v_r = w_1, \{w_1, w_2\}, w_2, \ldots, \{w_{s-1}, w_s\}, w_s).$$

Concatenation of directed walks is defined by the same way.

Remark

Concatenation $\mu(u, w) \oplus \mu(w, v)$ *of two paths* $\mu(u, w)$ $\mu(w, v)$ *need not be a paths – the result can be a walk.*



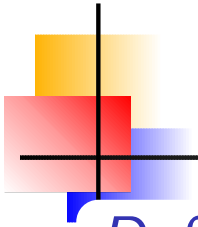
Reachability

Definition

Let $G = (V, H)$ is a graph, resp. digraph, let $u,\ v \in V$.
We say that the vertex v is je **reachable** *from the vertex u in a graph, resp. digraph G, if there exist a u–v walk in graph G, resp. if there exists a directed u–v walk in digraph G.*

Theorem

If there exists a u–v walk in graph $G = (V, H)$ then there exists a u–v path in G.



Reachability

Definition

Let $G = (V, H)$ is a graph, resp. digraph, let $u,\ v \in V$.
We say that the vertex v is je **reachable** *from the vertex u in a graph, resp. digraph G, if there exist a u–v walk in graph G, resp. if there exists a directed u–v walk in digraph G.*

Theorem

If there exists a u–v walk in graph $G = (V, H)$ then there exists a u–v path in G.

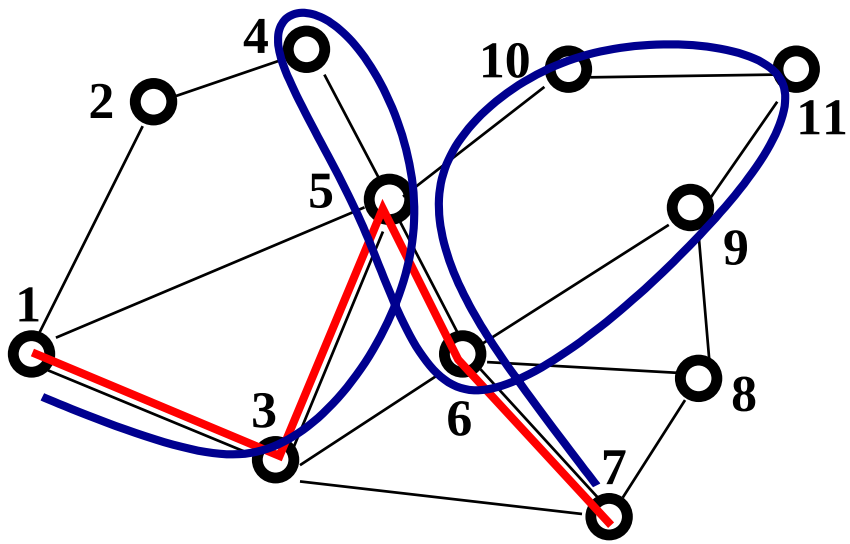
Reachability

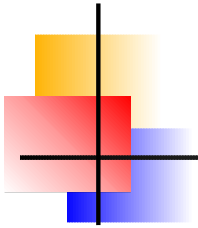
Definition

Let $G = (V, H)$ is a graph, resp. digraph, let $u,\ v \in V$.
We say that the vertex v is je **reachable** *from the vertex u in a graph, resp. digraph G, if there exist a u–v walk in graph G, resp. if there exists a directed u–v walk in digraph G.*

Theorem

If there exists a u–v walk in graph $G = (V, H)$ then there exists a u–v path in G.





Connectivity

Definition

We say that graph $G = (V, H)$ is **connected**, *if there exists a u–v path for every pair of vertices $u, v \in V$. Otherwise, the graph G is said to be* **disonnected**.

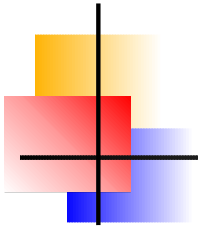
Definition

A component of a graph $G = (V, H)$ *is a maximal connected subgraph of the graph G.*

Definition

A bridge *in graph $G = (V, H)$ is such an adge removing of it results in increasing of the number of components of G.*

An articulation *in graph G is such a vertex removing of it (together with adjacent edges) results in increasing of the number of components of G.*



Connectivity

Definition

We say that graph $G = (V, H)$ is **connected**, *if there exists a u–v path for every pair of vertices $u, v \in V$. Otherwise, the graph G is said to be* **disonnected**.

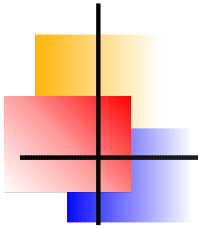
Definition

A component of a graph $G = (V, H)$ *is a maximal connected subgraph of the graph G.*

Definition

A bridge *in graph $G = (V, H)$ is such an adge removing of it results in increasing of the number of components of G.*

An articulation *in graph G is such a vertex removing of it (together with adjacent edges) results in increasing of the number of components of G.*



Connectivity

Definition

We say that graph $G = (V, H)$ is **connected**, *if there exists a u–v path for every pair of vertices $u, v \in V$. Otherwise, the graph G is said to be* **disonnected**.

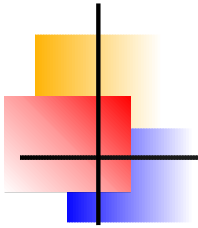
Definition

A component of a graph $G = (V, H)$ *is a maximal connected subgraph of the graph G.*

Definition

A bridge *in graph $G = (V, H)$ is such an adge removing of it results in increasing of the number of components of G.*

An articulation *in graph G is such a vertex removing of it (together with adjacent edges) results in increasing of the number of components of G.*



Connectivity

Definition

We say that graph $G = (V, H)$ is **connected**, *if there exists a u–v path for every pair of vertices $u, v \in V$. Otherwise, the graph G is said to be* **disonnected**.

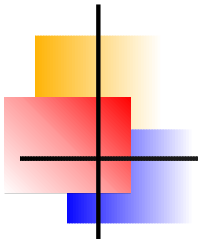
Definition

A component of a graph $G = (V, H)$ *is a maximal connected subgraph of the graph G.*

Definition

A bridge *in graph $G = (V, H)$ is such an adge removing of it results in increasing of the number of components of G.*

An articulation *in graph G is such a vertex removing of it (together with adjacent edges) results in increasing of the number of components of G.*



Connectivity of digraphs, component of a digraph

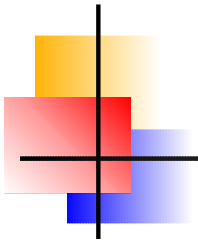
Definition

Let $\overrightarrow{G} = (V, H)$ be a digraph.
We will say that digraph $\overrightarrow{G}$ is **weakly connected**, *if there exists a u–v quasi-path in $\overrightarrow{G}$ for every pair of vertices $u, v \in V$.*
Otherwise we will say that digraph is $\overrightarrow{G}$ **disconnected**.

We will say that digraph $\overrightarrow{G}$ is **unilaterally connected** *if there exists a directed u–v path or a directed v–u path for every pair of vertices $u, v \in V$.*

Digraph $\overrightarrow{G}$ is **strongly connected**, *if there exist both a directed u–v path and a directed v–u path for every pair of vertices $u, v \in V$.*

A component of digraph $\overrightarrow{G}$ *is a maximal weakly connected subgraph of $\overrightarrow{G}$.*



Connectivity of digraphs, component of a digraph

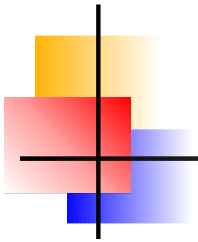
Definition

Let $\overrightarrow{G} = (V, H)$ be a digraph.
We will say that digraph $\overrightarrow{G}$ is **weakly connected**, *if there exists a u–v quasi-path in $\overrightarrow{G}$ for every pair of vertices $u, v \in V$.*
Otherwise we will say that digraph is $\overrightarrow{G}$ **disconnected**.

We will say that digraph $\overrightarrow{G}$ is **unilaterally connected** *if there exists a directed u–v path or a directed v–u path for every pair of vertices $u, v \in V$.*

Digraph $\overrightarrow{G}$ is **strongly connected**, *if there exist both a directed u–v path and a directed v–u path for every pair of vertices $u, v \in V$.*

A component of digraph $\overrightarrow{G}$ *is a maximal weakly connected subgraph of $\overrightarrow{G}$.*



Connectivity of digraphs, component of a digraph

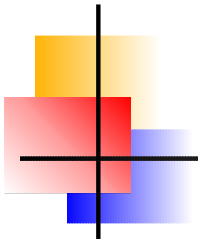
Definition

Let $\overrightarrow{G} = (V, H)$ be a digraph.
We will say that digraph $\overrightarrow{G}$ is **weakly connected**, *if there exists a u–v quasi-path in $\overrightarrow{G}$ for every pair of vertices $u, v \in V$.*
Otherwise we will say that digraph is $\overrightarrow{G}$ **disconnected**.

We will say that digraph $\overrightarrow{G}$ is **unilaterally connected** *if there exists a directed u–v path or a directed v–u path for every pair of vertices $u, v \in V$.*

Digraph $\overrightarrow{G}$ is **strongly connected**, *if there exist both a directed u–v path and a directed v–u path for every pair of vertices $u, v \in V$.*

A component of digraph $\overrightarrow{G}$ *is a maximal weakly connected subgraph of $\overrightarrow{G}$.*



Connectivity of digraphs, component of a digraph

Definition

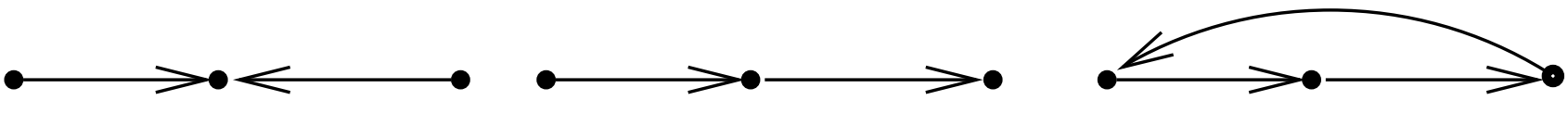
Let $\overrightarrow{G} = (V, H)$ be a digraph.
We will say that digraph $\overrightarrow{G}$ is **weakly connected**, *if there exists a u–v quasi-path in $\overrightarrow{G}$ for every pair of vertices $u, v \in V$.*
Otherwise we will say that digraph is $\overrightarrow{G}$ **disconnected**.

We will say that digraph $\overrightarrow{G}$ is **unilaterally connected** *if there exists a directed u–v path or a directed v–u path for every pair of vertices $u, v \in V$.*

Digraph $\overrightarrow{G}$ is **strongly connected**, *if there exist both a directed u–v path and a directed v–u path for every pair of vertices $u, v \in V$.*

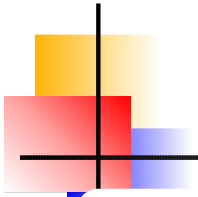
A component of digraph $\overrightarrow{G}$ *is a maximal weakly connected subgraph of $\overrightarrow{G}$.*

Example



Obr.: Digraphs with warious types of connesctivity.

a) weakly connected b) unilaterally connected c) strongly connected



Tarry's algorithm

Algorithm

Tarry's algorithm *for creating a closed walk in a graph* $G = (V, H)$ *starting in arbitrary vertex* $s \in V$ *and containing all edges of a component of* G *determined by vertex* s.
Resulting walk will be called **Tarry's walk**.

- **Step 1.** *Start with arbitrary vertex* $s \in V$, *set* $u := s$, $T = (u)$.
  $\{T$ *is inicialised as a trivial walk containing only one vertex.*$\}$
- **Step 2.** *If possible, choose an edge* $\{u, v\}$ *incident with last vertex* u *of the walk* T *observing undermentione rules* **T1**, **T2** *and include it into walk* T.

  Mark the direction in which the edge $\{u, v\}$ *is used in* T. *If the vertex* v *was not used in* T *till now denote the edge* $\{u, v\}$ *as the* **first access edge - FAE**.

  Rules for selection next edge:

  T1: *Every edge* $\{u, v\} \in H$ *can be used in one direction only once (i.e. once in direction* $\{u, v\}$ *and once in dirction* $\{v, u\}$*).*
  T2: *First access edge - FAE - can be used only if there is no other possibility.*

Tarry's algorithm

Algorithm

Tarry's algorithm *for creating a closed walk in a graph* $G = (V, H)$ *starting in arbitrary vertex* $s \in V$ *and containing all edges of a component of* G *determined by vertex* s.
Resulting walk will be called **Tarry's walk**.

- **Step 1.** *Start with arbitrary vertex* $s \in V$, *set* $u := s$, $T = (u)$.
  $\{T$ *is inicialised as a trivial walk containing only one vertex.*$\}$
- **Step 2.** *If possible, choose an edge* $\{u, v\}$ *incident with last vertex* u *of the walk* T *observing undermentione rules* **T1**, **T2** *and include it into walk* T.

  Mark the direction in which the edge $\{u, v\}$ *is used in* T. *If the vertex* v *was not used in* T *till now denote the edge* $\{u, v\}$ *as the* **first access edge - FAE**.

  Rules for selection next edge:

  T1: *Every edge* $\{u, v\} \in H$ *can be used in one direction only once (i.e. once in direction* $\{u, v\}$ *and once in dirction* $\{v, u\}$*).*
  T2: *First access edge - FAE - can be used only if there is no other possibility.*

Tarry's algorithm

Algorithm

Tarry's algorithm *for creating a closed walk in a graph* $G = (V, H)$ *starting in arbitrary vertex* $s \in V$ *and containing all edges of a component of* G *determined by vertex* s.
Resulting walk will be called **Tarry's walk**.

- **Step 1.** *Start with arbitrary vertex* $s \in V$, *set* $u := s$, $T = (u)$.
  $\{T$ *is inicialised as a trivial walk containing only one vertex.*$\}$
- **Step 2.** *If possible, choose an edge* $\{u, v\}$ *incident with last vertex* u *of the walk* T *observing undermentione rules* **T1**, **T2** *and include it into walk* T.

  Mark the direction in which the edge $\{u, v\}$ *is used in* T. *If the vertex* v *was not used in* T *till now denote the edge* $\{u, v\}$ *as the* **first access edge - FAE**.

  Rules for selection next edge:

  T1: *Every edge* $\{u, v\} \in H$ *can be used in one direction only once (i.e. once in direction* $\{u, v\}$ *and once in dirction* $\{v, u\}$*).*
  T2: *First access edge - FAE - can be used only if there is no other possibility.*

Tarry's algorithm

Algorithm

Tarry's algorithm *for creating a closed walk in a graph* $G = (V, H)$ *starting in arbitrary vertex* $s \in V$ *and containing all edges of a component of* G *determined by vertex* s.
Resulting walk will be called **Tarry's walk**.

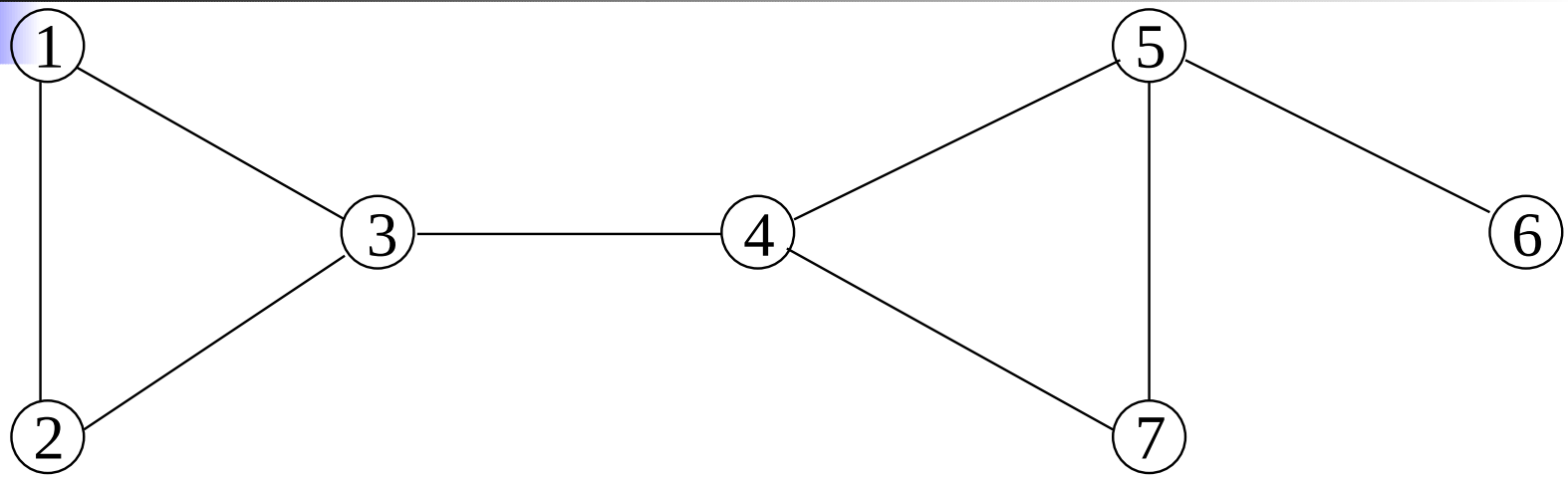
- **Step 1.** *Start with arbitrary vertex* $s \in V$, *set* $u := s$, $T = (u)$.
  $\{T$ *is inicialised as a trivial walk containing only one vertex.*$\}$
- **Step 2.** *If possible, choose an edge* $\{u, v\}$ *incident with last vertex* u *of the walk* T *observing undermentione rules* **T1**, **T2** *and include it into walk* T.

  Mark the direction in which the edge $\{u, v\}$ *is used in* T. *If the vertex* v *was not used in* T *till now denote the edge* $\{u, v\}$ *as the* **first access edge - FAE**.

  Rules for selection next edge:

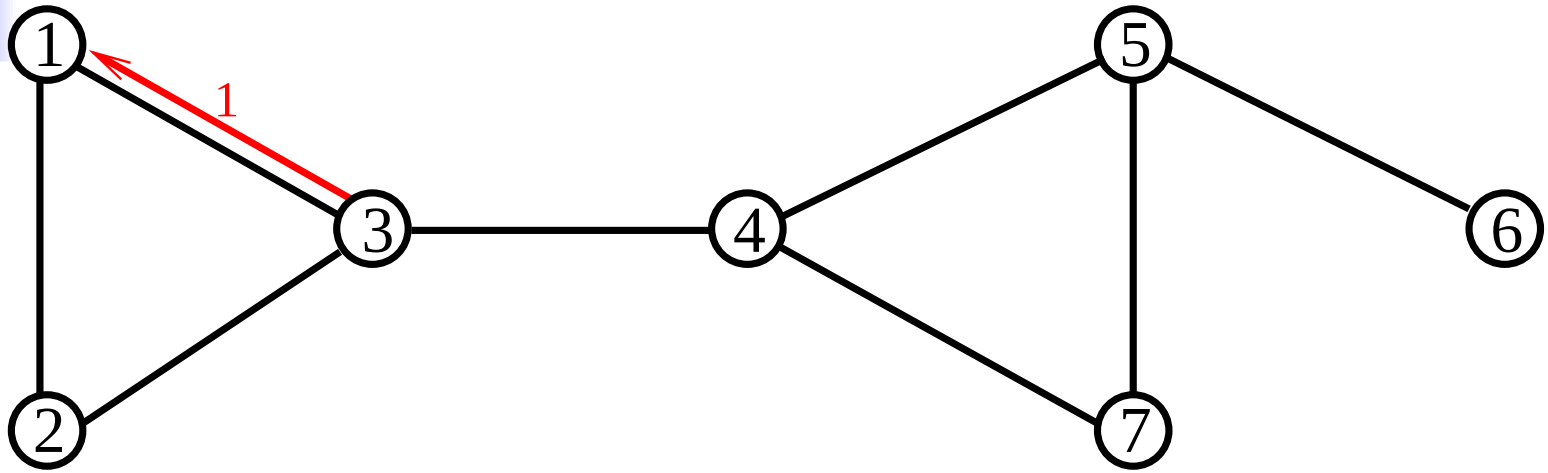
  T1: *Every edge* $\{u, v\} \in H$ *can be used in one direction only once (i.e. once in direction* $\{u, v\}$ *and once in dirction* $\{v, u\}$*).*
  T2: *First access edge - FAE - can be used only if there is no other possibility.*

Example



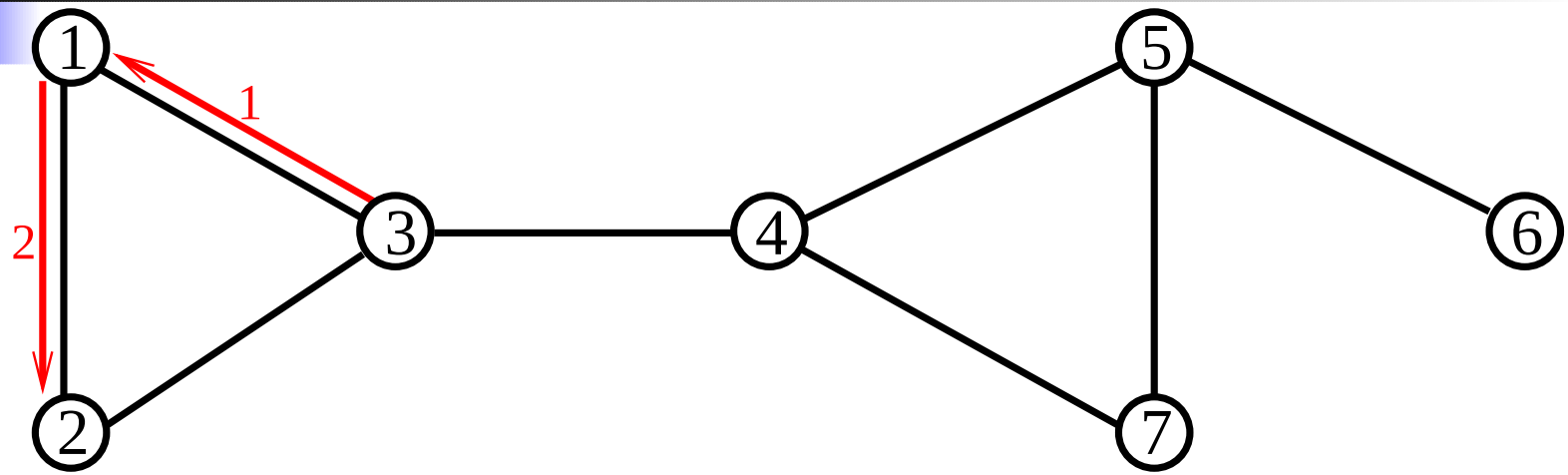
r		{1,2}	{1,3}	{2,3}	{3,4}	{4,5}	{4,7}	{5,6}	{5,7}	1	2	3	4	5	6	7
0												•				
1	{3,1}		⇐							•						
2	{1,2}	⇒									•					
3	{2,3}			→												
4	{3,2}			←												
5	{2,1}	←														
6	{1,3}		→													
7	{3,4}				⇒								•			
8	{4,5}					⇒								•		
9	{5,6}							⇒							•	
10	{6,5}							←								
11	{5,7}								⇒							•
12	{7,4}						←									
13	{4,7}						→									
14	{7,5}								←							
15	{5,4}					←										
16	{4,3}				←											

Example



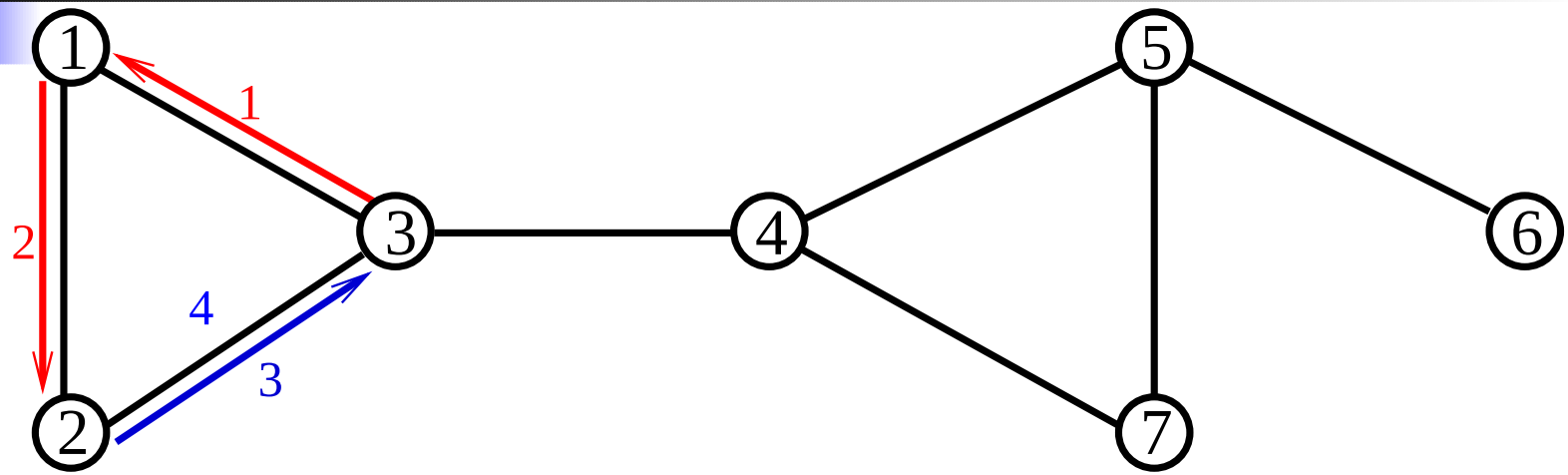
r		{1,2}	{1,3}	{2,3}	{3,4}	{4,5}	{4,7}	{5,6}	{5,7}	1	2	3	4	5	6	7
0												•				
1	{3,1}		⇐							•						
2	{1,2}	⇒									•					
3	{2,3}			→												
4	{3,2}			←												
5	{2,1}	←														
6	{1,3}		→													
7	{3,4}				⇒								•			
8	{4,5}					⇒								•		
9	{5,6}							⇒							•	
10	{6,5}							←								
11	{5,7}								⇒							•
12	{7,4}						←									
13	{4,7}						→									
14	{7,5}								←							
15	{5,4}					←										
16	{4,3}				←											

Example



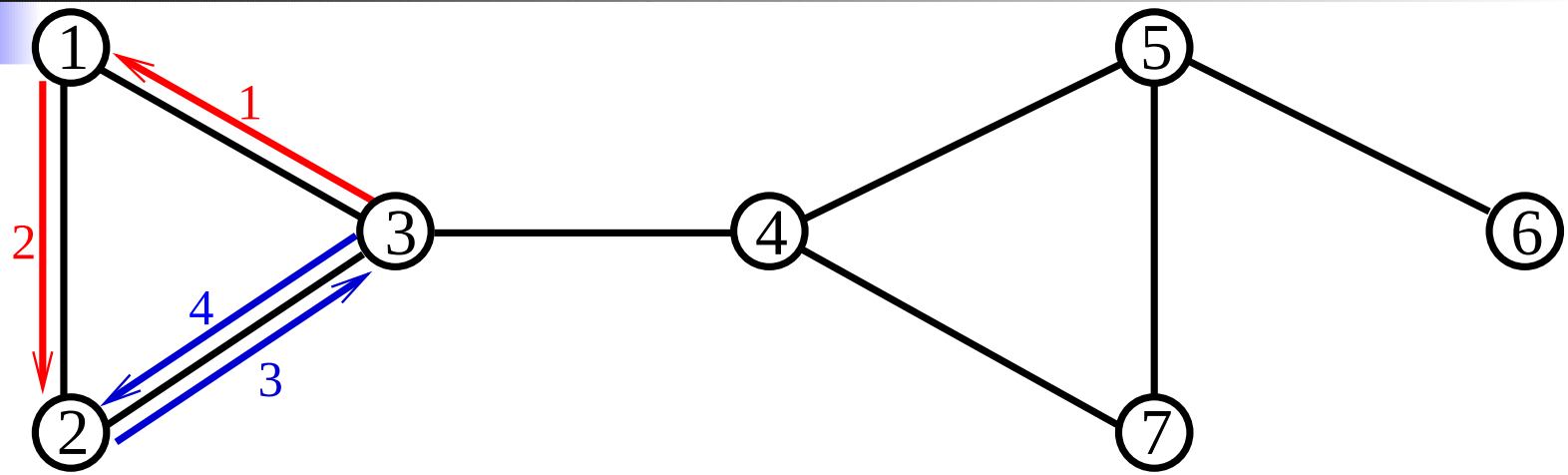
r		{1,2}	{1,3}	{2,3}	{3,4}	{4,5}	{4,7}	{5,6}	{5,7}	1	2	3	4	5	6	7
0												•				
1	{3,1}		⇐							•						
2	{1,2}	⇒									•					
3	{2,3}			→												
4	{3,2}			←												
5	{2,1}	←														
6	{1,3}		→													
7	{3,4}				⇒								•			
8	{4,5}					⇒								•		
9	{5,6}							⇒							•	
10	{6,5}							←								
11	{5,7}								⇒							•
12	{7,4}						←									
13	{4,7}						→									
14	{7,5}								←							
15	{5,4}					←										
16	{4,3}				←											

Example



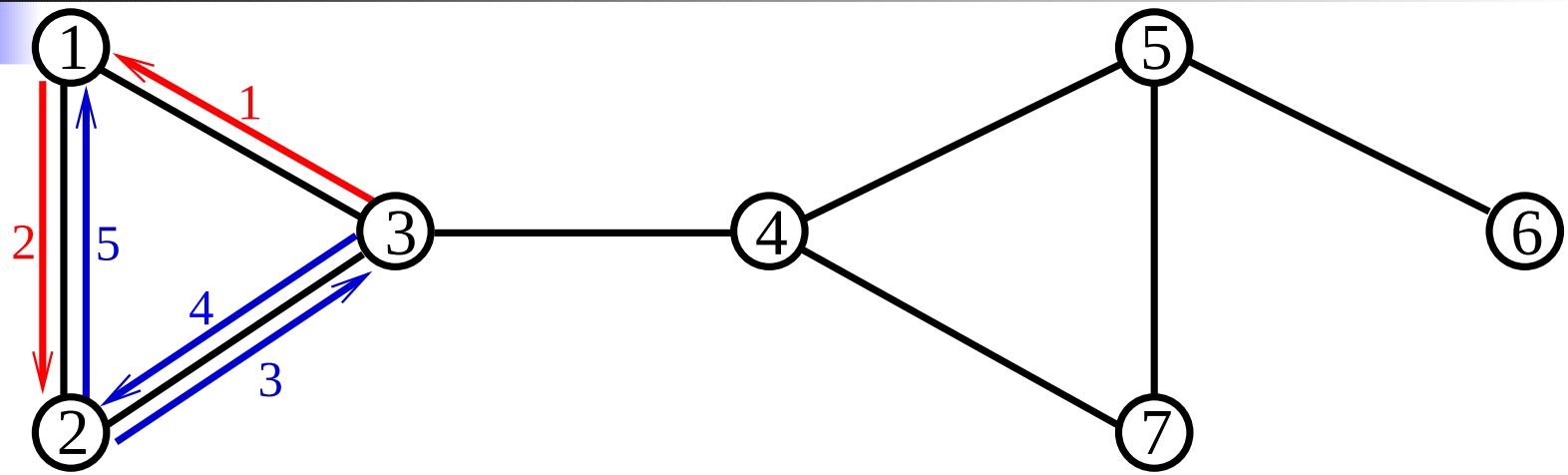
r		{1,2}	{1,3}	{2,3}	{3,4}	{4,5}	{4,7}	{5,6}	{5,7}	1	2	3	4	5	6	7
0												•				
1	{3,1}		⇐							•						
2	{1,2}	⇒									•					
3	{2,3}			→												
4	{3,2}			←												
5	{2,1}	←														
6	{1,3}		→													
7	{3,4}				⇒								•			
8	{4,5}					⇒								•		
9	{5,6}							⇒							•	
10	{6,5}							←								
11	{5,7}								⇒							•
12	{7,4}						←									
13	{4,7}						→									
14	{7,5}								←							
15	{5,4}					←										
16	{4,3}				←											

Example



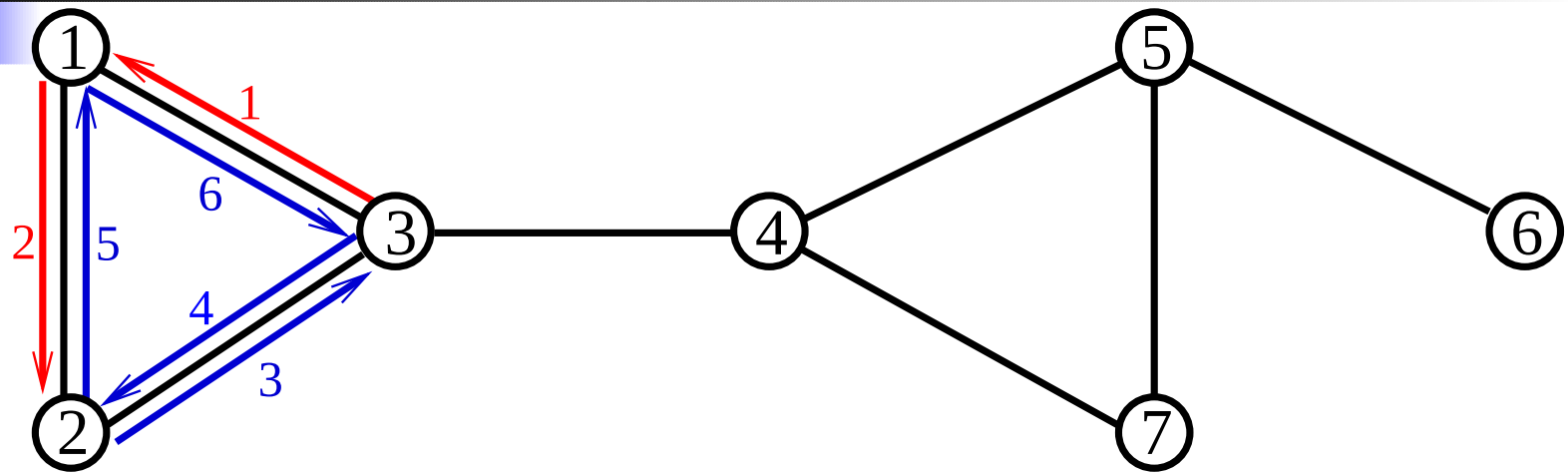
r		{1,2}	{1,3}	{2,3}	{3,4}	{4,5}	{4,7}	{5,6}	{5,7}	1	2	3	4	5	6	7
0												•				
1	{3,1}		⇐							•						
2	{1,2}	⇒									•					
3	{2,3}			→												
4	{3,2}			←												
5	{2,1}	←														
6	{1,3}		→													
7	{3,4}				⇒								•			
8	{4,5}					⇒								•		
9	{5,6}							⇒							•	
10	{6,5}							←								
11	{5,7}								⇒							•
12	{7,4}						←									
13	{4,7}						→									
14	{7,5}								←							
15	{5,4}					←										
16	{4,3}				←											

Example



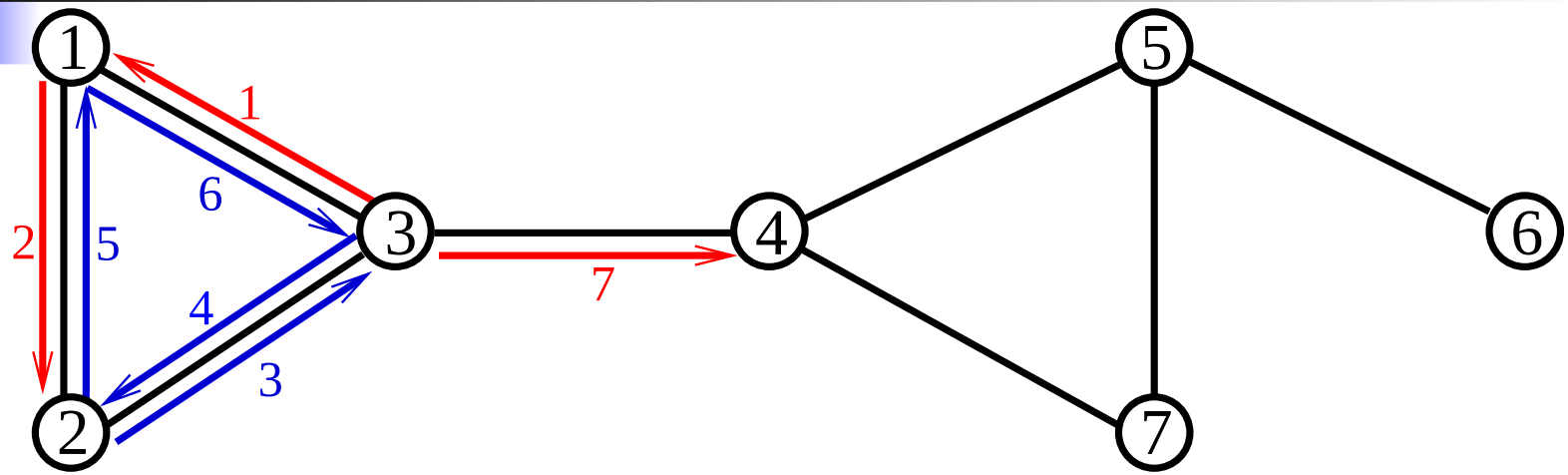
r		{1,2}	{1,3}	{2,3}	{3,4}	{4,5}	{4,7}	{5,6}	{5,7}	1	2	3	4	5	6	7
0												•				
1	{3,1}		⇐							•						
2	{1,2}	⇒									•					
3	{2,3}			→												
4	{3,2}			←												
5	{2,1}	←														
6	{1,3}		→										•			
7	{3,4}				⇒									•		
8	{4,5}					⇒									•	
9	{5,6}						⇒									•
10	{6,5}							←								
11	{5,7}								⇒							•
12	{7,4}						←									
13	{4,7}						→									
14	{7,5}								←							
15	{5,4}					←										
16	{4,3}				←											

Example



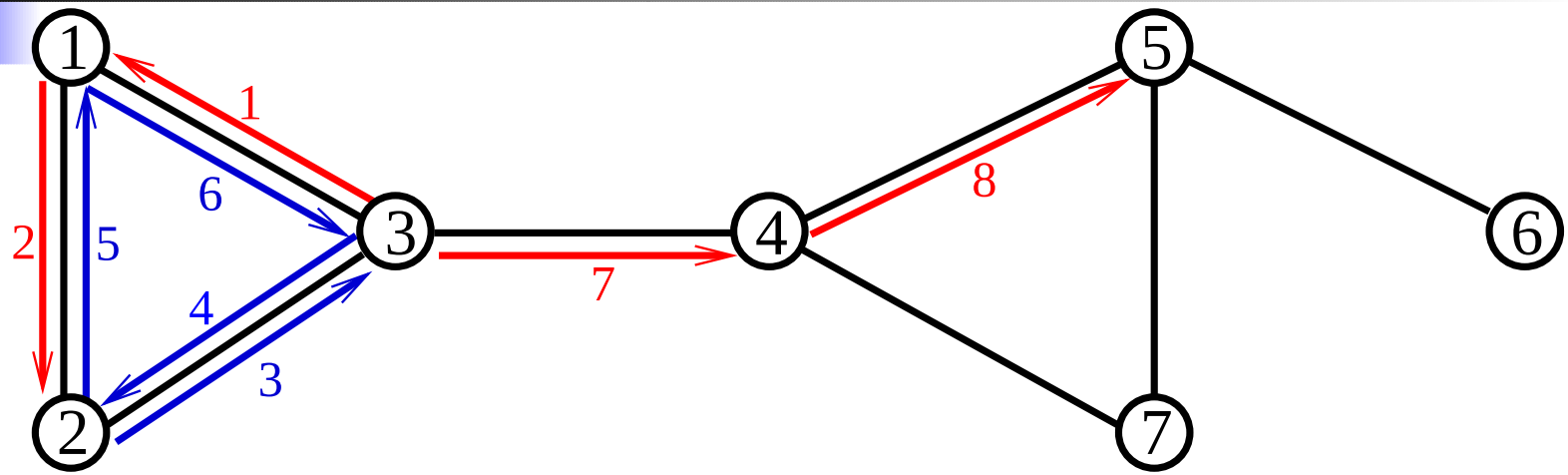
r		{1,2}	{1,3}	{2,3}	{3,4}	{4,5}	{4,7}	{5,6}	{5,7}	1	2	3	4	5	6	7
0												•				
1	{3,1}		⇐							•						
2	{1,2}	⇒									•					
3	{2,3}			→												
4	{3,2}			←												
5	{2,1}	←														
6	{1,3}		→													
7	{3,4}				⇒								•			
8	{4,5}					⇒								•		
9	{5,6}							⇒							•	
10	{6,5}							←								
11	{5,7}								⇒							•
12	{7,4}						←									
13	{4,7}						→									
14	{7,5}								←							
15	{5,4}					←										
16	{4,3}				←											

Example



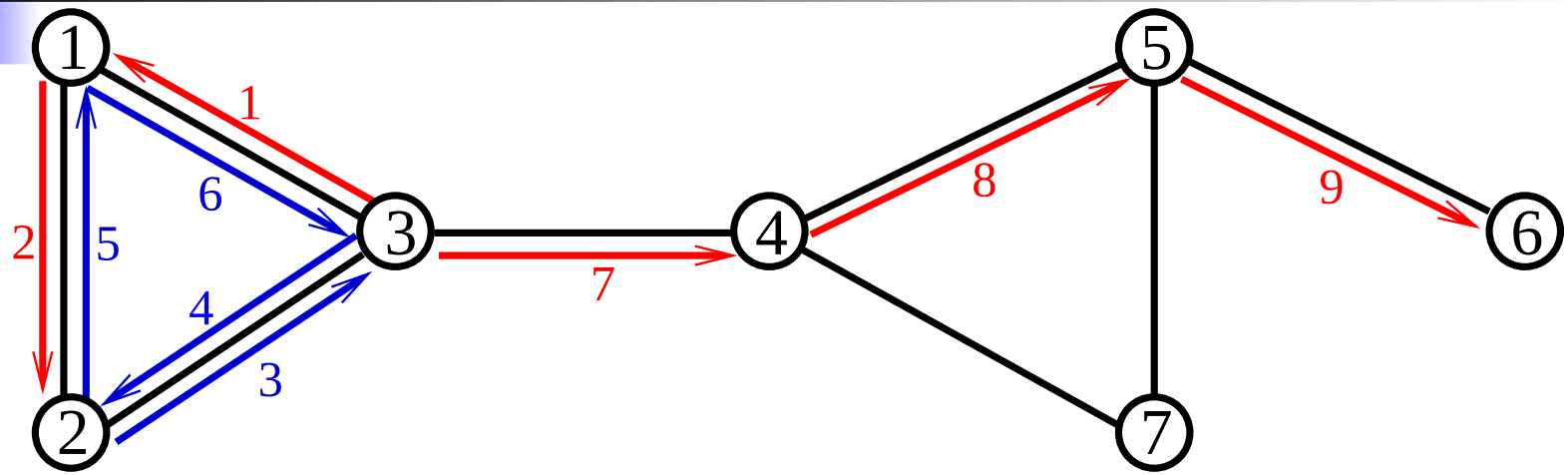
r		{1,2}	{1,3}	{2,3}	{3,4}	{4,5}	{4,7}	{5,6}	{5,7}	1	2	3	4	5	6	7
0												•				
1	{3,1}		⇐							•						
2	{1,2}	⇒									•					
3	{2,3}			→												
4	{3,2}			←												
5	{2,1}	←														
6	{1,3}		→													
7	{3,4}				⇒								•			
8	{4,5}					⇒								•		
9	{5,6}						⇒								•	
10	{6,5}						←									
11	{5,7}							⇒								•
12	{7,4}							←								
13	{4,7}							→								
14	{7,5}								←							
15	{5,4}					←										
16	{4,3}				←											

Example



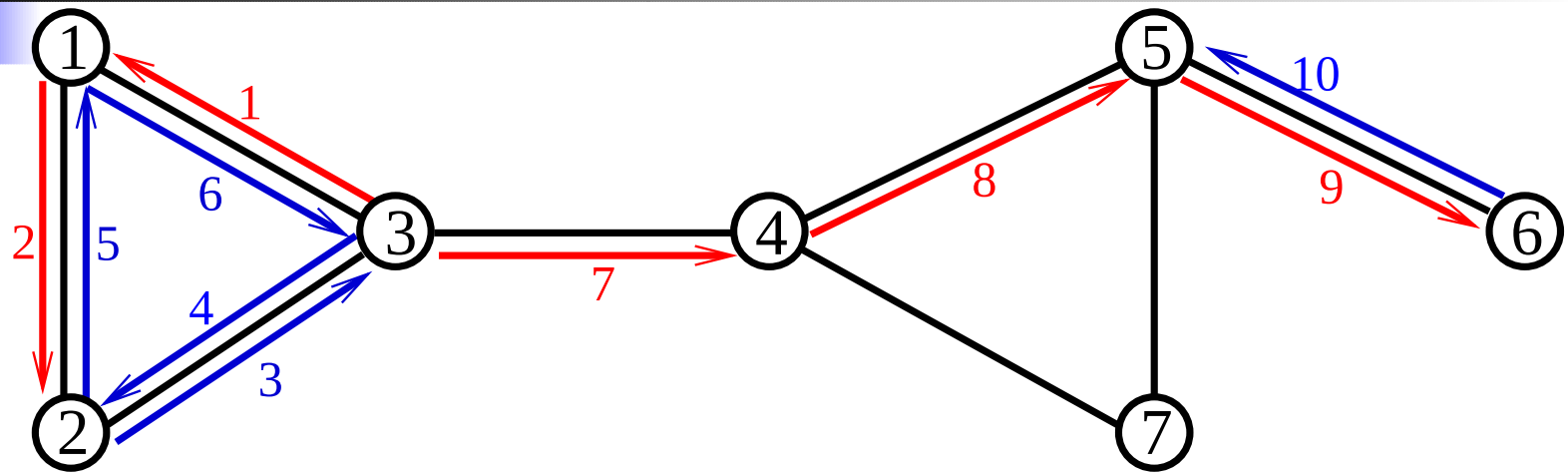
r		{1,2}	{1,3}	{2,3}	{3,4}	{4,5}	{4,7}	{5,6}	{5,7}	1	2	3	4	5	6	7
0												•				
1	{3,1}		⇐							•						
2	{1,2}	⇒									•					
3	{2,3}			→												
4	{3,2}			←												
5	{2,1}	←														
6	{1,3}		→													
7	{3,4}				⇒								•			
8	{4,5}					⇒								•		
9	{5,6}							⇒							•	
10	{6,5}							←								
11	{5,7}								⇒							•
12	{7,4}						←									
13	{4,7}						→									
14	{7,5}								←							
15	{5,4}					←										
16	{4,3}				←											

Example



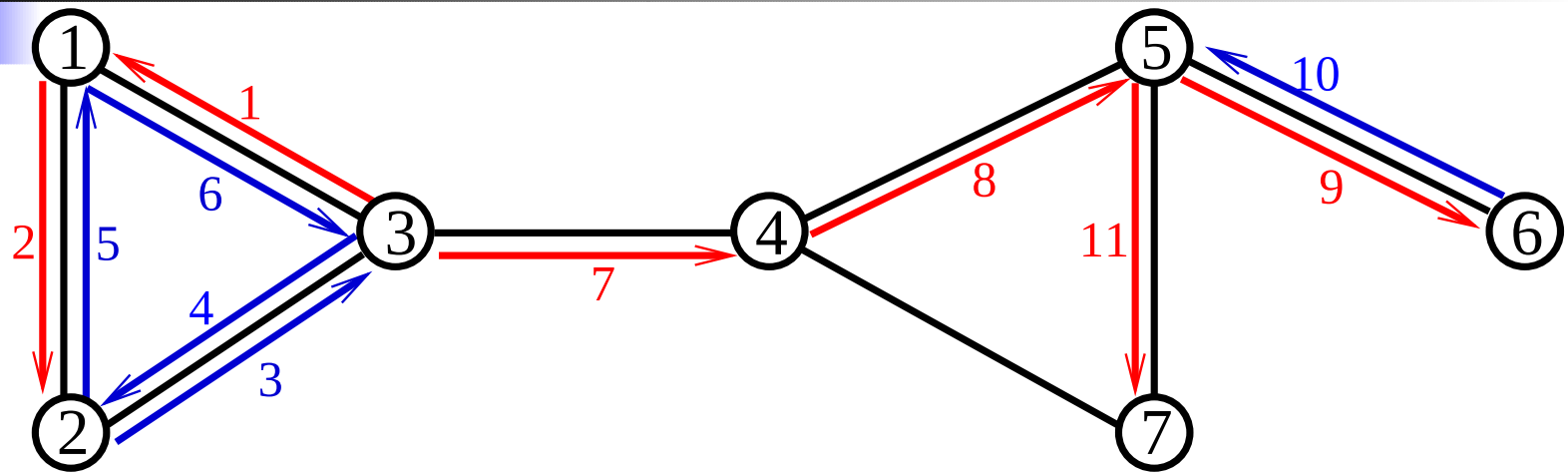
r		{1,2}	{1,3}	{2,3}	{3,4}	{4,5}	{4,7}	{5,6}	{5,7}	1	2	3	4	5	6	7
0												•				
1	{3,1}		⇐							•						
2	{1,2}	⇒									•					
3	{2,3}			→												
4	{3,2}			←												
5	{2,1}	←														
6	{1,3}		→													
7	{3,4}				⇒								•			
8	{4,5}					⇒								•		
9	{5,6}							⇒							•	
10	{6,5}							←								
11	{5,7}								⇒							•
12	{7,4}						←									
13	{4,7}						→									
14	{7,5}								←							
15	{5,4}					←										
16	{4,3}				←											

Example



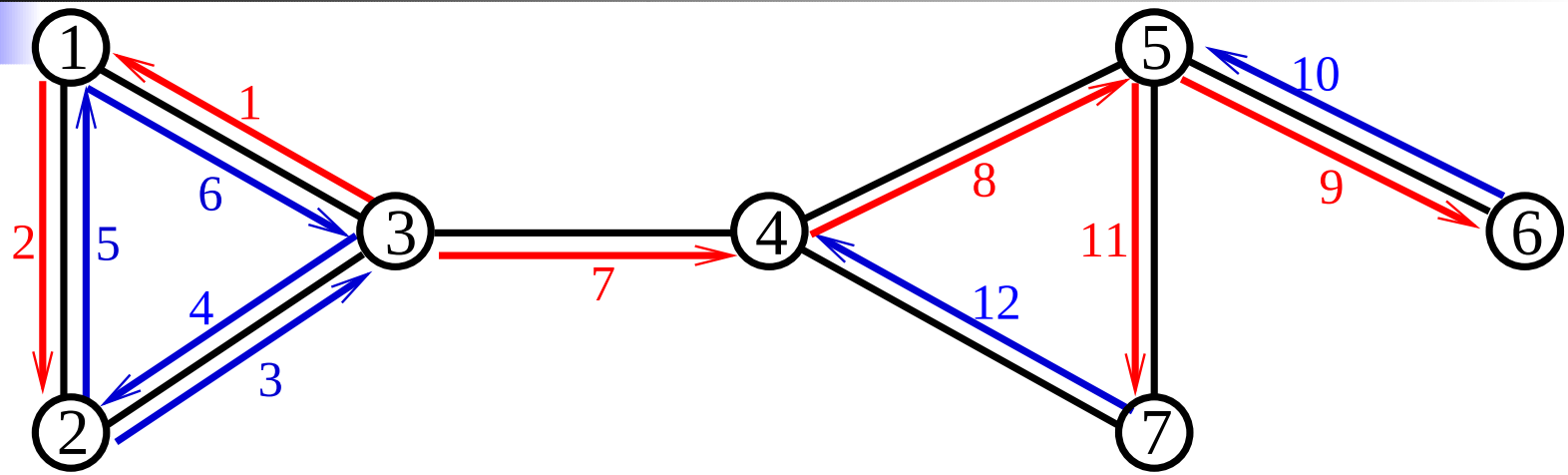
r		{1,2}	{1,3}	{2,3}	{3,4}	{4,5}	{4,7}	{5,6}	{5,7}	1	2	3	4	5	6	7
0												•				
1	{3,1}		⇐							•						
2	{1,2}	⇒									•					
3	{2,3}			→												
4	{3,2}			←												
5	{2,1}	←														
6	{1,3}		→													
7	{3,4}				⇒								•			
8	{4,5}					⇒								•		
9	{5,6}							⇒							•	
10	{6,5}							←								
11	{5,7}								⇒							•
12	{7,4}						←									
13	{4,7}						→									
14	{7,5}								←							
15	{5,4}					←										
16	{4,3}				←											

Example



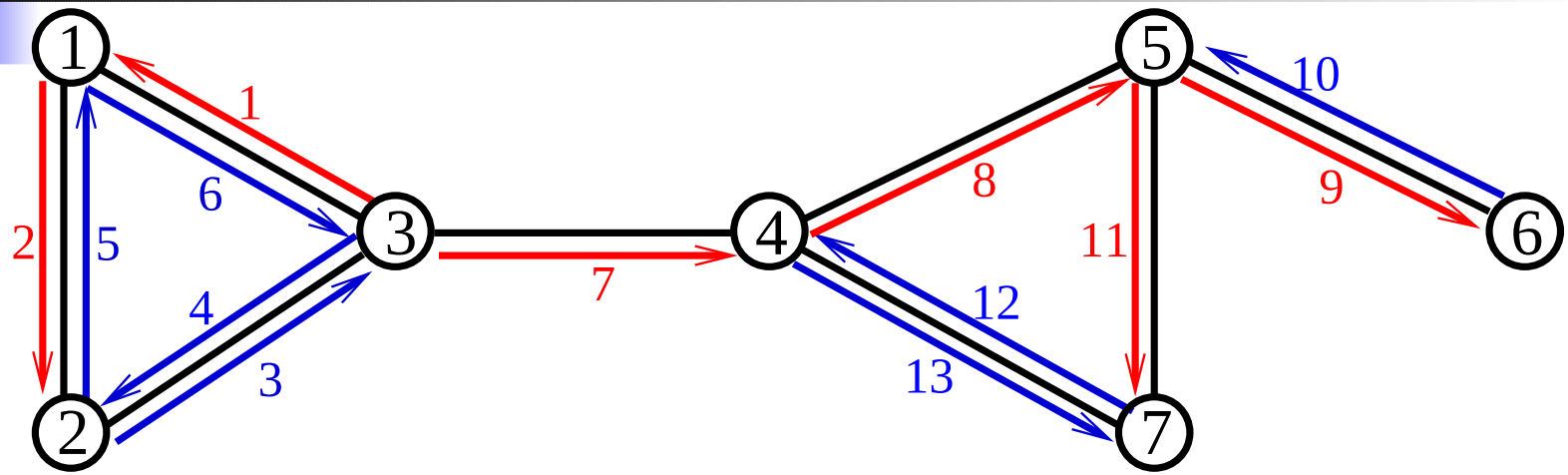
r		{1,2}	{1,3}	{2,3}	{3,4}	{4,5}	{4,7}	{5,6}	{5,7}	1	2	3	4	5	6	7
0												•				
1	{3,1}		⇐							•						
2	{1,2}	⇒									•					
3	{2,3}			→												
4	{3,2}			←												
5	{2,1}	←														
6	{1,3}		→													
7	{3,4}				⇒								•			
8	{4,5}					⇒								•		
9	{5,6}							⇒							•	
10	{6,5}							←								
11	{5,7}								⇒							•
12	{7,4}						←									
13	{4,7}						→									
14	{7,5}								←							
15	{5,4}					←										
16	{4,3}				←											

Example



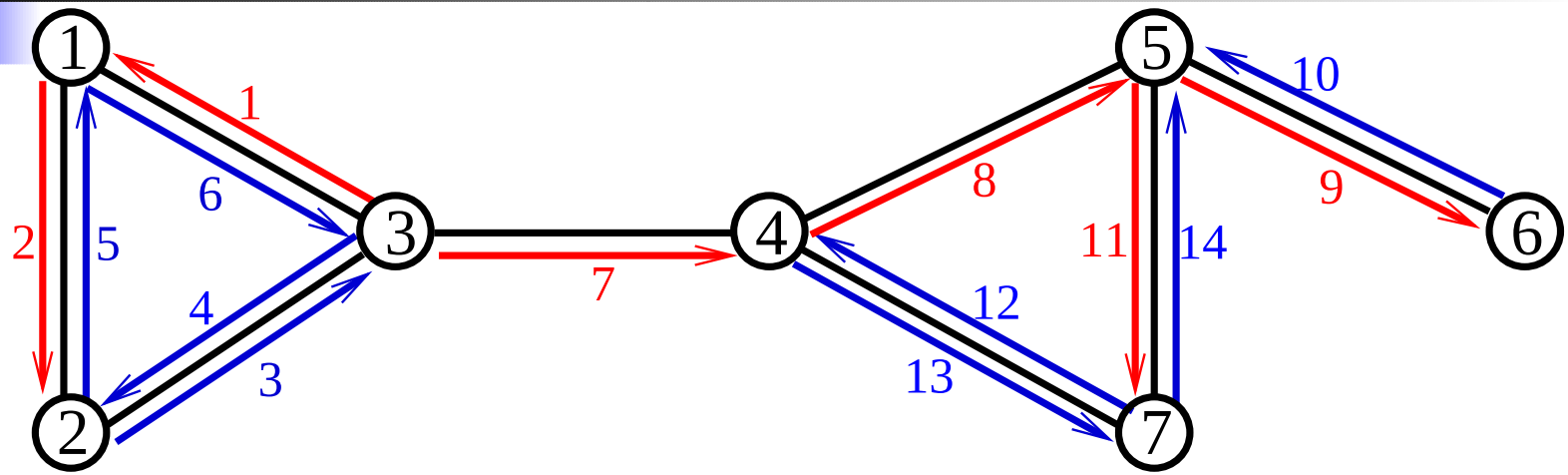
r		{1,2}	{1,3}	{2,3}	{3,4}	{4,5}	{4,7}	{5,6}	{5,7}	1	2	3	4	5	6	7
0												•				
1	{3,1}		⇐							•						
2	{1,2}	⇒									•					
3	{2,3}			→												
4	{3,2}			←												
5	{2,1}	←														
6	{1,3}		→													
7	{3,4}				⇒								•			
8	{4,5}					⇒								•		
9	{5,6}							⇒							•	
10	{6,5}							←								
11	{5,7}								⇒							•
12	{7,4}						←									
13	{4,7}						→									
14	{7,5}								←							
15	{5,4}					←										
16	{4,3}				←											

Example



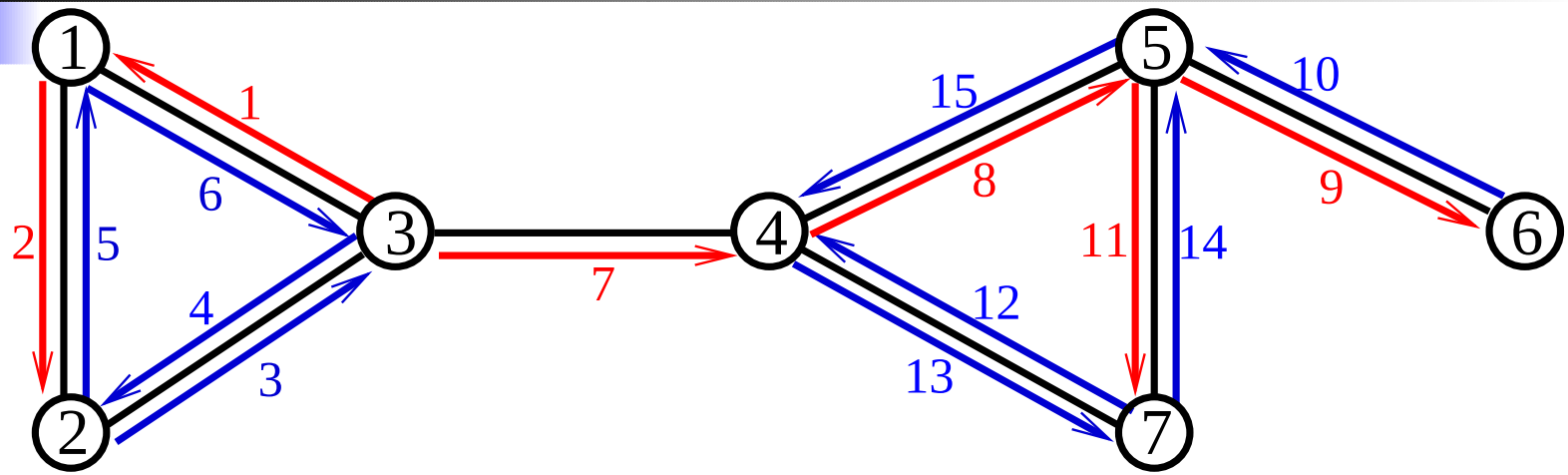
r		{1,2}	{1,3}	{2,3}	{3,4}	{4,5}	{4,7}	{5,6}	{5,7}	1	2	3	4	5	6	7
0												•				
1	{3,1}		⇐							•						
2	{1,2}	⇒									•					
3	{2,3}			→												
4	{3,2}			←												
5	{2,1}	←														
6	{1,3}		→													
7	{3,4}				⇒								•			
8	{4,5}					⇒								•		
9	{5,6}							⇒							•	
10	{6,5}							←								
11	{5,7}								⇒							•
12	{7,4}						←									
13	{4,7}						→									
14	{7,5}								←							
15	{5,4}					←										
16	{4,3}				←											

Example



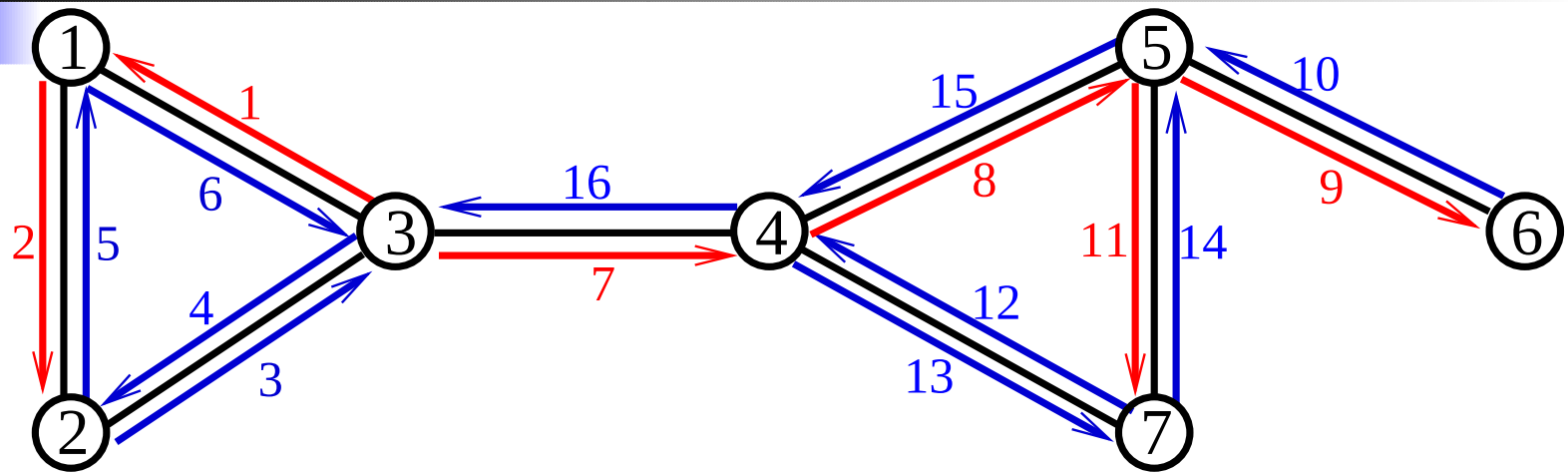
r		{1,2}	{1,3}	{2,3}	{3,4}	{4,5}	{4,7}	{5,6}	{5,7}	1	2	3	4	5	6	7
0												•				
1	{3,1}		⇐							•						
2	{1,2}	⇒									•					
3	{2,3}			→												
4	{3,2}			←												
5	{2,1}	←														
6	{1,3}		→													
7	{3,4}				⇒								•			
8	{4,5}					⇒								•		
9	{5,6}							⇒							•	
10	{6,5}							←								
11	{5,7}								⇒							•
12	{7,4}						←									
13	{4,7}						→									
14	{7,5}								←							
15	{5,4}					←										
16	{4,3}				←											

Example

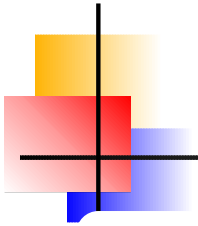


r		{1,2}	{1,3}	{2,3}	{3,4}	{4,5}	{4,7}	{5,6}	{5,7}	1	2	3	4	5	6	7
0												•				
1	{3,1}		⇐							•						
2	{1,2}	⇒									•					
3	{2,3}			→												
4	{3,2}			←												
5	{2,1}	←														
6	{1,3}		→													
7	{3,4}				⇒								•			
8	{4,5}					⇒								•		
9	{5,6}							⇒							•	
10	{6,5}							←								
11	{5,7}								⇒							•
12	{7,4}						←									
13	{4,7}						→									
14	{7,5}								←							
15	{5,4}					←										
16	{4,3}				←											

Example



r		{1,2}	{1,3}	{2,3}	{3,4}	{4,5}	{4,7}	{5,6}	{5,7}	1	2	3	4	5	6	7
0												•				
1	{3,1}		⇐							•						
2	{1,2}	⇒									•					
3	{2,3}			→												
4	{3,2}			←												
5	{2,1}	←														
6	{1,3}		→													
7	{3,4}				⇒								•			
8	{4,5}					⇒								•		
9	{5,6}							⇒							•	
10	{6,5}							←								
11	{5,7}								⇒							•
12	{7,4}						←									
13	{4,7}						→									
14	{7,5}								←							
15	{5,4}					←										
16	{4,3}				←											



Length of a walk

Definition

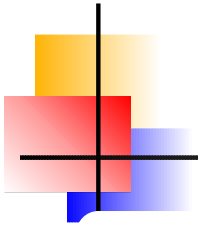
Let $\mu(u,v)$ be a u–v walk (resp. directed walk resp. quasi-walk) in an edge weighted graph $G = (V, H, c)$ (resp. digraph $\overrightarrow{G} = (V, H, c)$).
The length of the walk (resp. quasi-walk) $\mu(u,v)$ *or* **the cost of the walk (resp. quasi-walk)** *is the sum of edge weights of all edges in $\mu(u,v)$, wheras the edge weight is added so many times how many times corresponding edge occurs in the walk $\mu(u,v)$.*
The length of the walk $\mu(u,v)$ will be denoted by $d(\mu(u,v))$.

Remark

Definition of a walk allows also trivial walk having only one vertex. The length of such a walk is zero.

Remark

Last definition defines also the length of trail, path, cycle and all directed variants of these notions, since all are special cases of walk.



Length of a walk

Definition

Let $\mu(u,v)$ be a u–v walk (resp. directed walk resp. quasi-walk) in an edge weighted graph $G = (V, H, c)$ (resp. digraph $\overrightarrow{G} = (V, H, c)$).
The length of the walk (resp. quasi-walk) $\mu(u,v)$ *or* **the cost of the walk (resp. quasi-walk)** *is the sum of edge weights of all edges in $\mu(u,v)$, wheras the edge weight is added so many times how many times corresponding edge occurs in the walk $\mu(u,v)$.*
The length of the walk $\mu(u,v)$ will be denoted by $d(\mu(u,v))$.

Remark

Definition of a walk allows also trivial walk having only one vertex. The length of such a walk is zero.

Remark

Last definition defines also the length of trail, path, cycle and all directed variants of these notions, since all are special cases of walk.

Length of a walk

Definition

Let $\mu(u, v)$ be a u–v walk (resp. directed walk resp. quasi-walk) in an edge weighted graph $G = (V, H, c)$ (resp. digraph $\overrightarrow{G} = (V, H, c)$).
The length of the walk (resp. quasi-walk) $\mu(u, v)$ *or* **the cost of the walk (resp. quasi-walk)** *is the sum of edge weights of all edges in $\mu(u, v)$, wheras the edge weight is added so many times how many times corresponding edge occurs in the walk $\mu(u, v)$.*
The length of the walk $\mu(u, v)$ will be denoted by $d(\mu(u, v))$.

Remark

Definition of a walk allows also trivial walk having only one vertex. The length of such a walk is zero.

Remark

Last definition defines also the length of trail, path, cycle and all directed variants of these notions, since all are special cases of walk.

Dĺžka ťahu, dĺžka cesty

Remark

Definition of length can be simplified for trail, cycle etc. i.e. for instances of walk with no repeated edges as follows:

The length $d(\mu(u,v))$ of a trail (quasi-trail, path or quasi-path) $\mu(u,v)$ is the sum of edge weights of all edges of $\mu(u,v)$, i. e.

$$d(\mu(u,v)) = \sum_{h \in \mu(u,v)} c(h).$$

Remark

Sometimes it is useful to define the length of a walk $\mu(u,v)$ in a graph $G = (V,H)$, resp. digraph $\overrightarrow{G} = (V,H)$, which is not edge weighted. In mentioned case, the length of a walk $\mu(u,v)$ is defined as the number of edges of the walk $\mu(u,v)$.
Such defined length of walk is identical with the length of the walk in an edge weighted graph (resp digraph) $G' = (V,H,c)$, where $c(h) = 1$ for every edge $h \in H$.[a]

Dĺžka ťahu, dĺžka cesty

Remark

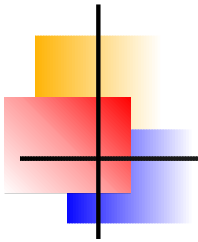
Definition of length can be simplified for trail, cycle etc. i.e. for instances of walk with no repeated edges as follows:

The length $d(\mu(u,v))$ of a trail (quasi-trail, path or quasi-path) $\mu(u,v)$ is the sum of edge weights of all edges of $\mu(u,v)$, i. e.

$$d(\mu(u,v)) = \sum_{h \in \mu(u,v)} c(h).$$

Remark

Sometimes it is useful to define the length of a walk $\mu(u,v)$ in a graph $G = (V,H)$, resp. digraph $\overrightarrow{G} = (V,H)$, which is not edge weighted. In mentioned case, the length of a walk $\mu(u,v)$ is defined as the number of edges of the walk $\mu(u,v)$.
Such defined length of walk is identical with the length of the walk in an edge weighted graph (resp digraph) $G' = (V,H,c)$, where $c(h) = 1$ for every edge $h \in H$.[a]



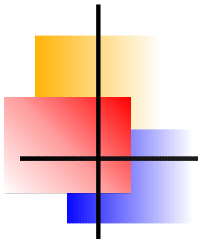
Shortest path in graph and digraph

Definition

Shortest *u-v* **path** *in an edge weighted graph* $G = (V, H, c)$ *(resp. in an arc weighted digraph* $\overrightarrow{G} = (V, H, c)$*) is that u-v path in* G *(resp. that directed u-v path in* $\overrightarrow{G}$*), which has the least length.*

Agreement

- *All shortest path algorithms will be formulated* **for arc weighted digraphs** $\overrightarrow{G} = (V, H, c)$ *with nonnegative arc weight* $c(h) \geq 0$.
- *We will suppose that* $0 \notin V$.



Shortest path in graph

Shortet path in a non directed graph $G = (V, H, c)$ can be found as the shortest path in digraph $\overrightarrow{G} = (V, \overrightarrow{H}, \overrightarrow{c})$, in which $\overrightarrow{H}$ is the set of directed edges containing for every non directed edge $h = \{u, v \in H\}$ two directed edges (u, v), (v, u), both with the same edge weight equal to $c(h) = c(\{u, v\})$, i.e.

$$\overrightarrow{c}(u, v) = \overrightarrow{c}(v, u) = c(h).$$

Fundamental shortest path algorithm

Algorithm

Fundamental point-to-all shortest path algorithm. *This algorithm will find all shortest u–v directed paths from a fixed vertex $u \in V$ into all reachable vertices $v \in V$ in an edge weighted digraph $\overrightarrow{G} = (V, H, c)$ with nonnegative edge weight $c(h)$ (and where $0 \notin V$).*

- **Step 1. Initialisation.**
  Assign two labels $t(i)$ a $x(i)$ for every vertex $i \in V$.
  {Label $t(i)$ is upper estimate of the till now the best found u–i path and $x(i)$ is it's last but one vertex.}
  Set $t(u) := 0$, $t(i) := \infty$ for $i \in V$, $i \neq u$ a $x(i) := 0$ for all $i \in V$.
- **Step 2.** *Find out if there exists an arc $(i, j) \in H$, for which it holds*

$$t(j) > t(i) + c(i, j). \tag{3}$$

  If $(i, j) \in H$ is an arc for which it holds (3), set:

$$t(j) := t(i) + c(i, j), \quad x(j) := i$$

  and GOTO Step 2.
- **Step 3.** *If for no arc from $(i, j) \in H$ holds (3), STOP.*

Fundamental shortest path algorithm

Algorithm

Fundamental point-to-all shortest path algorithm. *This algorithm will find all shortest u–v directed paths from a fixed vertex $u \in V$ into all reachable vertices $v \in V$ in an edge weighted digraph $\overrightarrow{G} = (V, H, c)$ with nonnegative edge weight $c(h)$ (and where $0 \notin V$).*

- **Step 1. Initialisation.**
  Assign two labels $t(i)$ a $x(i)$ for every vertex $i \in V$.
  {Label $t(i)$ is upper estimate of the till now the best found u–i path and $x(i)$ is it's last but one vertex.}
  Set $t(u) := 0$, $t(i) := \infty$ for $i \in V$, $i \neq u$ a $x(i) := 0$ for all $i \in V$.
- **Step 2.** *Find out if there exists an arc $(i, j) \in H$, for which it holds*

$$t(j) > t(i) + c(i, j). \tag{3}$$

  If $(i, j) \in H$ is an arc for which it holds (3), set:

$$t(j) := t(i) + c(i, j), \quad x(j) := i$$

  and GOTO Step 2.
- **Step 3.** *If for no arc from $(i, j) \in H$ holds (3), STOP.*

Fundamental shortest path algorithm

Algorithm

Fundamental point-to-all shortest path algorithm. *This algorithm will find all shortest u–v directed paths from a fixed vertex $u \in V$ into all reachable vertices $v \in V$ in an edge weighted digraph $\overrightarrow{G} = (V, H, c)$ with nonnegative edge weight $c(h)$ (and where $0 \notin V$).*

- **Step 1. Initialisation.**
  Assign two labels $t(i)$ a $x(i)$ for every vertex $i \in V$.
  {Label $t(i)$ is upper estimate of the till now the best found u–i path and $x(i)$ is it's last but one vertex.}
  Set $t(u) := 0$, $t(i) := \infty$ for $i \in V$, $i \neq u$ a $x(i) := 0$ for all $i \in V$.
- **Step 2.** *Find out if there exists an arc $(i, j) \in H$, for which it holds*

$$t(j) > t(i) + c(i, j). \tag{3}$$

  If $(i, j) \in H$ is an arc for which it holds (3), set:

$$t(j) := t(i) + c(i, j), \quad x(j) := i$$

  and GOTO Step 2.
- **Step 3.** *If for no arc from $(i, j) \in H$ holds (3), STOP.*

Fundamental shortest path algorithm

Algorithm

Fundamental point-to-all shortest path algorithm. *This algorithm will find all shortest u–v directed paths from a fixed vertex $u \in V$ into all reachable vertices $v \in V$ in an edge weighted digraph $\overrightarrow{G} = (V, H, c)$ with nonnegative edge weight $c(h)$ (and where $0 \notin V$).*

- **Step 1. Initialisation.**
  Assign two labels $t(i)$ a $x(i)$ for every vertex $i \in V$.
  {Label $t(i)$ is upper estimate of the till now the best found u–i path and $x(i)$ is it's last but one vertex.}
  Set $t(u) := 0$, $t(i) := \infty$ for $i \in V$, $i \neq u$ a $x(i) := 0$ for all $i \in V$.
- **Step 2.** *Find out if there exists an arc $(i, j) \in H$, for which it holds*

$$t(j) > t(i) + c(i, j). \tag{3}$$

  If $(i, j) \in H$ is an arc for which it holds (3), set:

$$t(j) := t(i) + c(i, j), \quad x(j) := i$$

  and GOTO Step 2.
- **Step 3.** *If for no arc from $(i, j) \in H$ holds (3), STOP.*

Fundamental shortest path algorithm

The shortest u–i path can be constructed backwards using labels $x(i)$ as the path going trough vertices

$$i, x(i), x(x(i)), x(x(x(i))), \ldots, u$$

This shortest path can be written in form:

$$(u, \ldots, x(x(x(i))), \underbrace{(x(x(x(i))), x(x(i)))}_{\text{arc}}, x(x(i)),$$

$$\underbrace{(x(x(i)), x(i))}_{\text{arc}}, x(i), \underbrace{(x(i), i)}_{\text{arc}}, i)$$

Finite value of label $t(i)$ after algorithm stopped represents the length of the shortest u–i path.

If $t(i) = \infty$, the the vertex i is unreachable from vertex u.

Notation of shortest path

Agreement (about notation)

Array $x(\ )$ contains after finishing a shortest path algorithm for every vertex $i \in V$, such that $t(i) < \infty$ pointer to the last but one vertex of the shortest u-i path.
Suppose that $x(\ )$ is the array of pointers obtained by a shortest path algorithm. We define recursively $x^{(k)}(j)$ for $j \in V$ as follows:

- $x^{(1)}(j) = x(j)$
- $x^{(k)}(j) = x(x^{(k-1)}(j))$

$$x^{(k)}(j) = \underbrace{x(x(\ldots x(}_{k\text{-times}} j) \ldots))$$

Then the sequence of vertices of the shortest u-j path can be written in reverse order like this:

$$j,\ x^{(1)}(j),\ x^{(2)}(j),\ \ldots\ x^{(k)}(j) \equiv u,$$

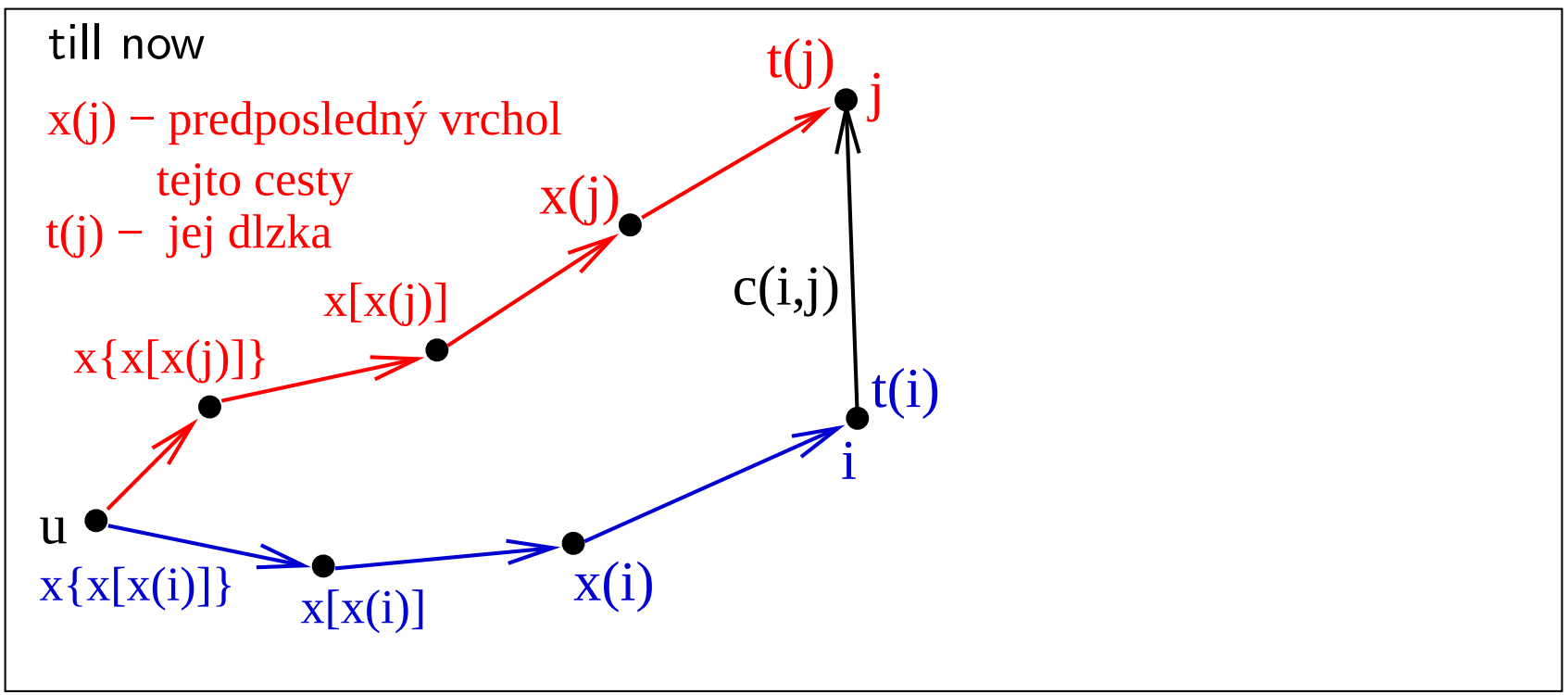
hence the shortest u-j path traverses through vertices

$$u \equiv x^{(k)}(j),\ x^{(k-1)}(j),\ \ldots,\ x^{(2)}(j),\ x^{(1)}(j),\ j$$

Principle of fundamental algorithm

Theorem

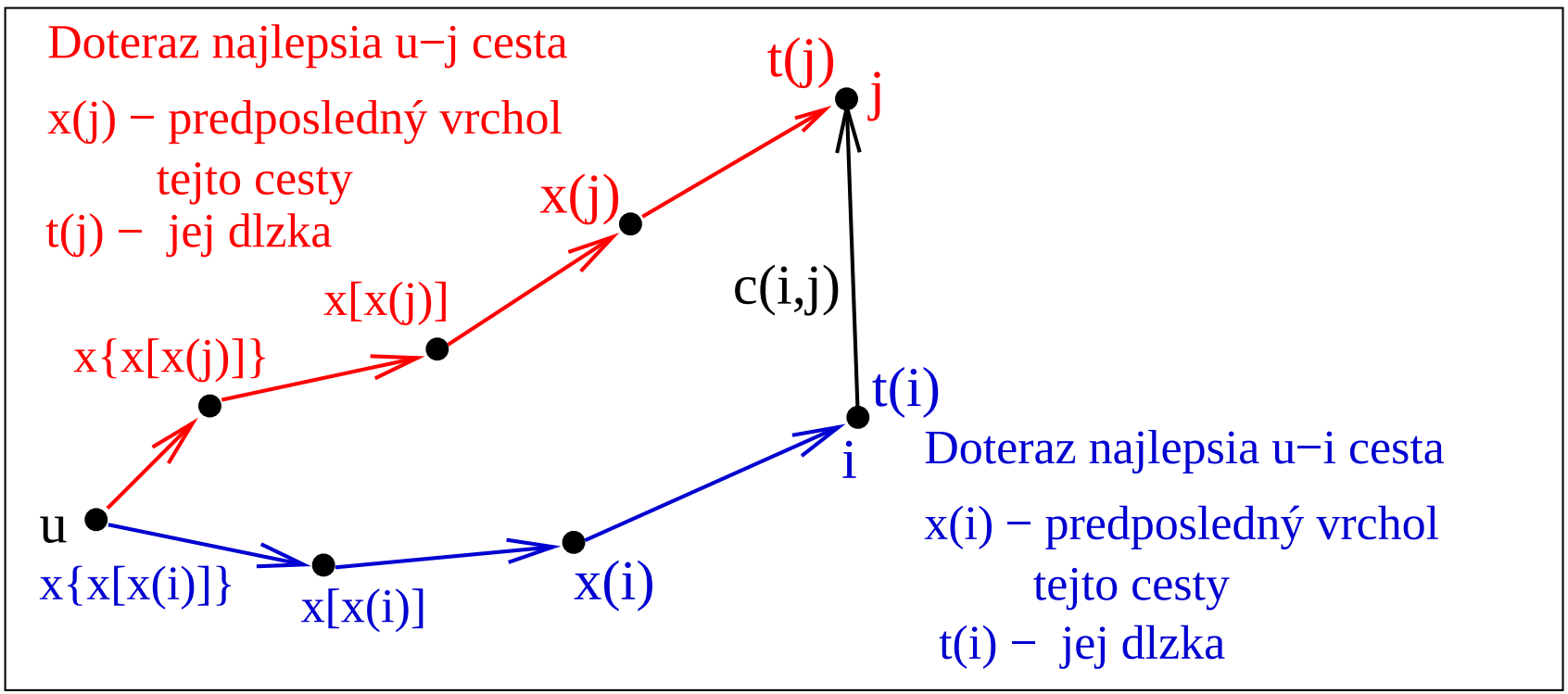
Fundamental algorithm will stop after a finite number of steps for arbitrary digraph $\overrightarrow{G} = (V, H, c)$ *with nonegative arc weight.*



Principle of fundamental algorithm

Theorem

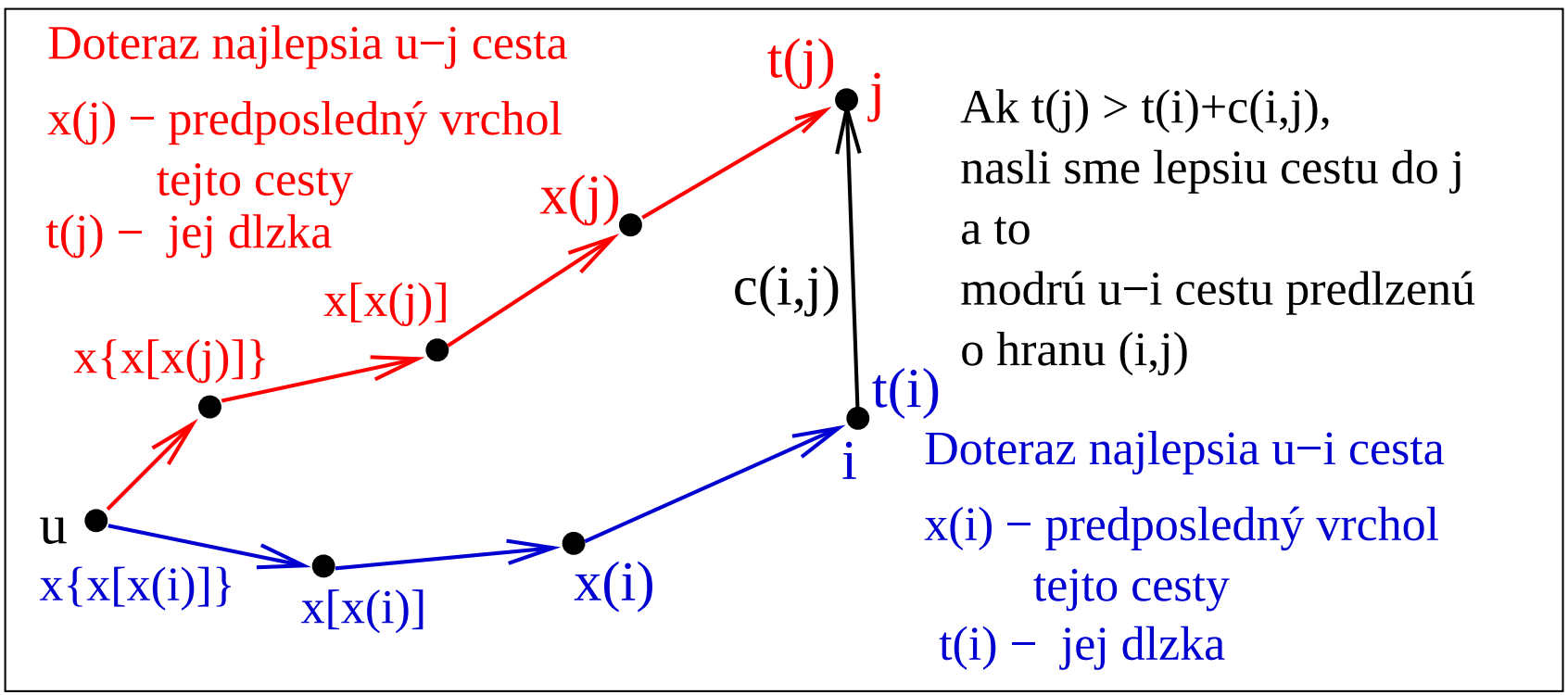
Fundamental algorithm will stop after a finite number of steps for arbitrary digraph $\overrightarrow{G} = (V, H, c)$ *with nonegative arc weight.*



Principle of fundamental algorithm

Theorem

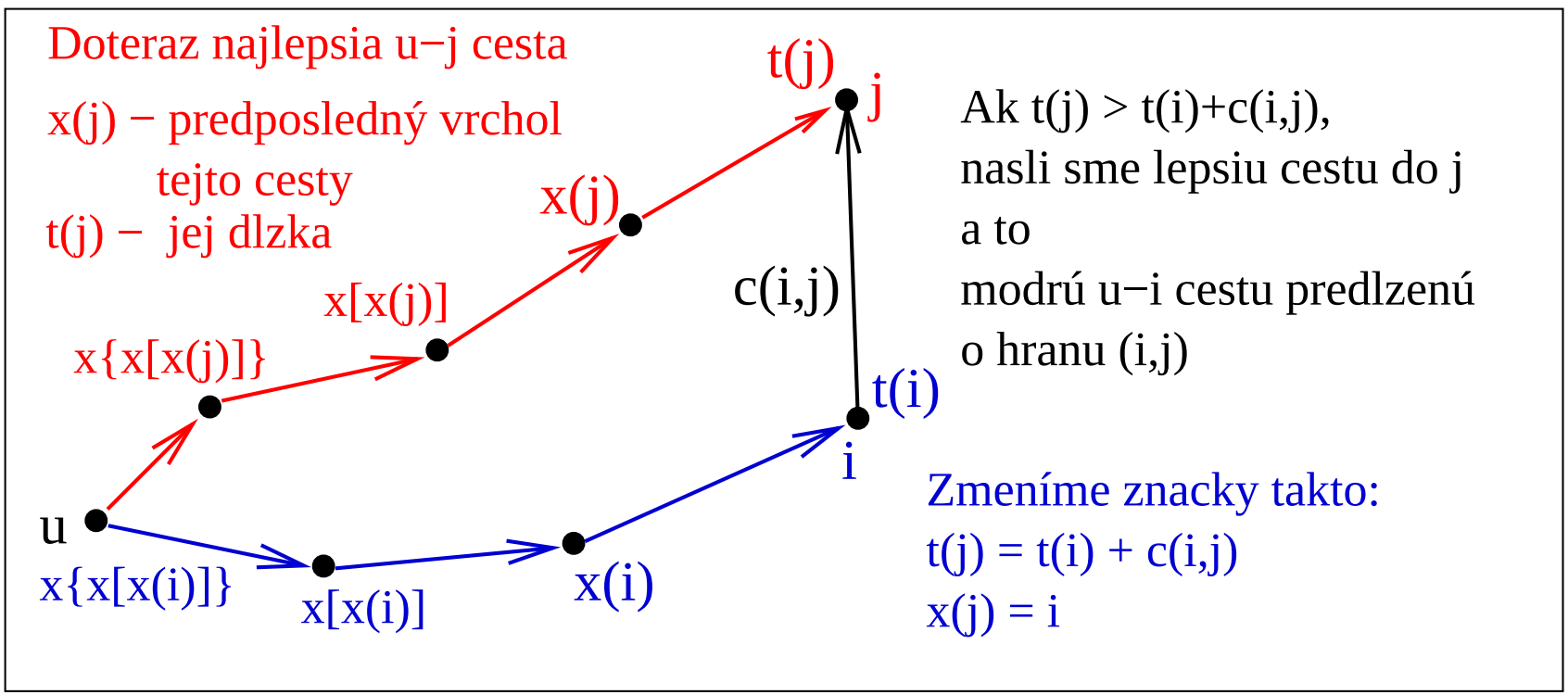
Fundamental algorithm will stop after a finite number of steps for arbitrary digraph $\overrightarrow{G} = (V, H, c)$ with nonegative arc weight.



Principle of fundamental algorithm

Theorem

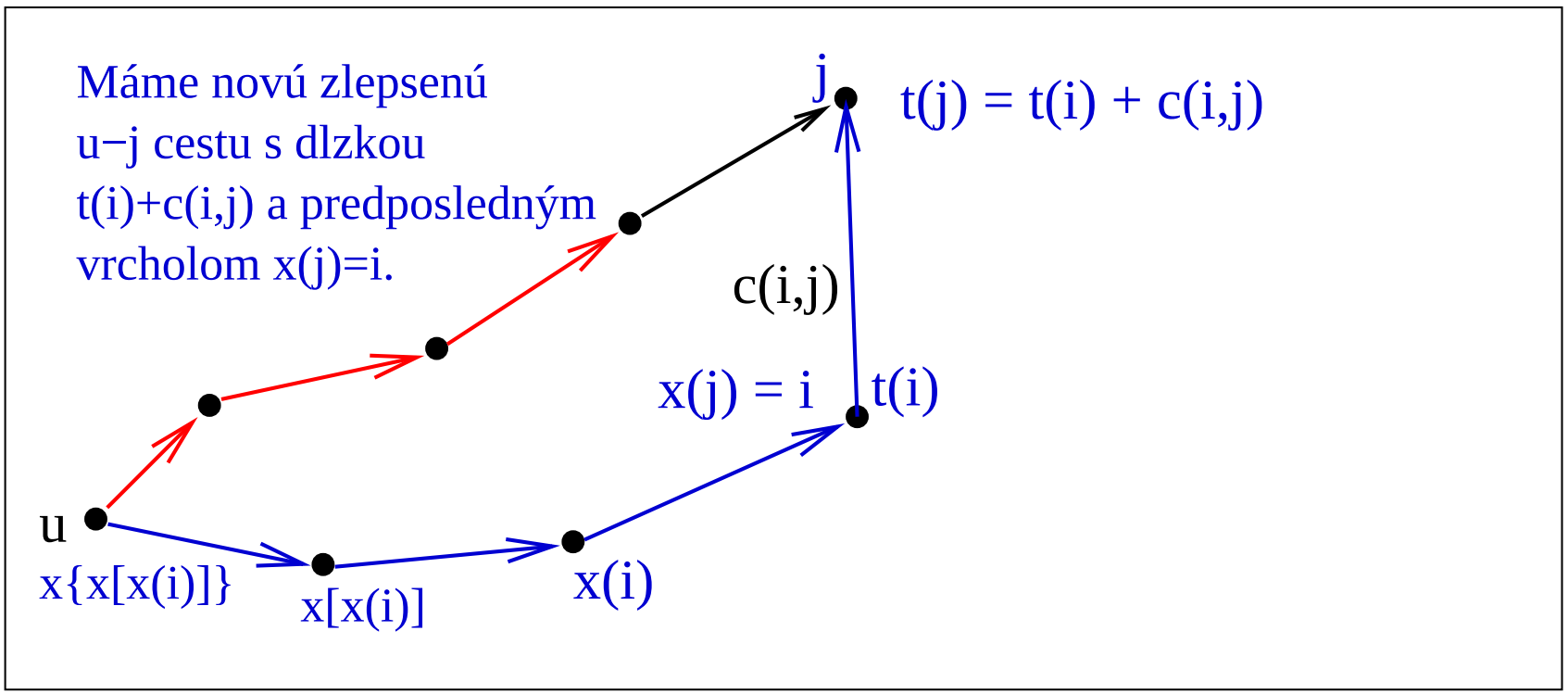
Fundamental algorithm will stop after a finite number of steps for arbitrary digraph $\overrightarrow{G} = (V, H, c)$ with nonegative arc weight.



Principle of fundamental algorithm

Theorem

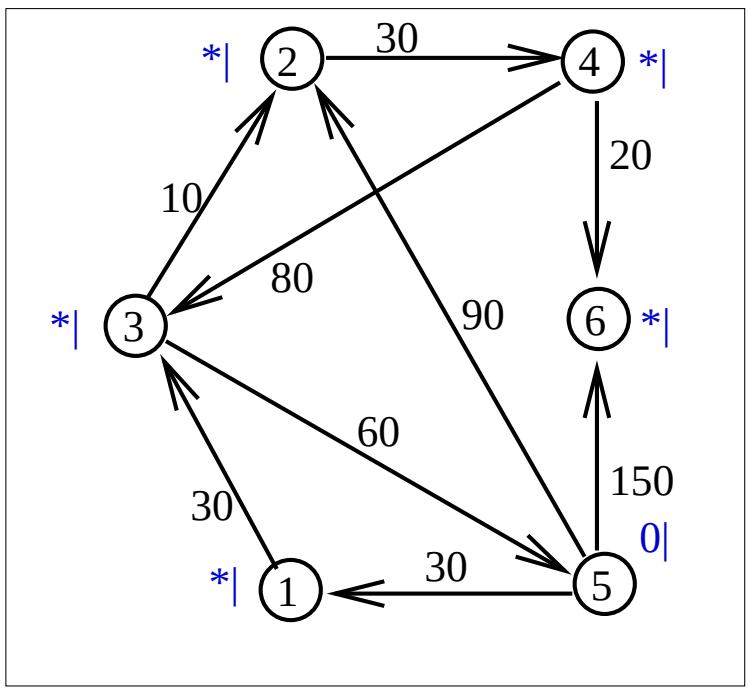
Fundamental algorithm will stop after a finite number of steps for arbitrary digraph $\overrightarrow{G} = (V, H, c)$ *with nonegative arc weight.*



Example. Searching for all shortest paths for vertex 5.

Arc weight table for digraph $\overrightarrow{G}$

h	$(1,3)$	$(2,4)$	$(3,2)$	$(3,5)$	$(4,3)$	$(4,6)$	$(5,1)$	$(5,2)$	$(5,6)$
$c(h)$	30	30	10	60	80	20	30	90	150

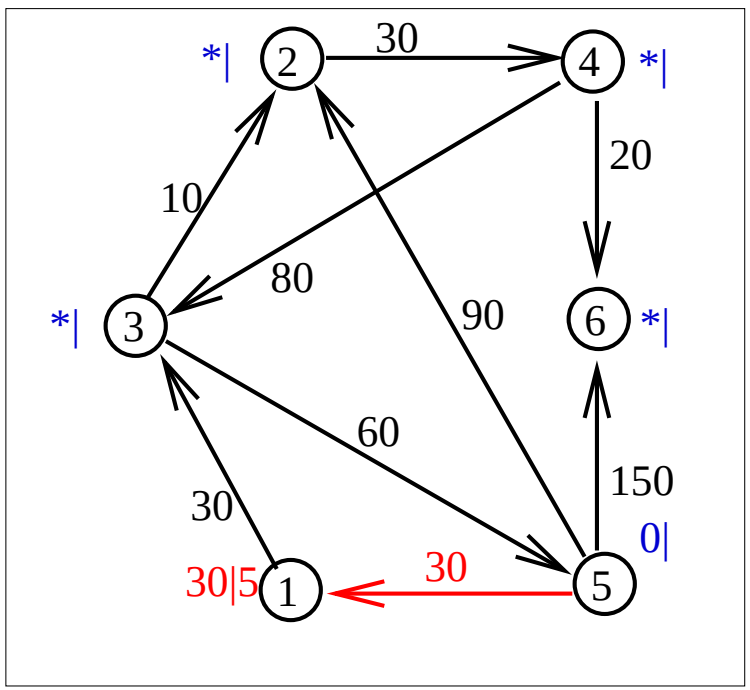


$h=(i,j)$	$t(i)$	c(h)	1	2	3	4	5	6
			$t(v)\|x(v)$					
-			$\infty\|$	$\infty\|$	$\infty\|$	$\infty\|$	$0\|$	$\infty\|$
$(5,1)$	0	30	$30\|5$					
$(5,2)$	0	90		$90\|5$				
$(5,6)$	0	150						$150\|5$
$(1,3)$	30	30			$60\|1$			
$(2,4)$	90	30				$120\|2$		
$(3,2)$	60	10		$70\|3$				
$(4,6)$	120	20						$140\|4$
$(2,4)$	70	30				$100\|2$		
$(4,6)$	100	20						$120\|4$

Example. Searching for all shortest paths for vertex 5.

Arc weight table for digraph $\overrightarrow{G}$

h	$(1,3)$	$(2,4)$	$(3,2)$	$(3,5)$	$(4,3)$	$(4,6)$	$(5,1)$	$(5,2)$	$(5,6)$
$c(h)$	30	30	10	60	80	20	30	90	150

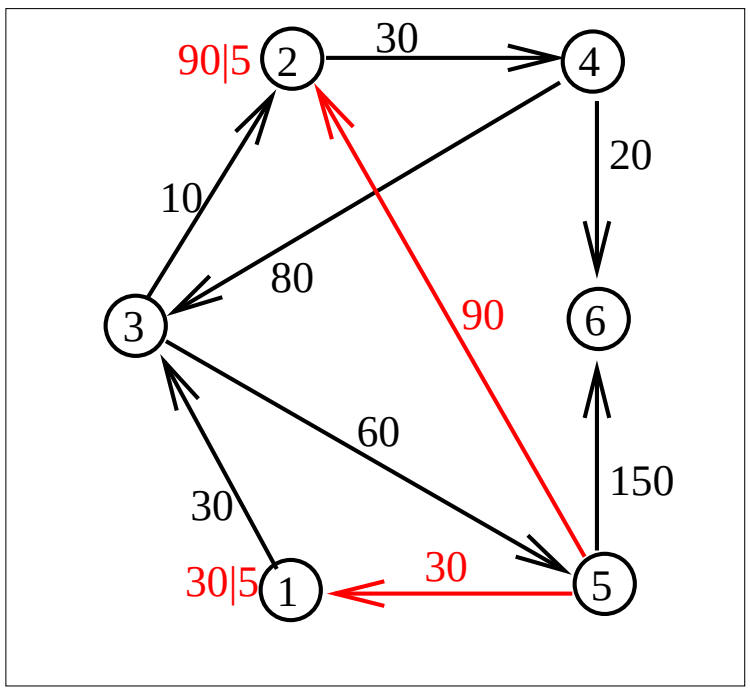


$h=(i,j)$	$t(i)$	c(h)	1	2	3	4	5	6
			$t(v)\|x(v)$					
-			$\infty\|$	$\infty\|$	$\infty\|$	$\infty\|$	$0\|$	$\infty\|$
$(5,1)$	0	30	$30\|5$					
$(5,2)$	0	90		$90\|5$				
$(5,6)$	0	150						$150\|5$
$(1,3)$	30	30			$60\|1$			
$(2,4)$	90	30				$120\|2$		
$(3,2)$	60	10		$70\|3$				
$(4,6)$	120	20						$140\|4$
$(2,4)$	70	30				$100\|2$		
$(4,6)$	100	20						$120\|4$

Example. Searching for all shortest paths for vertex 5.

Arc weight table for digraph $\overrightarrow{G}$

h	$(1,3)$	$(2,4)$	$(3,2)$	$(3,5)$	$(4,3)$	$(4,6)$	$(5,1)$	$(5,2)$	$(5,6)$
$c(h)$	30	30	10	60	80	20	30	90	150

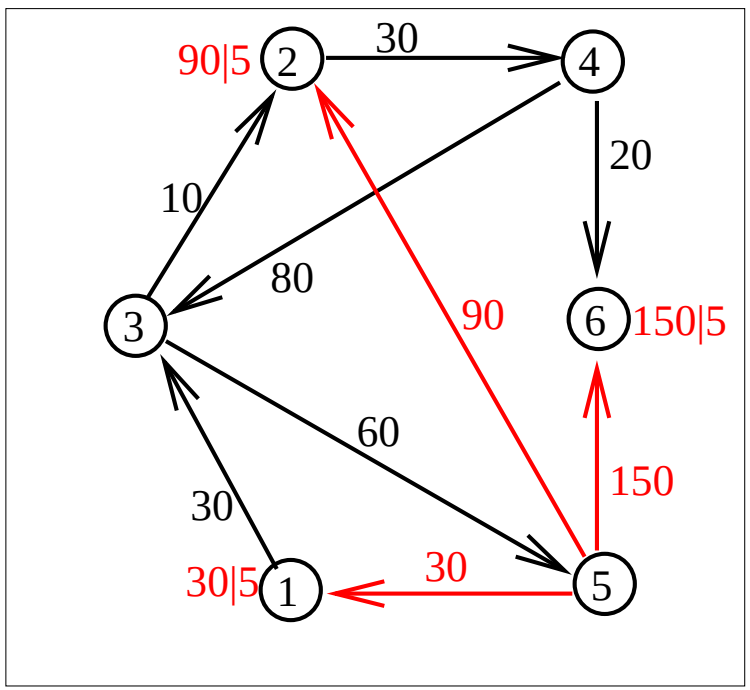


$h=(i,j)$	$t(i)$	c(h)	1	2	3	4	5	6
			$t(v)\|x(v)$					
-			$\infty\|$	$\infty\|$	$\infty\|$	$\infty\|$	$0\|$	$\infty\|$
$(5,1)$	0	30	$30\|5$					
$(5,2)$	0	90		$90\|5$				
$(5,6)$	0	150						$150\|5$
$(1,3)$	30	30			$60\|1$			
$(2,4)$	90	30				$120\|2$		
$(3,2)$	60	10		$70\|3$				
$(4,6)$	120	20						$140\|4$
$(2,4)$	70	30				$100\|2$		
$(4,6)$	100	20						$120\|4$

Example. Searching for all shortest paths for vertex 5.

Arc weight table for digraph $\overrightarrow{G}$

h	$(1,3)$	$(2,4)$	$(3,2)$	$(3,5)$	$(4,3)$	$(4,6)$	$(5,1)$	$(5,2)$	$(5,6)$
$c(h)$	30	30	10	60	80	20	30	90	150

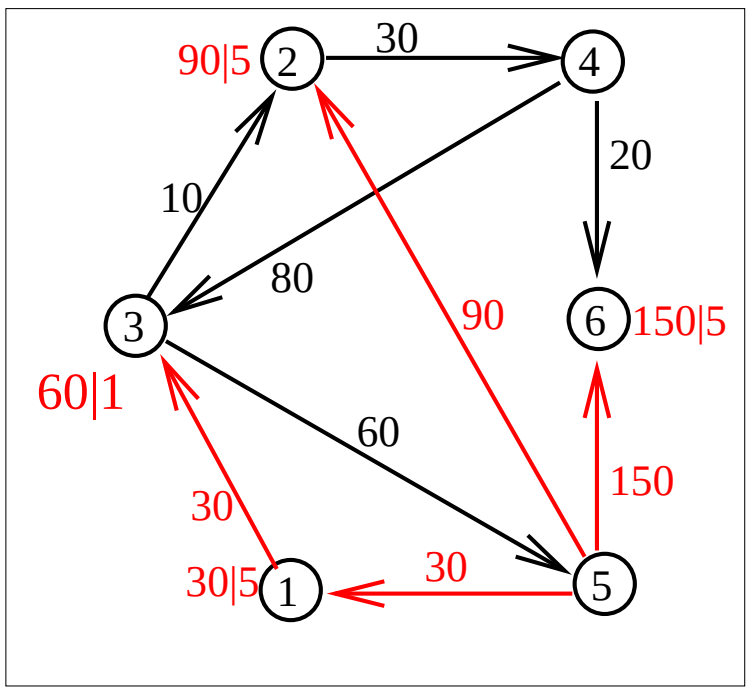


$h=(i,j)$	$t(i)$	c(h)	1	2	3	4	5	6
			$t(v)\|x(v)$					
-			$\infty\|$	$\infty\|$	$\infty\|$	$\infty\|$	$0\|$	$\infty\|$
$(5,1)$	0	30	$30\|5$					
$(5,2)$	0	90		$90\|5$				
$(5,6)$	0	150						$150\|5$
$(1,3)$	30	30			$60\|1$			
$(2,4)$	90	30				$120\|2$		
$(3,2)$	60	10		$70\|3$				
$(4,6)$	120	20						$140\|4$
$(2,4)$	70	30				$100\|2$		
$(4,6)$	100	20						$120\|4$

Example. Searching for all shortest paths for vertex 5.

Arc weight table for digraph $\overrightarrow{G}$

h	$(1,3)$	$(2,4)$	$(3,2)$	$(3,5)$	$(4,3)$	$(4,6)$	$(5,1)$	$(5,2)$	$(5,6)$
$c(h)$	30	30	10	60	80	20	30	90	150

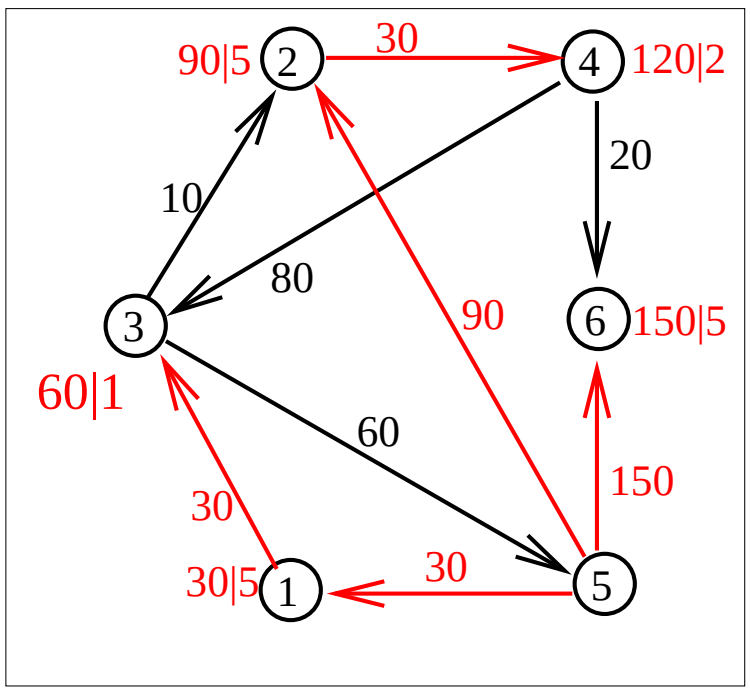


$h=(i,j)$	$t(i)$	c(h)	1	2	3	4	5	6
			$t(v)\|x(v)$					
-			$\infty\|$	$\infty\|$	$\infty\|$	$\infty\|$	$0\|$	$\infty\|$
$(5,1)$	0	30	$30\|5$					
$(5,2)$	0	90		$90\|5$				
$(5,6)$	0	150						$150\|5$
$(1,3)$	30	30			$60\|1$			
$(2,4)$	90	30				$120\|2$		
$(3,2)$	60	10		$70\|3$				
$(4,6)$	120	20						$140\|4$
$(2,4)$	70	30				$100\|2$		
$(4,6)$	100	20						$120\|4$

Example. Searching for all shortest paths for vertex 5.

Arc weight table for digraph $\overrightarrow{G}$

h	$(1,3)$	$(2,4)$	$(3,2)$	$(3,5)$	$(4,3)$	$(4,6)$	$(5,1)$	$(5,2)$	$(5,6)$
$c(h)$	30	30	10	60	80	20	30	90	150

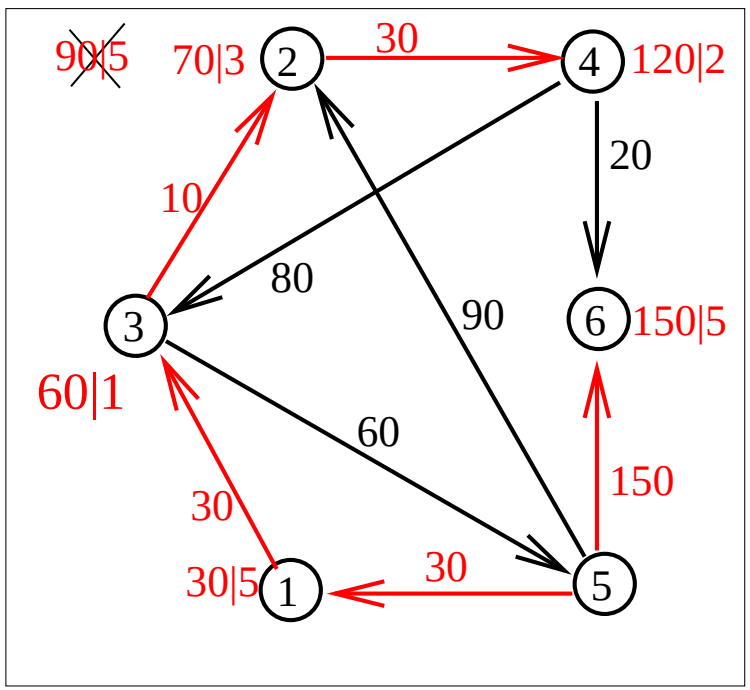


$h=(i,j)$	$t(i)$	c(h)	1	2	3	4	5	6
			$t(v)\|x(v)$					
-			$\infty\|$	$\infty\|$	$\infty\|$	$\infty\|$	$0\|$	$\infty\|$
$(5,1)$	0	30	$30\|5$					
$(5,2)$	0	90		$90\|5$				
$(5,6)$	0	150						$150\|5$
$(1,3)$	30	30			$60\|1$			
$(2,4)$	90	30				$120\|2$		
$(3,2)$	60	10		$70\|3$				
$(4,6)$	120	20						$140\|4$
$(2,4)$	70	30				$100\|2$		
$(4,6)$	100	20						$120\|4$

Example. Searching for all shortest paths for vertex 5.

Arc weight table for digraph $\overrightarrow{G}$

h	$(1,3)$	$(2,4)$	$(3,2)$	$(3,5)$	$(4,3)$	$(4,6)$	$(5,1)$	$(5,2)$	$(5,6)$
$c(h)$	30	30	10	60	80	20	30	90	150

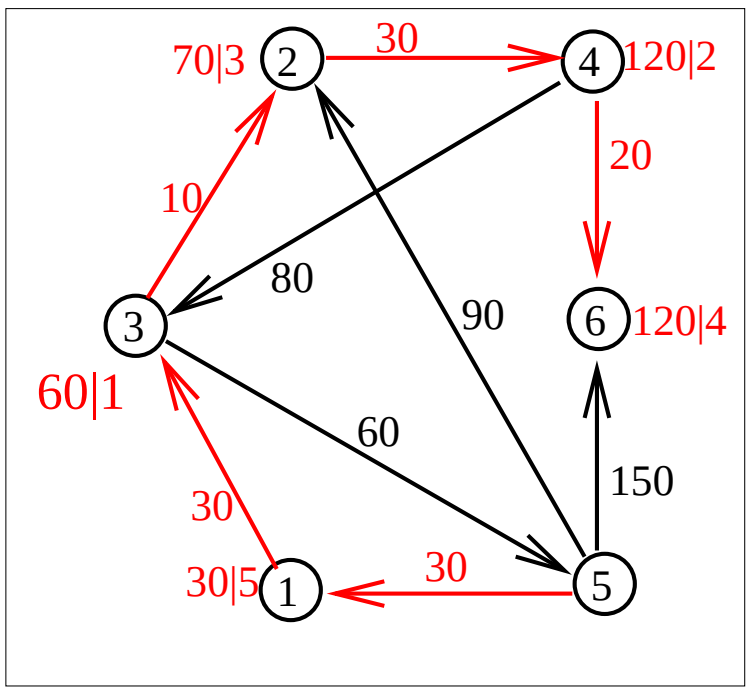


$h=(i,j)$	$t(i)$	c(h)	1	2	3	4	5	6
			$t(v)\|x(v)$					
-			$\infty\|$	$\infty\|$	$\infty\|$	$\infty\|$	$0\|$	$\infty\|$
$(5,1)$	0	30	$30\|5$					
$(5,2)$	0	90		$90\|5$				
$(5,6)$	0	150						$150\|5$
$(1,3)$	30	30			$60\|1$			
$(2,4)$	90	30				$120\|2$		
$(3,2)$	60	10		$70\|3$				
$(4,6)$	120	20						$140\|4$
$(2,4)$	70	30				$100\|2$		
$(4,6)$	100	20						$120\|4$

Example. Searching for all shortest paths for vertex 5.

Arc weight table for digraph $\overrightarrow{G}$

h	$(1,3)$	$(2,4)$	$(3,2)$	$(3,5)$	$(4,3)$	$(4,6)$	$(5,1)$	$(5,2)$	$(5,6)$
$c(h)$	30	30	10	60	80	20	30	90	150

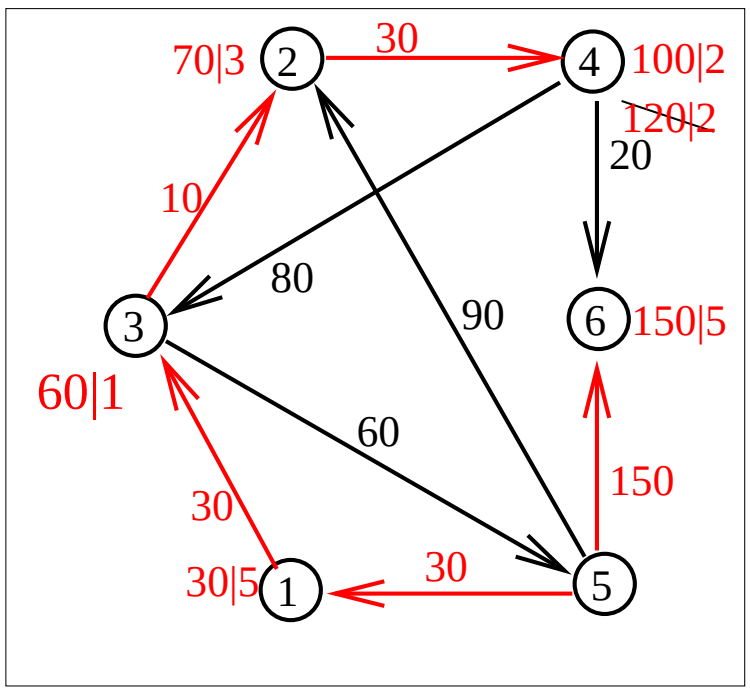


$h=(i,j)$	$t(i)$	c(h)	1	2	3	4	5	6
			$t(v)\|x(v)$					
-			$\infty\|$	$\infty\|$	$\infty\|$	$\infty\|$	$0\|$	$\infty\|$
$(5,1)$	0	30	$30\|5$					
$(5,2)$	0	90		$90\|5$				
$(5,6)$	0	150						$150\|5$
$(1,3)$	30	30			$60\|1$			
$(2,4)$	90	30				$120\|2$		
$(3,2)$	60	10		$70\|3$				
$(4,6)$	120	20						$140\|4$
$(2,4)$	70	30				$100\|2$		
$(4,6)$	100	20						$120\|4$

Example. Searching for all shortest paths for vertex 5.

Arc weight table for digraph $\overrightarrow{G}$

h	$(1,3)$	$(2,4)$	$(3,2)$	$(3,5)$	$(4,3)$	$(4,6)$	$(5,1)$	$(5,2)$	$(5,6)$
$c(h)$	30	30	10	60	80	20	30	90	150

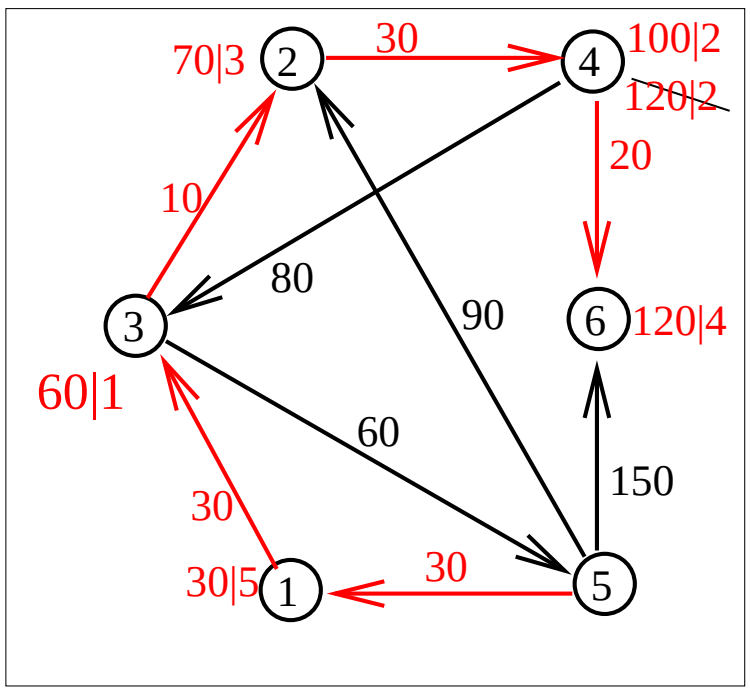


$h=(i,j)$	$t(i)$	c(h)	1	2	3	4	5	6
			$t(v)\|x(v)$					
-			$\infty\|$	$\infty\|$	$\infty\|$	$\infty\|$	$0\|$	$\infty\|$
$(5,1)$	0	30	$30\|5$					
$(5,2)$	0	90		$90\|5$				
$(5,6)$	0	150						$150\|5$
$(1,3)$	30	30			$60\|1$			
$(2,4)$	90	30				$120\|2$		
$(3,2)$	60	10		$70\|3$				
$(4,6)$	120	20						$140\|4$
$(2,4)$	70	30				$100\|2$		
$(4,6)$	100	20						$120\|4$

Example. Searching for all shortest paths for vertex 5.

Arc weight table for digraph $\overrightarrow{G}$

h	$(1,3)$	$(2,4)$	$(3,2)$	$(3,5)$	$(4,3)$	$(4,6)$	$(5,1)$	$(5,2)$	$(5,6)$
$c(h)$	30	30	10	60	80	20	30	90	150



$h=(i,j)$	$t(i)$	c(h)	1	2	3	4	5	6
			$t(v)\|x(v)$					
-			$\infty\|$	$\infty\|$	$\infty\|$	$\infty\|$	$0\|$	$\infty\|$
$(5,1)$	0	30	30|5					
$(5,2)$	0	90		90|5				
$(5,6)$	0	150						150|5
$(1,3)$	30	30			60|1			
$(2,4)$	90	30				120|2		
$(3,2)$	60	10		70|3				
$(4,6)$	120	20						140|4
$(2,4)$	70	30				100|2		
$(4,6)$	100	20						120|4

Algorithm

Dijkstra's point-to-point shortest path algorithm

This algorithm will find shortest u–v directed path from a fixed vertex $u \in V$ into a fixed vertex $v \in V$ in an arc weighted digraph $\overrightarrow{G} = (V, H, c)$ with nonnegative arc weight $c(h)$ (and where $0 \notin V$).

- **Step 1.** *Initialization.*
  Assign two labels $t(i)$ a $x(i)$ for every vertex $i \in V$.
  {Label $t(i)$ is upper estimate of the till now the best found u–i path and $x(i)$ is it's last but one vertex.}

  Every label $t(i)$ can be temporary – it can change in the course of computing, or it can be final – it contains the length of shortest u–i path and it can not change.

  Set $t(u) := 0$, $t(i) := \infty$ for $i \in V$, $i \neq u$ and $x(i) := 0$ for all $i \in V$.

  Set pivot vertex $r := u$ and declare label $t(\ )$ assigned to pivot vertex $r = u$ as final. Declare other labels as temporary.

Algorithm

Dijkstra's point-to-point shortest path algorithm

This algorithm will find shortest u–v directed path from a fixed vertex $u \in V$ into a fixed vertex $v \in V$ in an arc weighted digraph $\overrightarrow{G} = (V, H, c)$ with nonnegative arc weight $c(h)$ (and where $0 \notin V$).

- **Step 1.** *Initialization.*
  Assign two labels $t(i)$ a $x(i)$ for every vertex $i \in V$.
  {Label $t(i)$ is upper estimate of the till now the best found u–i path and $x(i)$ is it's last but one vertex.}

  Every label $t(i)$ can be temporary – it can change in the course of computing, or it can be final – it contains the length of shortest u–i path and it can not change.

  Set $t(u) := 0$, $t(i) := \infty$ for $i \in V$, $i \neq u$ and $x(i) := 0$ for all $i \in V$.

  Set pivot vertex $r := u$ and declare label $t(\ )$ assigned to pivot vertex $r = u$ as final. Declare other labels as temporary.

Algorithm

Dijkstra's point-to-point shortest path algorithm

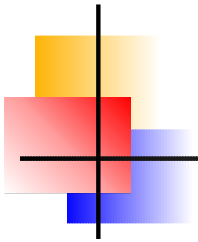
This algorithm will find shortest u–v directed path from a fixed vertex $u \in V$ into a fixed vertex $v \in V$ in an arc weighted digraph $\overrightarrow{G} = (V, H, c)$ with nonnegative arc weight $c(h)$ (and where $0 \notin V$).

- **Step 1.** *Initialization.*
  Assign two labels $t(i)$ a $x(i)$ for every vertex $i \in V$.
  {Label $t(i)$ is upper estimate of the till now the best found u–i path and $x(i)$ is it's last but one vertex.}

  Every label $t(i)$ can be temporary – it can change in the course of computing, or it can be final – it contains the length of shortest u–i path and it can not change.

  Set $t(u) := 0$, $t(i) := \infty$ for $i \in V$, $i \neq u$ and $x(i) := 0$ for all $i \in V$.

  Set pivot vertex $r := u$ and declare label $t(\ )$ assigned to pivot vertex $r = u$ as final. Declare other labels as temporary.



Dijkstra's algorithm

Algorithm (- continued)

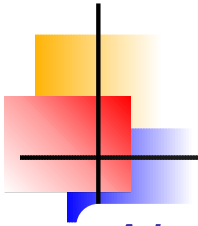
- **Step 2.** *If $r = v$, STOP.*

  If $t(v) < \infty$, then the label $t(v)$ is equal to the length of the shortest u–v path, which can be constructed in reverse order from vertex z v using ponters $x(i)$.

  Otherwise for all arcs $(r, j))$ such that $(r, j) \in H^+(r)$ do:

  If $t(j) > t(r) + c(r, j)$, then $t(j) := t(r) + c(r, j)$, $x(j) := r$.

  Keep all chainged labels as temporary.



Dijkstra's algorithm

Algorithm (- continued)

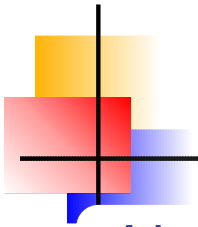
- **Step 3.** *Find vertex i with minimal temporary label $t(i)$.*

  Declare the label $t(i)$ as finite and set the new pivot $r := i$.
  {If there are more vertices having minimal label $t(\ )$ equal to $t(i)$, only one label can be declared as minimal and chosen as pivot. Other vertices with minimum label $t(\ )$ will be declared as final and chosen as pivots one by one in following steps of computation.}
  GOTO Step 2.

♣

Remark

If we change the stop condition in Step 2. of algorithm as follows:
If all labels $t(\ )$ are final, STOP,
we get a point-to-all version of Dijkstra's algorithm which computes all shortest paths from fixed vertex u to all reachable vertices.



Dijkstra's algorithm

Algorithm (- continued)

- **Step 3.** *Find vertex i with minimal temporary label $t(i)$.*

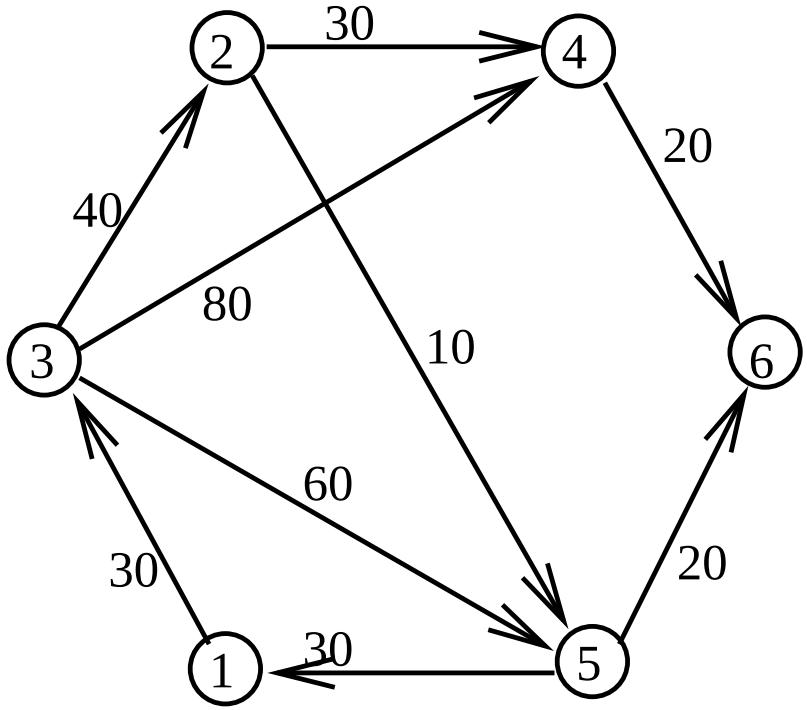
  Declare the label $t(i)$ as finite and set the new pivot $r := i$.
  {If there are more vertices having minimal label $t(\ )$ equal to $t(i)$, only one label can be declared as minimal and chosen as pivot. Other vertices with minimum label $t(\ )$ will be declared as final and chosen as pivots one by one in following steps of computation.}
  GOTO Step 2.

♣

Remark

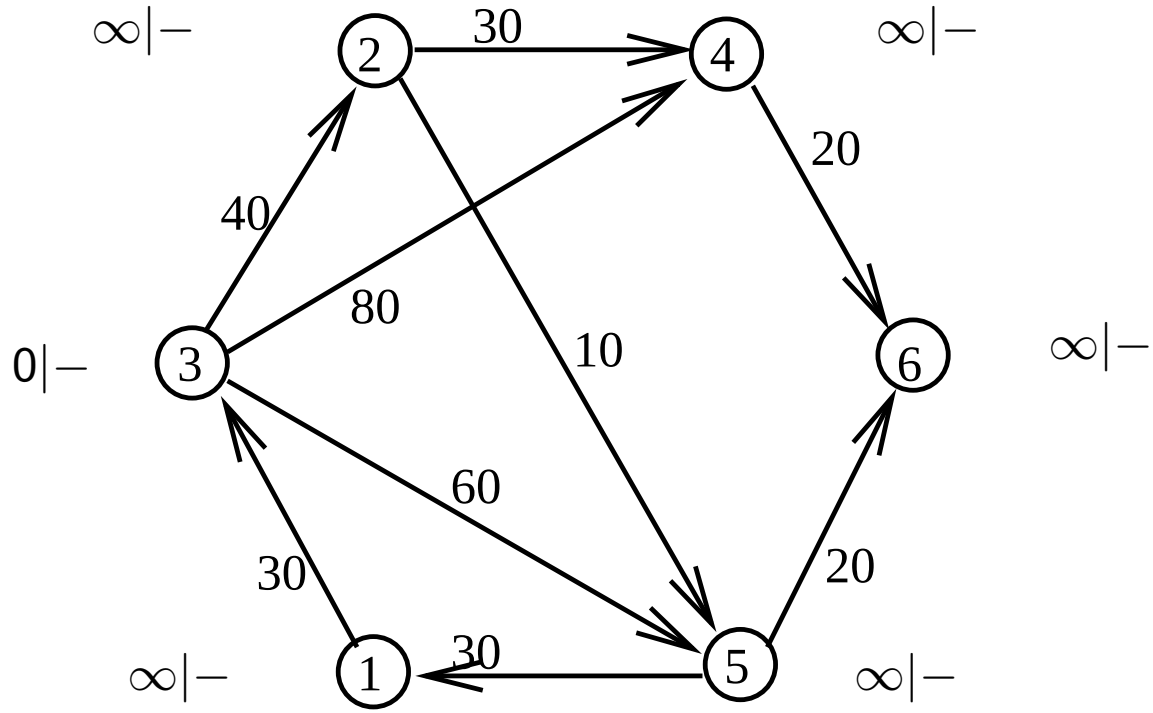
If we change the stop condition in Step 2. of algorithm as follows:
If all labels $t(\ )$ are final, STOP,
we get a point-to-all version of Dijkstra's algorithm which computes all shortest paths from fixed vertex u to all reachable vertices.

Dijkstra's algorithm – Example



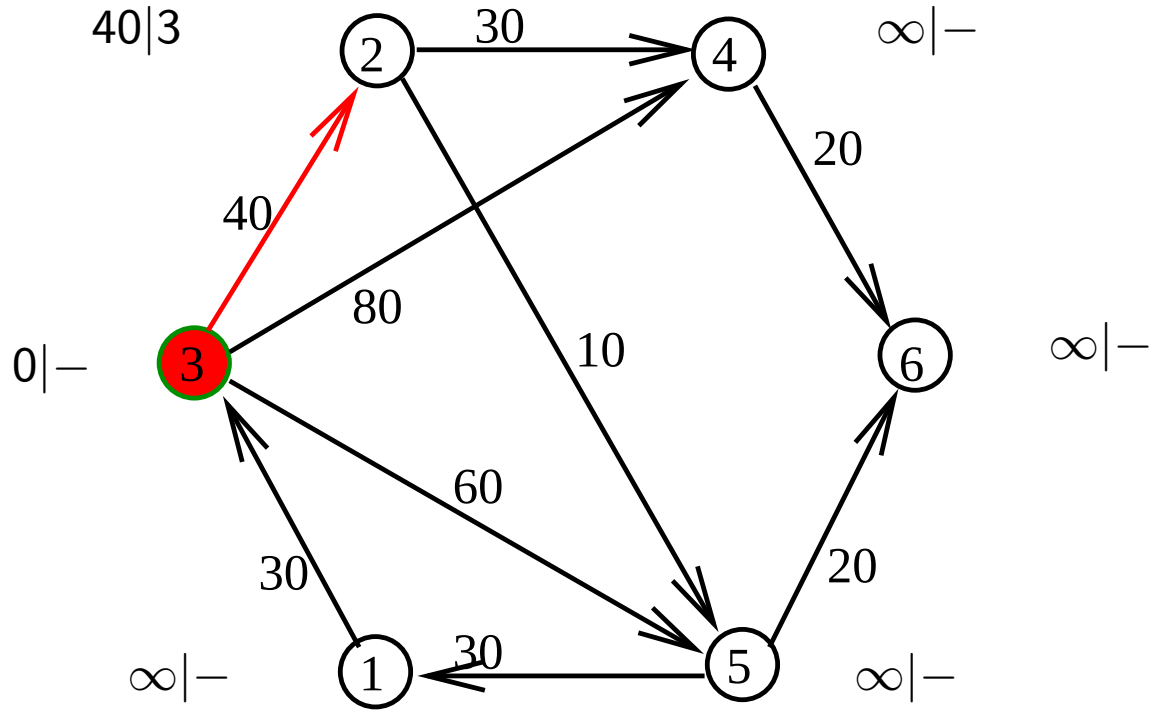
r	$x(r)$	$t(r)$	1	2	3	4	5	6
			$t(v)\|x(v)$					
-	-	-	∞ |	∞ |	**0** |	∞ |	∞ |	∞ |
3	-	0	∞ |	**40 | 3**		80 | 3	60 | 3	∞ |
2	3	40	∞ |			70 | 2	**50 | 2**	∞ |
5	2	50	80 | 5			**70 | 2**		70 | 5
4	2	70	80 | 5					**70 | 5**
6	5	70	**80 | 5**					
1	5	80						

Dijkstra's algorithm – Example



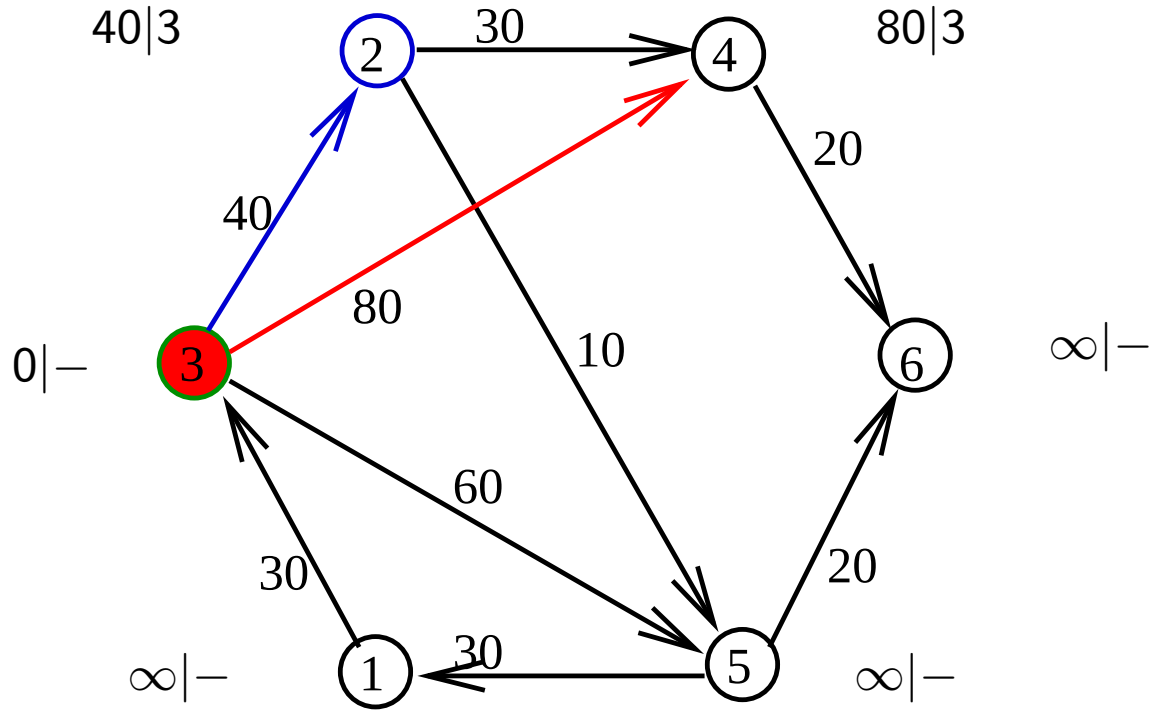
r	$x(r)$	$t(r)$	1	2	3	4	5	6
			$t(v)\|x(v)$					
-	-	-	∞|	∞|	**0**|	∞|	∞|	∞|
3	-	0	∞|	**40|3**		80|3	60|3	∞|
2	3	40	∞|			70|2	**50|2**	∞|
5	2	50	80|5			**70|2**		70|5
4	2	70	80|5					**70|5**
6	5	70	**80|5**					
1	5	80						

Dijkstra's algorithm – Example



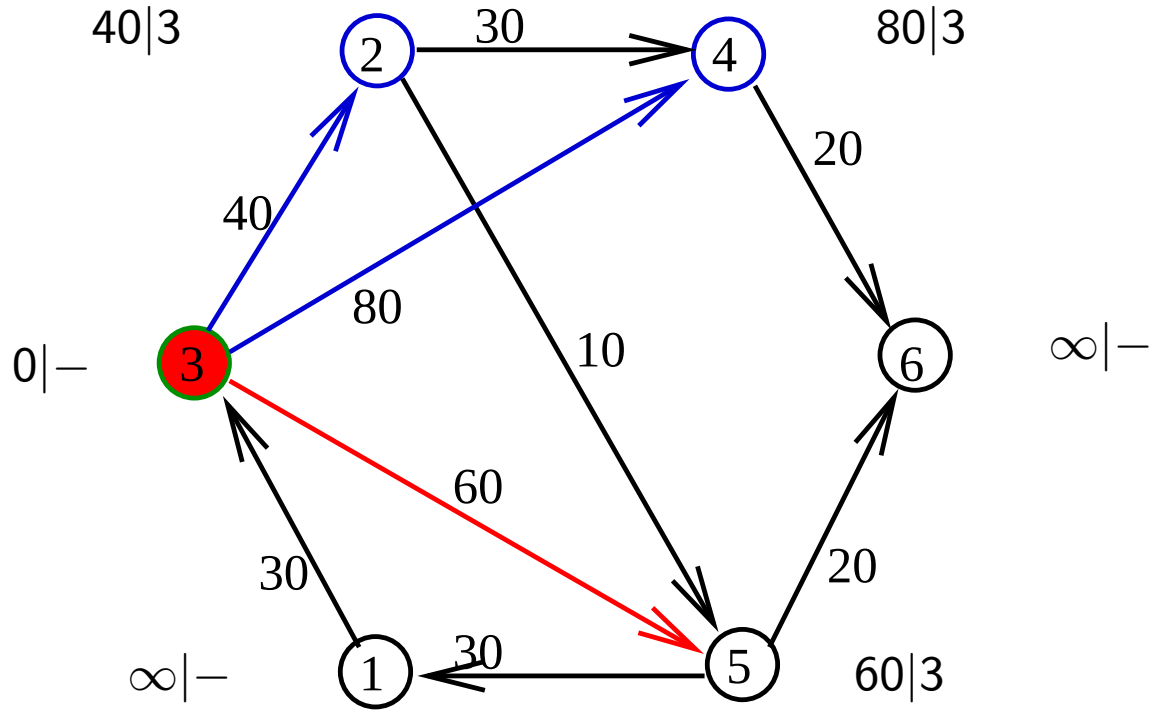
r	$x(r)$	$t(r)$	1	2	3	4	5	6
			$t(v)\|x(v)$					
-	-	-	∞|	∞|	**0**|	∞|	∞|	∞|
3	-	0	∞|	**40|3**		80|3	60|3	∞|
2	3	40	∞|			70|2	**50|2**	∞|
5	2	50	80|5			**70|2**		70|5
4	2	70	80|5					**70|5**
6	5	70	**80|5**					
1	5	80						

Dijkstra's algorithm – Example



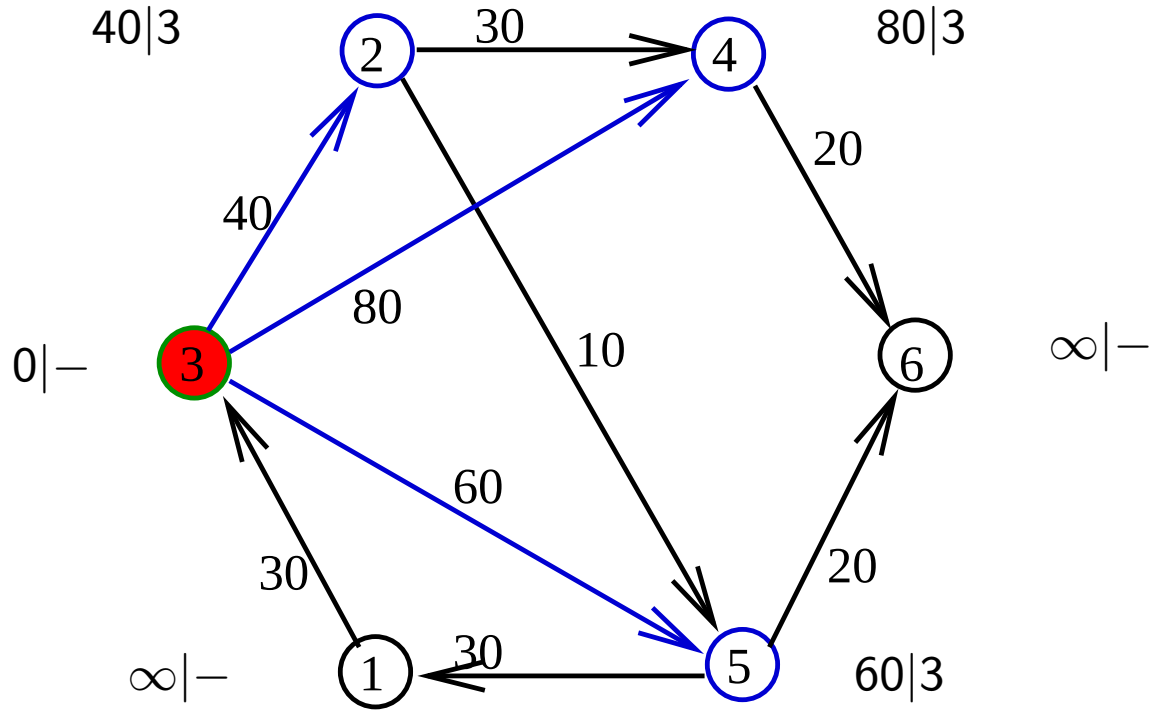
r	$x(r)$	$t(r)$	1	2	3	4	5	6
			$t(v)\|x(v)$					
-	-	-	∞|	∞|	**0**|	∞|	∞|	∞|
3	-	0	∞|	**40|3**		80|3	60|3	∞|
2	3	40	∞|			70|2	**50|2**	∞|
5	2	50	80|5			**70|2**		70|5
4	2	70	80|5					**70|5**
6	5	70	**80|5**					
1	5	80						

Dijkstra's algorithm – Example



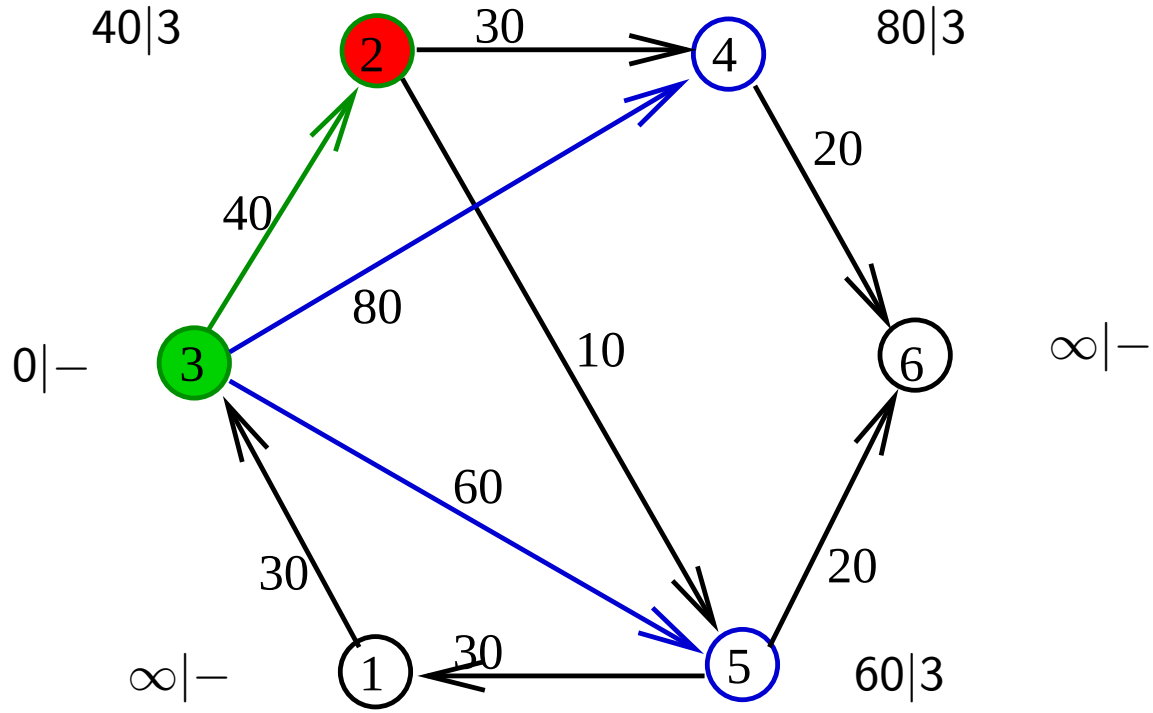
r	$x(r)$	$t(r)$	1	2	3	4	5	6
			$t(v)\|x(v)$					
-	-	-	∞|	∞|	**0**|	∞|	∞|	∞|
3	-	0	∞|	**40|3**		80|3	60|3	∞|
2	3	40	∞|			70|2	**50|2**	∞|
5	2	50	80|5			**70|2**		70|5
4	2	70	80|5					**70|5**
6	5	70	**80|5**					
1	5	80						

Dijkstra's algorithm – Example



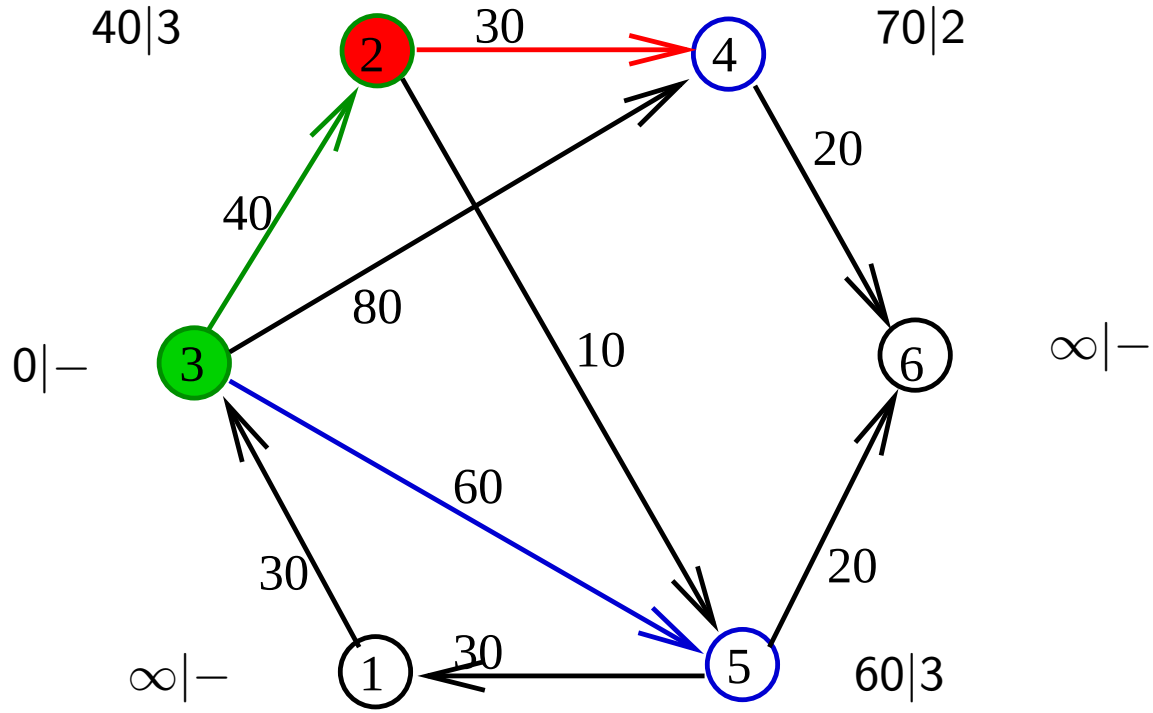
r	$x(r)$	$t(r)$	1	2	3	4	5	6
			$t(v)\|x(v)$					
-	-	-	∞|	∞|	**0**|	∞|	∞|	∞|
3	-	0	∞|	**40|3**		80|3	60|3	∞|
2	3	40	∞|			70|2	**50|2**	∞|
5	2	50	80|5			**70|2**		70|5
4	2	70	80|5					**70|5**
6	5	70	**80|5**					
1	5	80						

Dijkstra's algorithm – Example



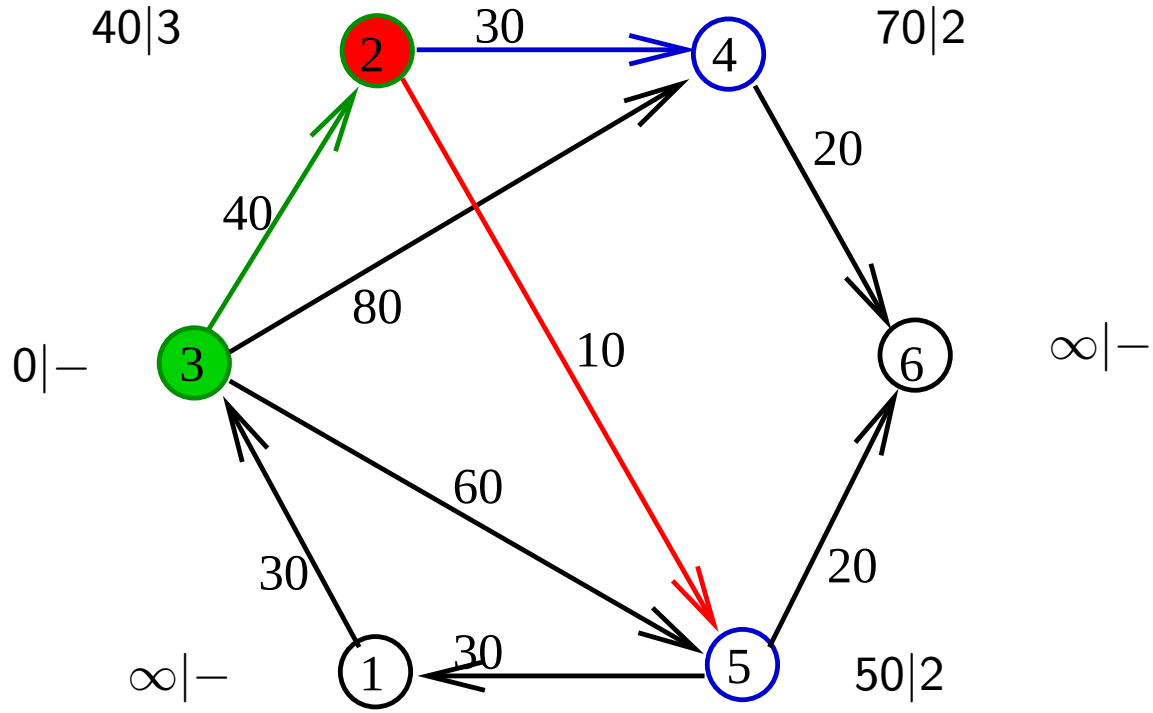
r	$x(r)$	$t(r)$	1	2	3	4	5	6
			$t(v)\|x(v)$					
-	-	-	∞|	∞|	**0**|	∞|	∞|	∞|
3	-	0	∞|	**40|3**		80|3	60|3	∞|
2	3	40	∞|			70|2	**50|2**	∞|
5	2	50	80|5			**70|2**		70|5
4	2	70	80|5					**70|5**
6	5	70	**80|5**					
1	5	80						

Dijkstra's algorithm – Example



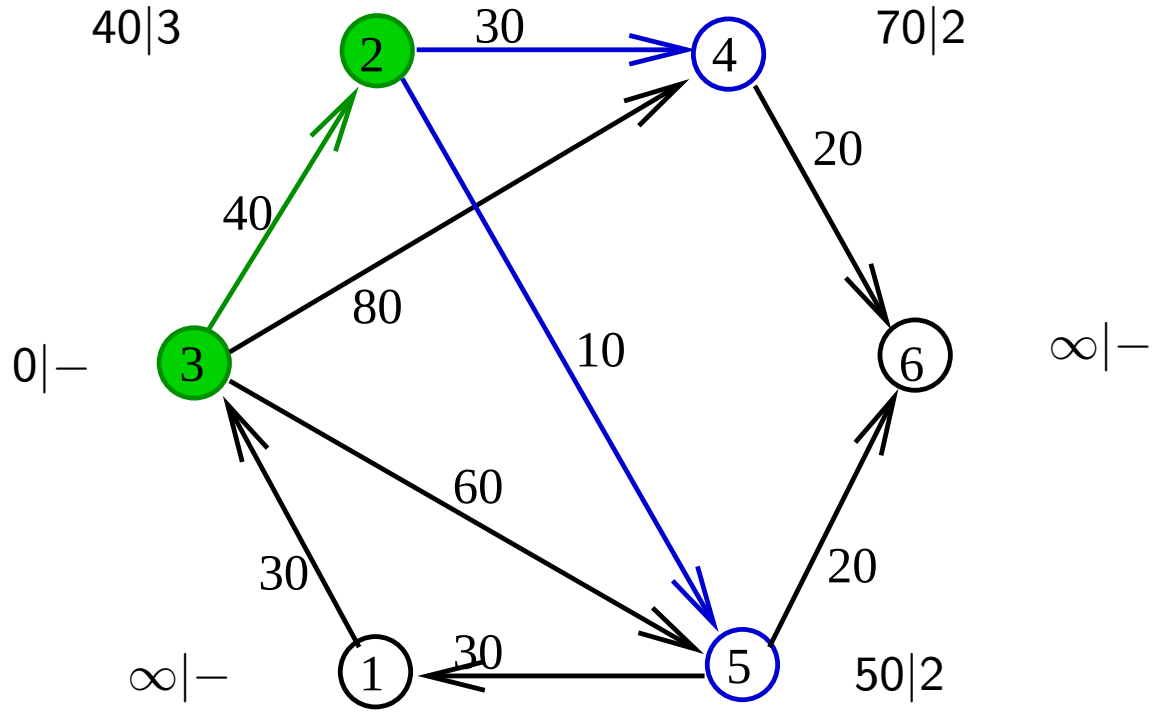
r	$x(r)$	$t(r)$	1	2	3	4	5	6
			$t(v)\|x(v)$					
-	-	-	∞|	∞|	**0**|	∞|	∞|	∞|
3	-	0	∞|	**40**|**3**		80|3	60|3	∞|
2	3	40	∞|			70|2	**50**|**2**	∞|
5	2	50	80|5			**70**|**2**		70|5
4	2	70	80|5					**70**|**5**
6	5	70	**80**|**5**					
1	5	80						

Dijkstra's algorithm – Example



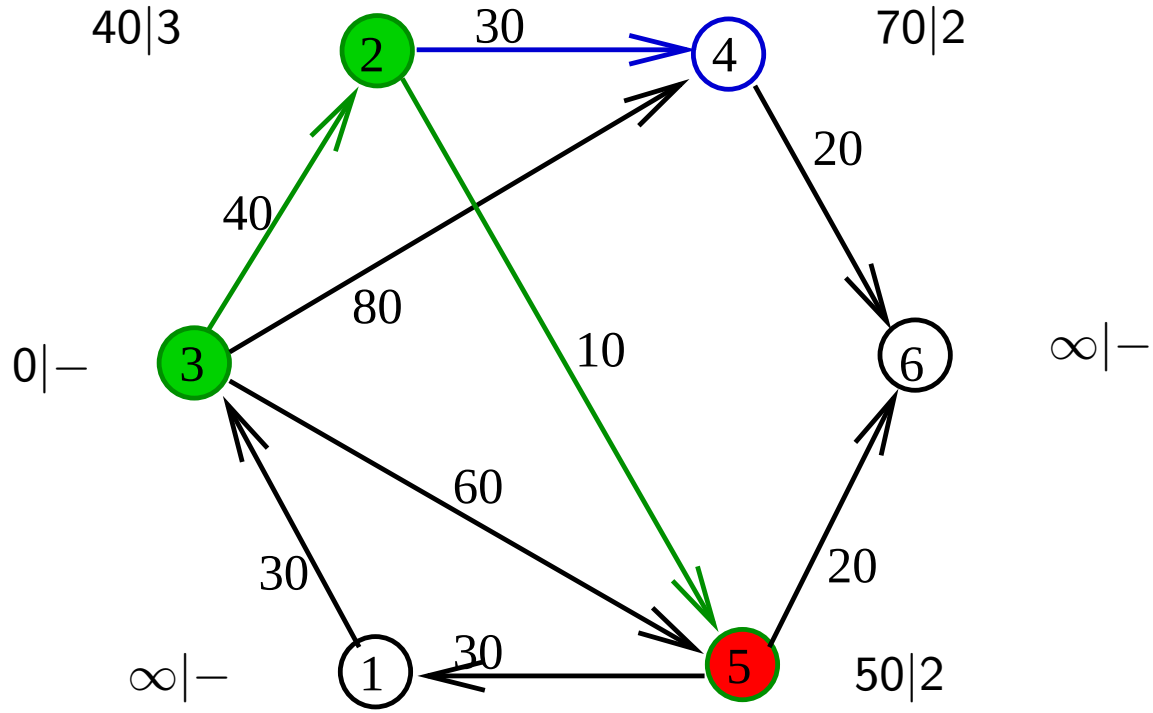
r	$x(r)$	$t(r)$	1	2	3	4	5	6
			$t(v)\|x(v)$					
-	-	-	∞|	∞|	**0**|	∞|	∞|	∞|
3	-	0	∞|	**40|3**		80|3	60|3	∞|
2	3	40	∞|			70|2	**50|2**	∞|
5	2	50	80|5			**70|2**		70|5
4	2	70	80|5					**70|5**
6	5	70	**80|5**					
1	5	80						

Dijkstra's algorithm – Example



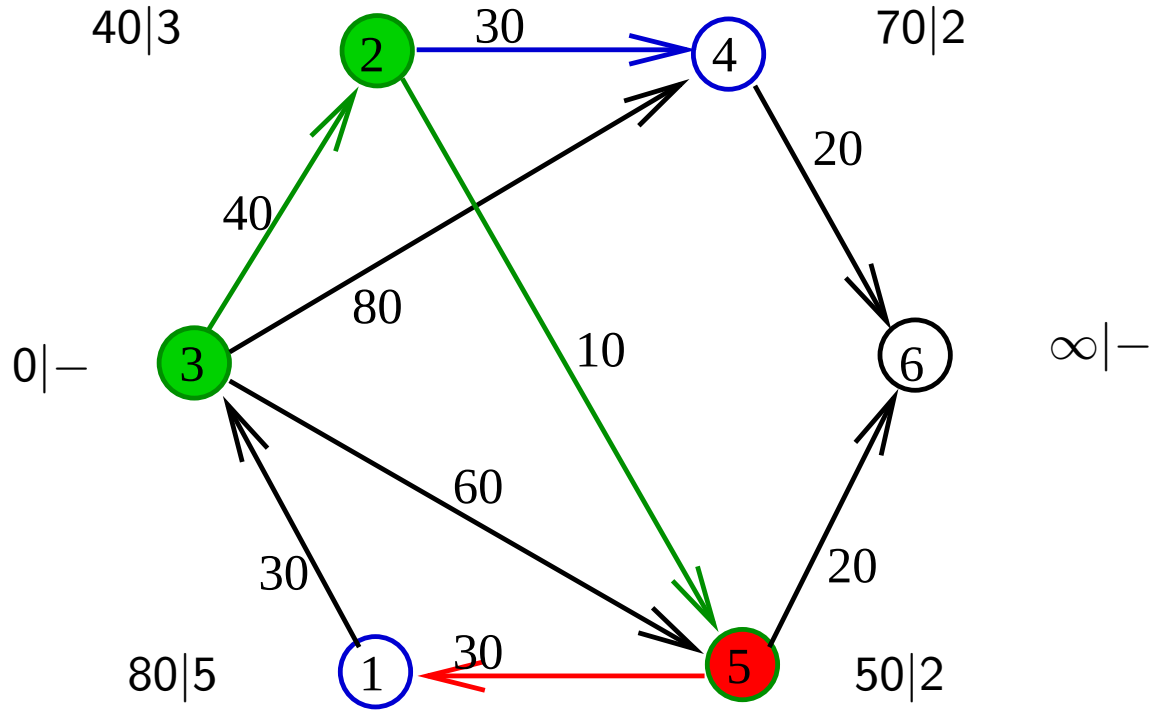
r	$x(r)$	$t(r)$	1	2	3	4	5	6
			$t(v)\|x(v)$					
-	-	-	∞|	∞|	**0**|	∞|	∞|	∞|
3	-	0	∞|	**40|3**		80|3	60|3	∞|
2	3	40	∞|			70|2	**50|2**	∞|
5	2	50	80|5			**70|2**		70|5
4	2	70	80|5					**70|5**
6	5	70	**80|5**					
1	5	80						

Dijkstra's algorithm – Example



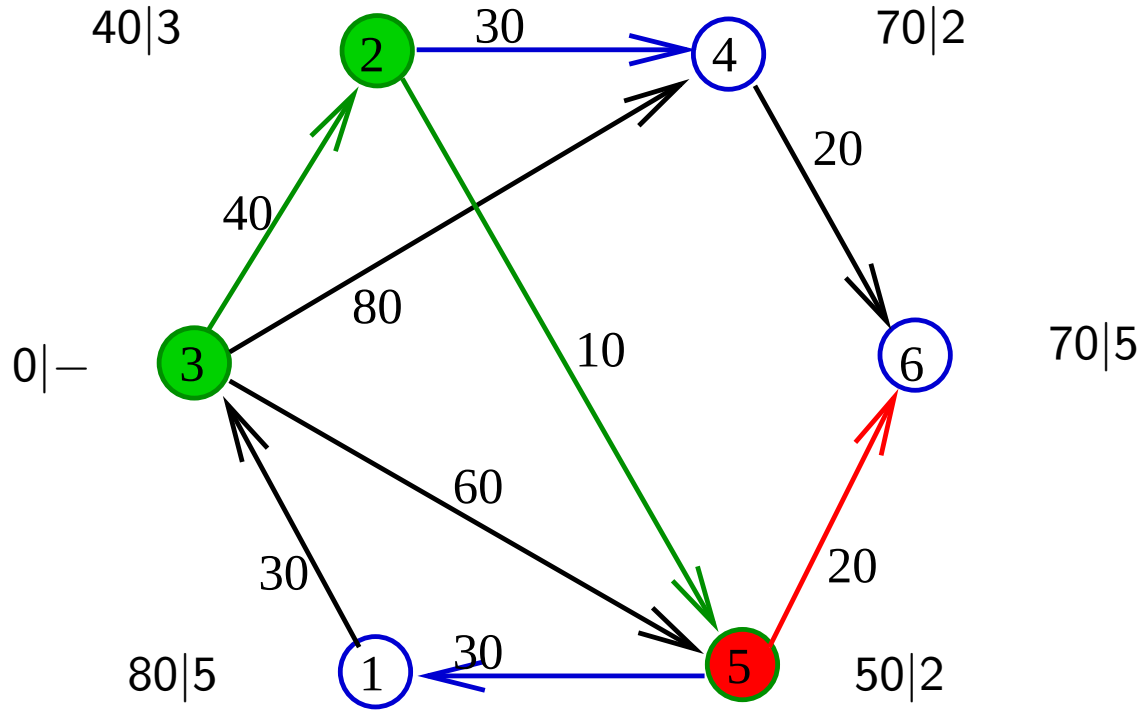
r	$x(r)$	$t(r)$	1	2	3	4	5	6
			$t(v)\|x(v)$					
-	-	-	∞|	∞|	**0**|	∞|	∞|	∞|
3	-	0	∞|	**40**|**3**		80|3	60|3	∞|
2	3	40	∞|			70|2	**50**|**2**	∞|
5	2	50	80|5			**70**|**2**		70|5
4	2	70	80|5					**70**|**5**
6	5	70	**80**|**5**					
1	5	80						

Dijkstra's algorithm – Example



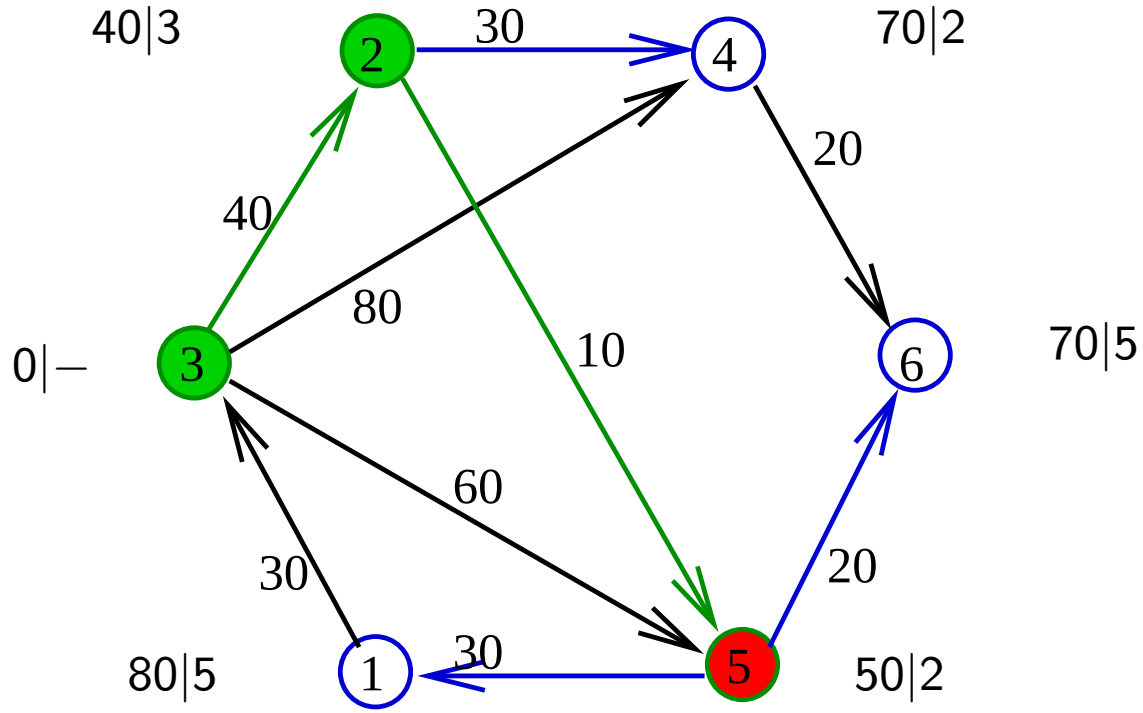
r	$x(r)$	$t(r)$	1	2	3	4	5	6
			$t(v)\|x(v)$					
-	-	-	∞ |	∞ |	**0** |	∞ |	∞ |	∞ |
3	-	0	∞ |	**40|3**		80|3	60|3	∞ |
2	3	40	∞ |			70|2	**50|2**	∞ |
5	2	50	80|5			**70|2**		70|5
4	2	70	80|5					**70|5**
6	5	70	**80|5**					
1	5	80						

Dijkstra's algorithm – Example



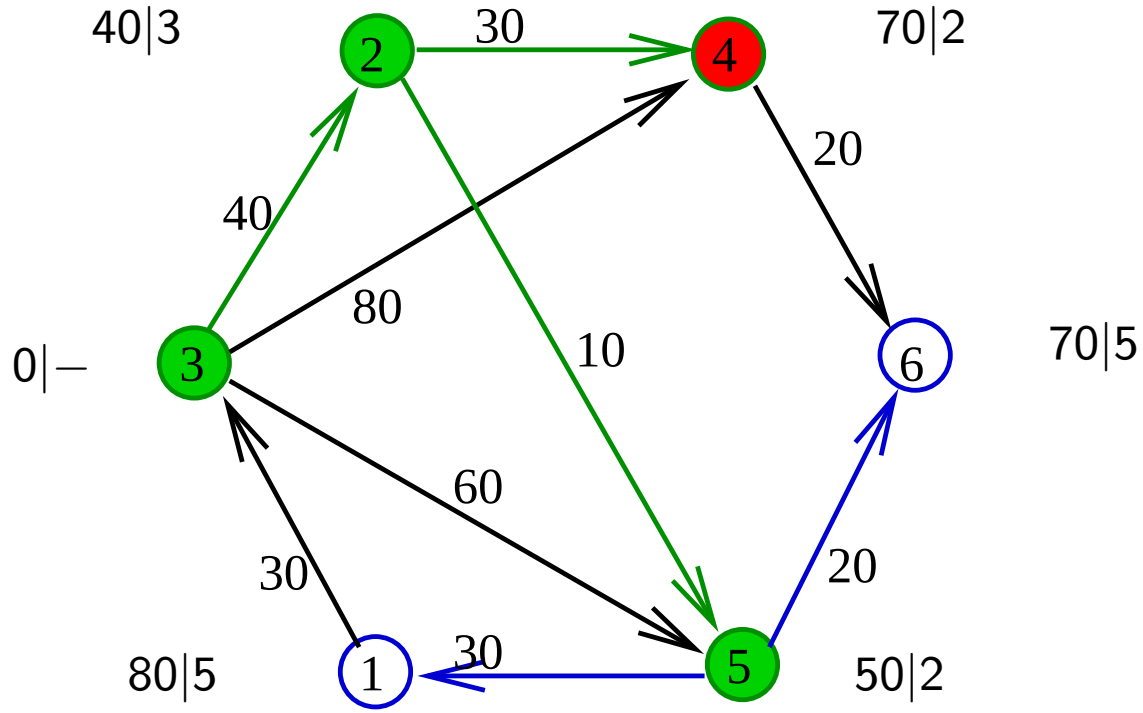
r	$x(r)$	$t(r)$	1	2	3	4	5	6
			$t(v)\|x(v)$					
-	-	-	∞ |	∞ |	**0**|	∞ |	∞ |	∞ |
3	-	0	∞ |	**40**|**3**		80|3	60|3	∞ |
2	3	40	∞ |			70|2	**50**|**2**	∞ |
5	2	50	80|5			**70**|**2**		70|5
4	2	70	80|5					**70**|**5**
6	5	70	**80**|**5**					
1	5	80						

Dijkstra's algorithm – Example



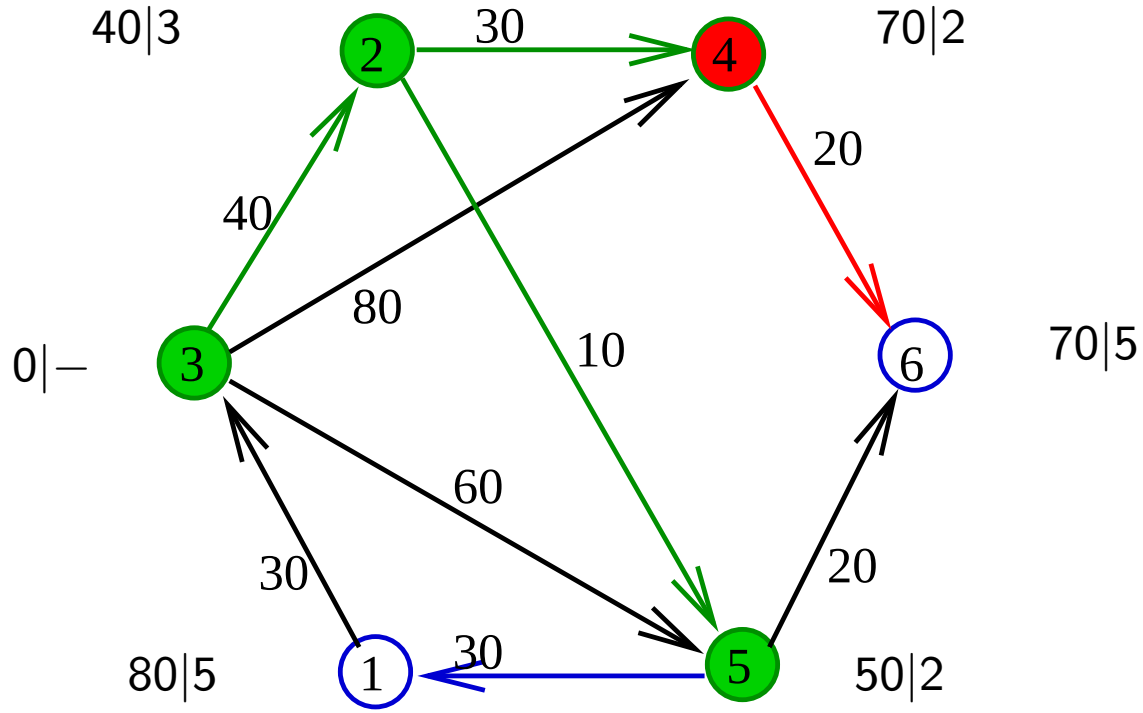
r	$x(r)$	$t(r)$	1	2	3	4	5	6
			$t(v)\|x(v)$					
-	-	-	∞ |	∞ |	**0**|	∞ |	∞ |	∞ |
3	-	0	∞ |	**40**|**3**		80|3	60|3	∞ |
2	3	40	∞ |			70|2	**50**|**2**	∞ |
5	2	50	80|5			**70**|**2**		70|5
4	2	70	80|5					**70**|**5**
6	5	70	**80**|**5**					
1	5	80						

Dijkstra's algorithm – Example



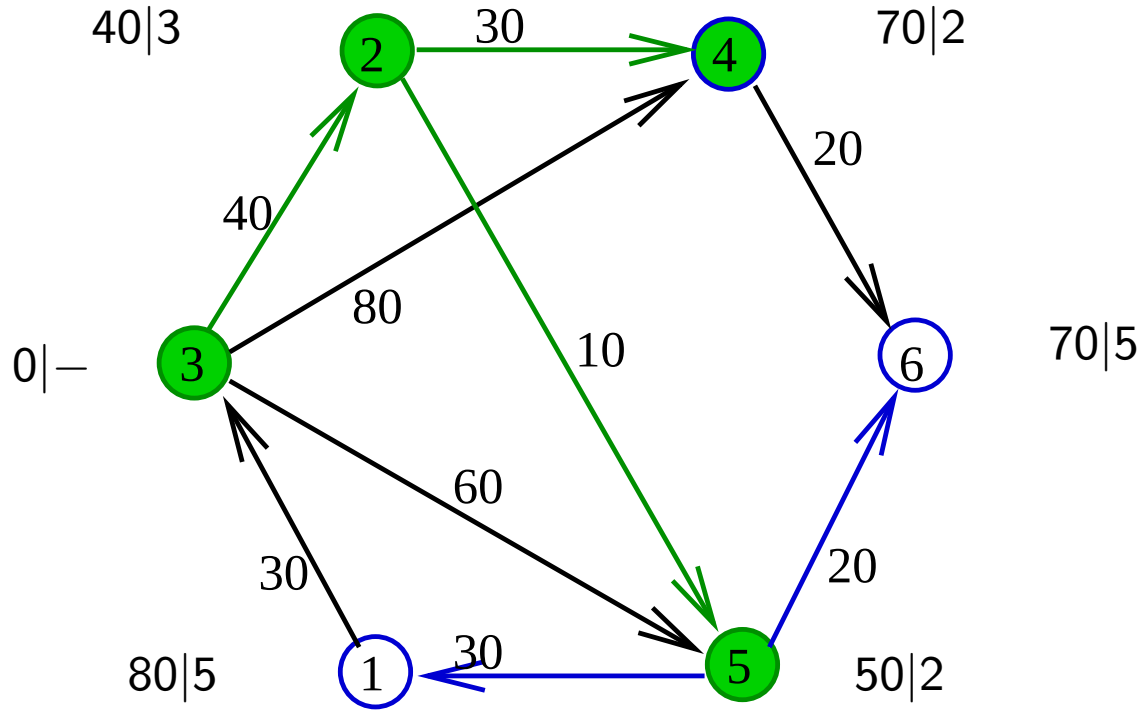
r	$x(r)$	$t(r)$	1	2	3	4	5	6
			$t(v)\|x(v)$					
-	-	-	∞ |	∞ |	**0**|	∞ |	∞ |	∞ |
3	-	0	∞ |	**40**|**3**		80|3	60|3	∞ |
2	3	40	∞ |			70|2	**50**|**2**	∞ |
5	2	50	80|5			**70**|**2**		70|5
4	2	70	80|5					**70**|**5**
6	5	70	**80**|**5**					
1	5	80						

Dijkstra's algorithm – Example



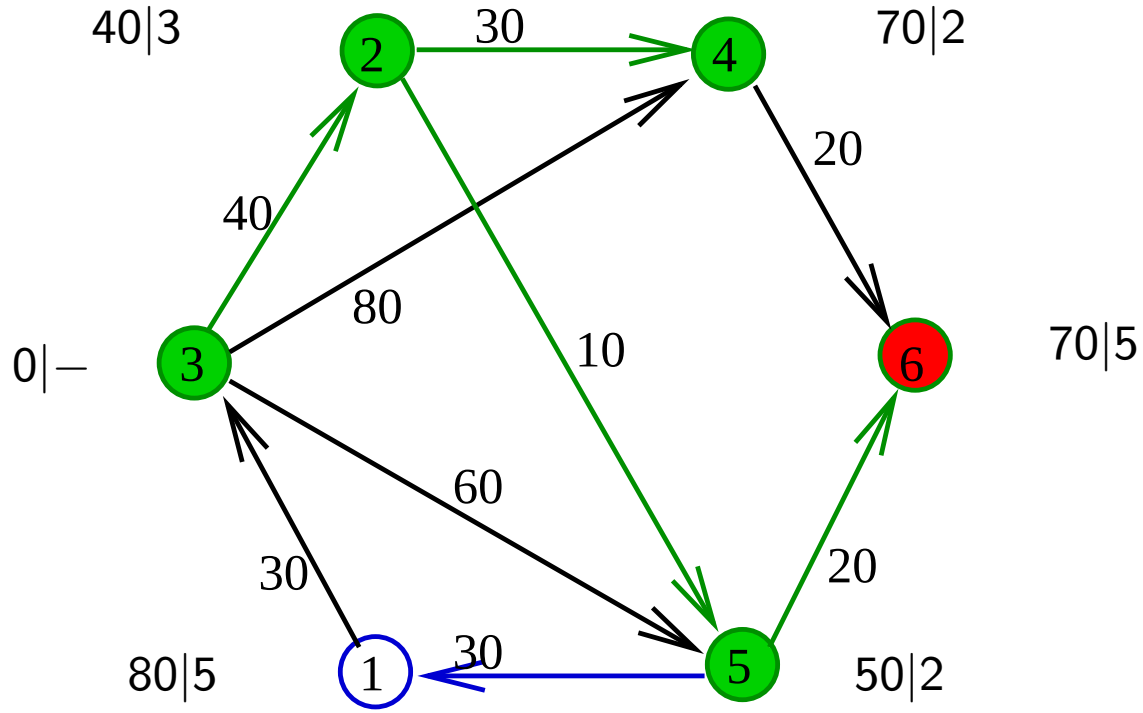
r	$x(r)$	$t(r)$	1	2	3	4	5	6
			$t(v)\|x(v)$					
-	-	-	∞ |	∞ |	**0**|	∞ |	∞ |	∞ |
3	-	0	∞ |	**40**|**3**		80|3	60|3	∞ |
2	3	40	∞ |			70|2	**50**|**2**	∞ |
5	2	50	80|5			**70**|**2**		70|5
4	2	70	80|5					**70**|**5**
6	5	70	**80**|**5**					
1	5	80						

Dijkstra's algorithm – Example



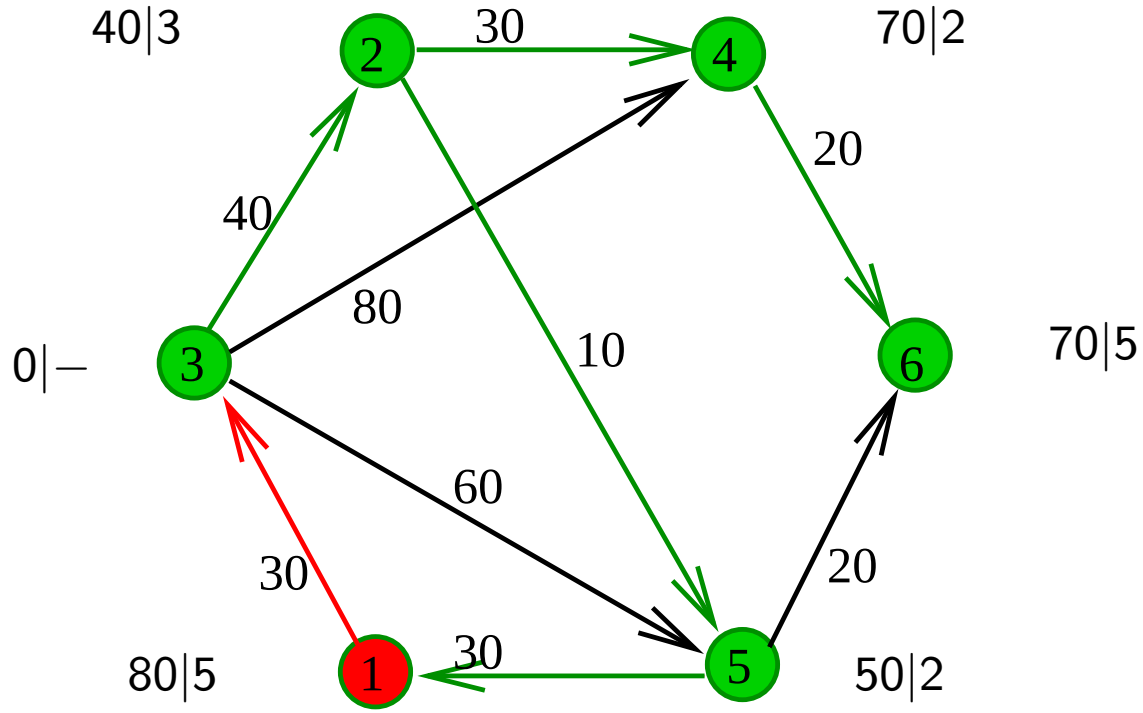
r	$x(r)$	$t(r)$	1	2	3	4	5	6
			$t(v)\|x(v)$					
-	-	-	∞ |	∞ |	**0**|	∞ |	∞ |	∞ |
3	-	0	∞ |	**40**|**3**		80|3	60|3	∞ |
2	3	40	∞ |			70|2	**50**|**2**	∞ |
5	2	50	80|5			**70**|**2**		70|5
4	2	70	80|5					**70**|**5**
6	5	70	**80**|**5**					
1	5	80						

Dijkstra's algorithm – Example



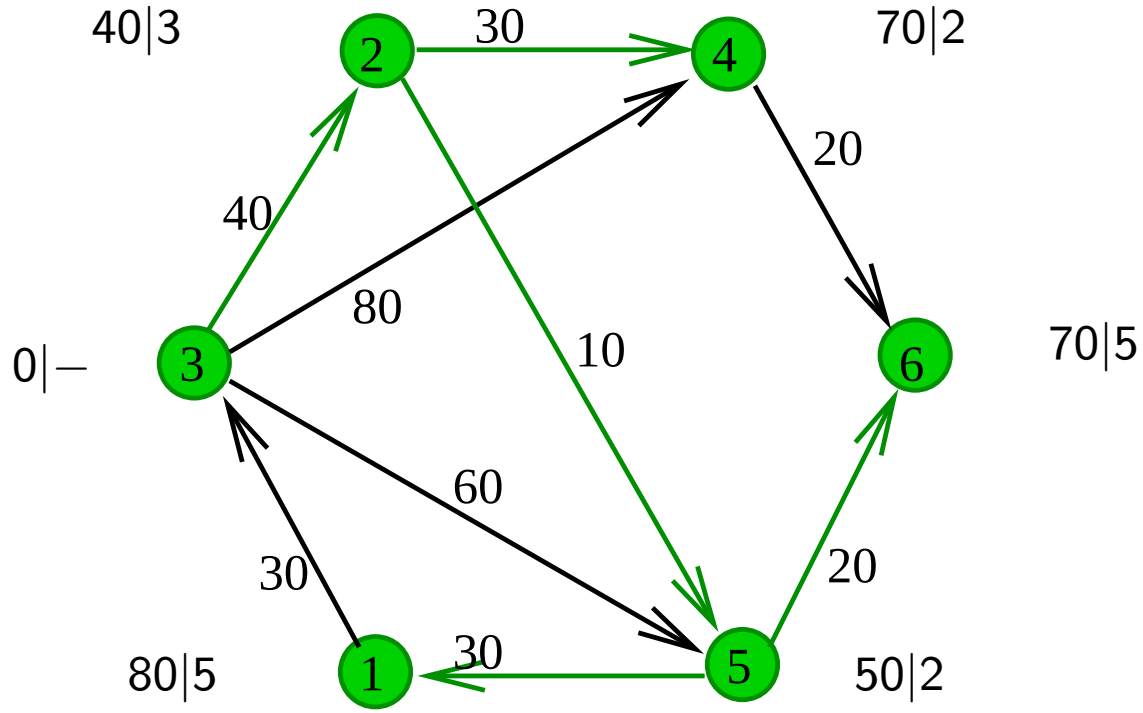
r	$x(r)$	$t(r)$	1	2	3	4	5	6
			$t(v)\|x(v)$					
-	-	-	∞ |	∞ |	**0**|	∞ |	∞ |	∞ |
3	-	0	∞ |	**40|3**		80|3	60|3	∞ |
2	3	40	∞ |			70|2	**50|2**	∞ |
5	2	50	80|5			**70|2**		70|5
4	2	70	80|5					**70|5**
6	5	70	**80|5**					
1	5	80						

Dijkstra's algorithm – Example

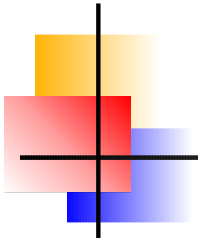


r	$x(r)$	$t(r)$	1	2	3	4	5	6
			$t(v)\|x(v)$					
-	-	-	∞|	∞|	**0**|	∞|	∞|	∞|
3	-	0	∞|	**40**|**3**		80|3	60|3	∞|
2	3	40	∞|			70|2	**50**|**2**	∞|
5	2	50	80|5			**70**|**2**		70|5
4	2	70	80|5					**70**|**5**
6	5	70	**80**|**5**					
1	5	80						

Dijkstra's algorithm – Example



r	$x(r)$	$t(r)$	1	2	3	4	5	6
			$t(v)\|x(v)$					
-	-	-	∞|	∞|	**0**|	∞|	∞|	∞|
3	-	0	∞|	**40|3**		80|3	60|3	∞|
2	3	40	∞|			70|2	**50|2**	∞|
5	2	50	80|5			**70|2**		70|5
4	2	70	80|5					**70|5**
6	5	70	**80|5**					
1	5	80						



Distance matrix

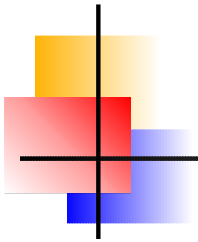
Definition

A real function d defined on cartesian product $V \times V$ i called **a metrics on the set** *V, if it holds:*

1. $d(u, v) \geq 0$ *for every* $u, v \in V$ *and* $d(u, v) = 0$ *if and only if* $u = v$.
2. $d(u, v) = d(v, u)$ *for every* $u, v \in V$.
3. $d(u, w) \leq d(u, v) + d(v, w)$ *for every* $u, v, w \in V$.

Definition

Let $G = (V, H, c)$ be a connected edge weighted graph resp. let $\overrightarrow{G} = (V, H, c)$ be a strongly connected arc weighted digraph, $c(h) > 0$.
The distance of vertices *$u, v \in V$ $d(u, v)$ is the length of the shortest u–v path.*



Distance matrix

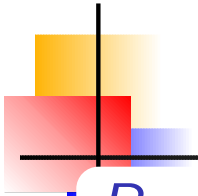
Definition

A real function d defined on cartesian product $V \times V$ i called **a metrics on the set** *V, if it holds:*

1. $d(u, v) \geq 0$ *for every* $u, v \in V$ *and* $d(u, v) = 0$ *if and only if* $u = v$.
2. $d(u, v) = d(v, u)$ *for every* $u, v \in V$.
3. $d(u, w) \leq d(u, v) + d(v, w)$ *for every* $u, v, w \in V$.

Definition

Let $G = (V, H, c)$ be a connected edge weighted graph resp. let $\overrightarrow{G} = (V, H, c)$ be a strongly connected arc weighted digraph, $c(h) > 0$.
The distance of vertices *$u, v \in V$ $d(u, v)$ is the length of the shortest u–v path.*



Distance matrix

Remark

Since we have admitted trivial u–u path (containing single vertex u having zero length), the distance $d(u,u) = 0$ for every $u \in V$.

Theorem

If $c(h) > 0$ in a connected graph $G = (V, H, c)$ then distance function $d : V \times V \to \mathbb{R}$ is a metrics on the vertex set V.

Remark

Last theorem does not hold in digraphs – in digraphs need not hold $d(u,v) = d(v,u)$).

Remark

If graph G is disconnected, resp. digraf $\overrightarrow{G}$ is not strongly connected then it is possebl to define $d(u,v) = \infty$ for $u \in V$, $v \in V$ such that v is not reachable from u.

Distance matrix

Remark

Since we have admitted trivial u–u path (containing single vertex u having zero length), the distance $d(u,u) = 0$ for every $u \in V$.

Theorem

If $c(h) > 0$ in a connected graph $G = (V, H, c)$ then distance function $d : V \times V \to \mathbb{R}$ is a metrics on the vertex set V.

Remark

Last theorem does not hold in digraphs – in digraphs need not hold $d(u,v) = d(v,u)$).

Remark

If graph G is disconnected, resp. digraf $\overrightarrow{G}$ is not strongly connected then it is possebl to define $d(u,v) = \infty$ for $u \in V$, $v \in V$ such that v is not reachable from u.

Distance matrix

Remark

Since we have admitted trivial u–u path (containing single vertex u having zero length), the distance $d(u,u) = 0$ for every $u \in V$.

Theorem

If $c(h) > 0$ in a connected graph $G = (V, H, c)$ then distance function $d : V \times V \to \mathbb{R}$ is a metrics on the vertex set V.

Remark

Last theorem does not hold in digraphs – in digraphs need not hold $d(u,v) = d(v,u)$).

Remark

If graph G is disconnected, resp. digraf $\overrightarrow{G}$ is not strongly connected then it is possebl to define $d(u,v) = \infty$ for $u \in V$, $v \in V$ such that v is not reachable from u.

Distance matrix

Remark

Since we have admitted trivial u–u path (containing single vertex u having zero length), the distance $d(u, u) = 0$ for every $u \in V$.

Theorem

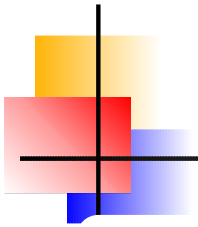
If $c(h) > 0$ in a connected graph $G = (V, H, c)$ then distance function $d : V \times V \to \mathbb{R}$ is a metrics on the vertex set V.

Remark

Last theorem does not hold in digraphs – in digraphs need not hold $d(u, v) = d(v, u)$).

Remark

If graph G is disconnected, resp. digraf $\overrightarrow{G}$ is not strongly connected then it is possebl to define $d(u, v) = \infty$ for $u \in V$, $v \in V$ such that v is not reachable from u.



Radius, center and eccentricity of a graph

Definition

Leth $G = (V, H, c)$ *be an edge weighted connected graph,* $c(h) > 0$. *We define:*

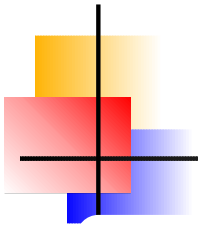
$$\begin{aligned} \textbf{eccentricity of vertex } v \in V \quad & e(v) = \max\{d(u,v) \mid u \in V\} \\ \textbf{radius of graph } G \quad & r(G) = \min\{e(v) \mid v \in V\} \\ \textbf{diameter of graph } G \quad & d(G) = \max\{e(v) \mid v \in V\} \end{aligned}$$

Every vertex of graph G *with minimum eccentricity* $e(v)$ *is called* **a central vertex** *of graph* G, *the set of all central vertices of graph is called* **center of graph** G.

Remark

It can be easily shown that:

$$d(G) = \max\{d(u,v) \mid u \in V, v \in V\}.$$



Radius, center and eccentricity of a graph

Definition

Leth $G = (V, H, c)$ be an edge weighted connected graph, $c(h) > 0$. We define:

$$\begin{aligned} \textbf{eccentricity of vertex } v \in V \quad & e(v) = \max\{d(u,v) \mid u \in V\} \\ \textbf{radius of graph } G \quad & r(G) = \min\{e(v) \mid v \in V\} \\ \textbf{diameter of graph } G \quad & d(G) = \max\{e(v) \mid v \in V\} \end{aligned}$$

Every vertex of graph G with minimum eccentricity $e(v)$ is called **a central vertex** *of graph G, the set of all central vertices of graph is called* **center of graph** *G.*

Remark

It can be easily shown that:

$$d(G) = \max\{d(u,v) \mid u \in V, v \in V\}.$$

Floyd's distance matrix algorithm

Algorithm

Floyd's algorithm *Distance matrix algorithm in an edge weighted graph or digraph* $G = (V, H, c)$, *where* $c(h) \geq 0$.

- **Step 1.** *Create a matrix* $\mathbf{C} = (c_{ij})$ *with entries defined as follows:*

$$c_{ii} = 0 \quad \text{for all} \quad i \in V$$

and for all $i,\ j$ *such that* $i \neq j$

$$c_{ij} = \begin{cases} c(i,j), & \text{if } \{i,j\} \in H, \text{ resp. } (i,j) \in H \\ \infty, & \text{if } \{i,j\} \notin H, \text{ resp. } (i,j) \notin H \end{cases}$$

Create a matrix $\mathbf{X} = (x_{ij})$ *with entries:*

$$x_{ii} = i \quad \text{for all} \quad i \in V$$

and for all $i,\ j$ *such that* $i \neq j$

$$x_{ij} = \begin{cases} i \quad , & \text{if } \{i,j\} \in H, \text{ resp. } (i,j) \in H \\ \infty, & \text{if } \{i,j\} \notin H, \text{ resp. } (i,j) \notin H \end{cases}$$

Floyd's distance matrix algorithm

Algorithm (- continued)

- **Step 2.** *For all* $k = 1, 2, \ldots, n = |V|$*:*
  For all $i \neq k$ *such that* $c_{ik} \neq \infty$*, and for all* $j \neq k$ *such that* $c_{kj} \neq \infty$ *do:*
  If $c_{ij} > c_{ik} + c_{kj}$ *the set:*

$$c_{ij} := c_{ik} + c_{kj}$$

$$x_{ij} := x_{kj}$$

♣

Matrix **C** is the distance matrix of graph, resp. digraph G after finishing Floyd's algorithm.

Matrix **X** contains for every pair i, j last but one vertex of the shortest i–j path after Floyd's algorithm stops.

If we need to find shortest i–j path we can make use of matrix **X** of pointers as follows:
Last but one vertex j_1 of i–j shortest path is $j_1 = x_{ij}$. Last but two wertex from behind is $j_2 = x_{ij_1}$, last but three is $j_3 = x_{ij_2}$ etc. until starting vertex i is reached.

Floyd's distance matrix algorithm

Algorithm (- continued)

- **Step 2.** *For all* $k = 1, 2, \ldots, n = |V|$*:*
  For all $i \neq k$ *such that* $c_{ik} \neq \infty$*, and for all* $j \neq k$ *such that* $c_{kj} \neq \infty$ *do:*
  If $c_{ij} > c_{ik} + c_{kj}$ *the set:*

$$c_{ij} := c_{ik} + c_{kj}$$

$$x_{ij} := x_{kj}$$

♣

Matrix **C** is the distance matrix of graph, resp. digraph G after finishing Floyd's algorithm.

Matrix **X** contains for every pair i, j last but one vertex of the shortest i–j path after Floyd's algorithm stops.

If we need to find shortest i–j path we can make use of matrix **X** of pointers as follows:
Last but one vertex j_1 of i–j shortest path is $j_1 = x_{ij}$. Last but two wertex from behind is $j_2 = x_{ij_1}$, last but three is $j_3 = x_{ij_2}$ etc. until starting vertex i is reached.

Floyd's distance matrix algorithm

Algorithm (- continued)

- **Step 2.** *For all* $k = 1, 2, \ldots, n = |V|$*:*
  For all $i \neq k$ *such that* $c_{ik} \neq \infty$*, and for all* $j \neq k$ *such that* $c_{kj} \neq \infty$ *do:*
  If $c_{ij} > c_{ik} + c_{kj}$ *the set:*

$$c_{ij} := c_{ik} + c_{kj}$$

$$x_{ij} := x_{kj}$$

♣

Matrix **C** is the distance matrix of graph, resp. digraph G after finishing Floyd's algorithm.

Matrix **X** contains for every pair i, j last but one vertex of the shortest i–j path after Floyd's algorithm stops.

If we need to find shortest i–j path we can make use of matrix **X** of pointers as follows:
Last but one vertex j_1 of i–j shortest path is $j_1 = x_{ij}$. Last but two wertex from behind is $j_2 = x_{ij_1}$, last but three is $j_3 = x_{ij_2}$ etc. until starting vertex i is reached.

Floyd's distance matrix algorithm

Algorithm (- continued)

- **Step 2.** *For all* $k = 1, 2, \ldots, n = |V|$*:*
  For all $i \neq k$ *such that* $c_{ik} \neq \infty$*, and for all* $j \neq k$ *such that* $c_{kj} \neq \infty$ *do:*
  If $c_{ij} > c_{ik} + c_{kj}$ *the set:*

$$c_{ij} := c_{ik} + c_{kj}$$

$$x_{ij} := x_{kj}$$

♣

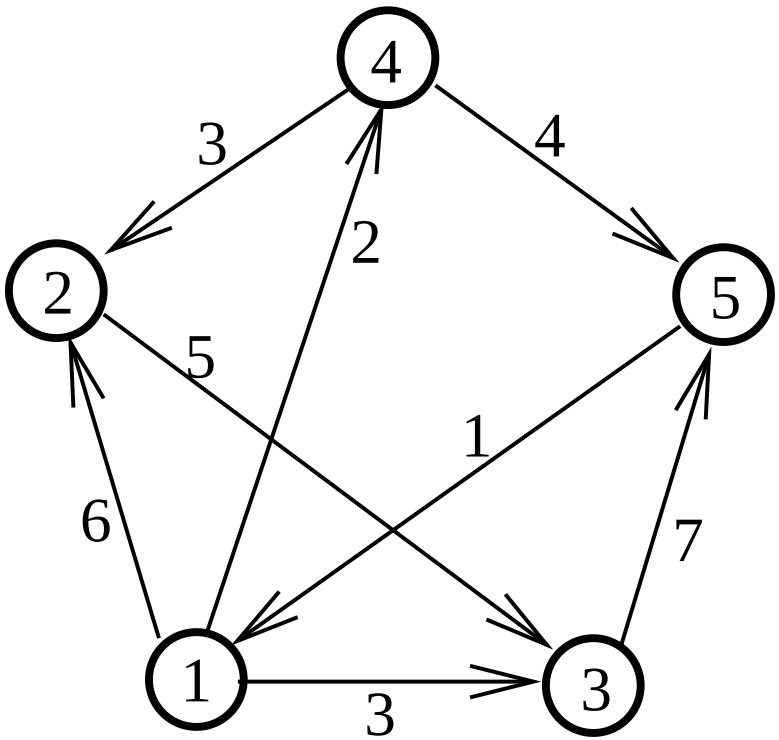
Matrix **C** is the distance matrix of graph, resp. digraph G after finishing Floyd's algorithm.

Matrix **X** contains for every pair i, j last but one vertex of the shortest i–j path after Floyd's algorithm stops.

If we need to find shortest i–j path we can make use of matrix **X** of pointers as follows:
Last but one vertex j_1 of i–j shortest path is $j_1 = x_{ij}$. Last but two wertex from behind is $j_2 = x_{ij_1}$, last but three is $j_3 = x_{ij_2}$ etc. until starting vertex i is reached.

Floyd's distance matrix algorithm – Example

Digraf $\overrightarrow{G} = (V, H, c)$ represented by diagram in the following picture. Our task is to compute corresponding distance matrix using Floyd's algorithm.



Obr.: Digraph $\overrightarrow{G} = (V, H, c)$.

Matrix **C**

	1	2	3	4	5
1	0	6	3	2	∞
2	∞	0	5	∞	∞
3	∞	∞	0	∞	7
4	∞	3	∞	0	4
5	1	∞	∞	∞	0

Matrix **X**

	1	2	3	4	5
1	1	1	1	1	∞
2	∞	2	2	∞	∞
3	∞	∞	3	∞	3
4	∞	4	∞	4	4
5	5	∞	∞	∞	5

Floyd's distance matrix algorithm – Example

Matrix **C**

	1	2	3	4	5
1	**0**	**6**	**3**	**2**	∞
2	∞	0	5	∞	∞
3	∞	∞	0	∞	7
4	∞	3	∞	0	4
5	**1**	∞	∞	∞	0

Matrix **X**

	1	2	3	4	5
1	**1**	**1**	**1**	**1**	∞
2	∞	2	2	∞	∞
3	∞	∞	3	∞	3
4	∞	4	∞	4	4
5	**5**	∞	∞	∞	5

Matrix **C**
after step 2 with $k = 1$

	1	2	3	4	5
1	0	**6**	3	2	∞
2	∞	**0**	**5**	∞	∞
3	∞	∞	0	∞	7
4	∞	**3**	∞	0	4
5	1	**7**	4	3	0

Matrix **X**
after step 2 with $k = 1$

	1	2	3	4	5
1	1	**1**	1	1	∞
2	∞	**2**	**2**	∞	∞
3	∞	∞	3	∞	3
4	∞	**4**	∞	4	4
5	5	**1**	1	1	5

Matrix **C**
after step 2 with $k = 2$

	1	2	3	4	5
1	0	6	**3**	2	∞
2	∞	0	**5**	∞	∞
3	∞	∞	**0**	∞	**7**
4	∞	3	**8**	0	4
5	1	7	**4**	3	0

Matrix **X**
after step 2 with $k = 2$

	1	2	3	4	5
1	1	1	**1**	1	∞
2	∞	2	**2**	∞	∞
3	∞	∞	**3**	∞	**3**
4	∞	4	**2**	4	4
5	5	1	**1**	1	5

Floyd's distance matrix algorithm – Example

Matrix **C**

	1	2	3	4	5
1	**0**	**6**	**3**	**2**	**∞**
2	**∞**	0	5	∞	∞
3	**∞**	∞	0	∞	7
4	**∞**	3	∞	0	4
5	**1**	∞	∞	∞	0

Matrix **C**
after step 2 with $k = 1$

	1	2	3	4	5
1	0	**6**	3	2	∞
2	**∞**	**0**	**5**	**∞**	**∞**
3	∞	**∞**	0	∞	7
4	∞	**3**	∞	0	4
5	1	**7**	4	3	0

Matrix **C**
after step 2 with $k = 2$

	1	2	3	4	5
1	0	6	**3**	2	∞
2	∞	0	**5**	∞	∞
3	∞	∞	**0**	∞	**7**
4	∞	3	**8**	0	4
5	1	7	**4**	3	0

Matrix **X**

	1	2	3	4	5
1	**1**	**1**	**1**	**1**	**∞**
2	**∞**	2	2	∞	∞
3	**∞**	∞	3	∞	3
4	**∞**	4	∞	4	4
5	**5**	∞	∞	∞	5

Matrix **X**
after step 2 with $k = 1$

	1	2	3	4	5
1	1	**1**	1	1	∞
2	**∞**	**2**	**2**	**∞**	**∞**
3	∞	**∞**	3	∞	3
4	∞	**4**	∞	4	4
5	5	**1**	1	1	5

Matrix **X**
after step 2 with $k = 2$

	1	2	3	4	5
1	1	1	**1**	1	∞
2	∞	2	**2**	∞	∞
3	∞	∞	**3**	∞	**3**
4	∞	4	**2**	4	4
5	5	1	**1**	1	5

Floyd's distance matrix algorithm – Example

Matrix **C**

	1	2	3	4	5
1	**0**	**6**	**3**	**2**	**∞**
2	**∞**	0	5	∞	∞
3	**∞**	∞	0	∞	7
4	**∞**	3	∞	0	4
5	**1**	∞	∞	∞	0

Matrix **C**
after step 2 with $k = 1$

	1	2	3	4	5
1	0	**6**	3	2	∞
2	**∞**	**0**	**5**	**∞**	**∞**
3	∞	∞	0	∞	7
4	∞	**3**	∞	0	4
5	1	**7**	4	3	0

Matrix **C**
after step 2 with $k = 2$

	1	2	**3**	4	5
1	0	6	**3**	2	∞
2	∞	0	**5**	∞	∞
3	**∞**	**∞**	**0**	**∞**	**7**
4	∞	3	**8**	0	4
5	1	7	**4**	3	0

Matrix **X**

	1	2	3	4	5
1	**1**	**1**	**1**	**1**	**∞**
2	**∞**	2	2	∞	∞
3	**∞**	∞	3	∞	3
4	**∞**	4	∞	4	4
5	**5**	∞	∞	∞	5

Matrix **X**
after step 2 with $k = 1$

	1	2	3	4	5
1	1	**1**	1	1	∞
2	**∞**	**2**	**2**	**∞**	**∞**
3	∞	∞	3	∞	3
4	∞	**4**	∞	4	4
5	5	**1**	1	1	5

Matrix **X**
after step 2 with $k = 2$

	1	2	**3**	4	5
1	1	1	**1**	1	∞
2	∞	2	**2**	∞	∞
3	**∞**	**∞**	**3**	**∞**	**3**
4	∞	4	**2**	4	4
5	5	1	**1**	1	5

Floyd's distance matrix algorithm – Example

Matrix **C**
after step 2 with $k = 3$

	1	2	3	4	5
1	0	6	3	2	10
2	∞	0	5	∞	12
3	∞	∞	0	∞	7
4	∞	3	8	0	4
5	1	7	4	3	0

Matrix **X**
after step 2 with $k = 3$

	1	2	3	4	5
1	1	1	1	1	3
2	∞	2	2	∞	3
3	∞	∞	3	∞	3
4	∞	4	2	4	4
5	5	1	1	1	5

Matrix **C**
after step 2 with $k = 4$

	1	2	3	4	5
1	0	5	3	2	6
2	∞	0	5	∞	12
3	∞	∞	0	∞	7
4	∞	3	8	0	4
5	1	6	4	3	0

Matrix **X**
after step 2 with $k = 4$

	1	2	3	4	5
1	1	4	1	1	4
2	∞	2	2	∞	3
3	∞	∞	3	∞	3
4	∞	4	4	4	4
5	5	4	1	1	5

Matrix **C**
after step 2 with $k = 5$

	1	2	3	4	5
1	0	5	3	2	6
2	13	0	5	15	12
3	6	13	0	10	7
4	5	3	8	0	4
5	1	6	4	3	0

Matrix **X**
after step 2 with $k = 5$

	1	2	3	4	5
1	1	4	1	1	4
2	5	2	2	1	3
3	5	4	3	1	3
4	5	4	2	4	4
5	5	4	1	1	5

Floyd's distance matrix algorithm – Example

Matrix **C**
after step 2 with $k = 3$

	1	2	3	4	5
1	0	6	3	2	10
2	∞	0	5	∞	12
3	∞	∞	0	∞	7
4	∞	3	8	0	4
5	1	7	4	3	0

Matrix **X**
after step 2 with $k = 3$

	1	2	3	4	5
1	1	1	1	1	3
2	∞	2	2	∞	3
3	∞	∞	3	∞	3
4	∞	4	2	4	4
5	5	1	1	1	5

Matrix **C**
after step 2 with $k = 4$

	1	2	3	4	5
1	0	5	3	2	6
2	∞	0	5	∞	12
3	∞	∞	0	∞	7
4	∞	3	8	0	4
5	1	6	4	3	0

Matrix **X**
after step 2 with $k = 4$

	1	2	3	4	5
1	1	4	1	1	4
2	∞	2	2	∞	3
3	∞	∞	3	∞	3
4	∞	4	4	4	4
5	5	4	1	1	5

Matrix **C**
after step 2 with $k = 5$

	1	2	3	4	5
1	0	5	3	2	6
2	13	0	5	15	12
3	6	13	0	10	7
4	5	3	8	0	4
5	1	6	4	3	0

Matrix **X**
after step 2 with $k = 5$

	1	2	3	4	5
1	1	4	1	1	4
2	5	2	2	1	3
3	5	4	3	1	3
4	5	4	2	4	4
5	5	4	1	1	5

Floyd's distance matrix algorithm – Example

Matrix **C**
after step 2 with $k = 3$

	1	2	3	4	5
1	0	6	3	**2**	10
2	∞	0	5	∞	12
3	∞	∞	0	∞	7
4	∞	**3**	**8**	**0**	**4**
5	1	7	4	**3**	0

Matrix **X**
after step 2 with $k = 3$

	1	2	3	4	5
1	1	1	1	**1**	3
2	∞	2	2	∞	3
3	∞	∞	3	∞	3
4	∞	**4**	**2**	**4**	**4**
5	5	1	1	**1**	5

Matrix **C**
after step 2 with $k = 4$

	1	2	3	4	**5**
1	0	5	3	2	**6**
2	∞	0	5	∞	**12**
3	∞	∞	0	∞	**7**
4	∞	3	8	0	**4**
5	**1**	**6**	**4**	**3**	**0**

Matrix **X**
after step 2 with $k = 4$

	1	2	3	4	**5**
1	1	4	1	1	**4**
2	∞	2	2	∞	**3**
3	∞	∞	3	∞	**3**
4	∞	4	4	4	**4**
5	**5**	**4**	**1**	**1**	**5**

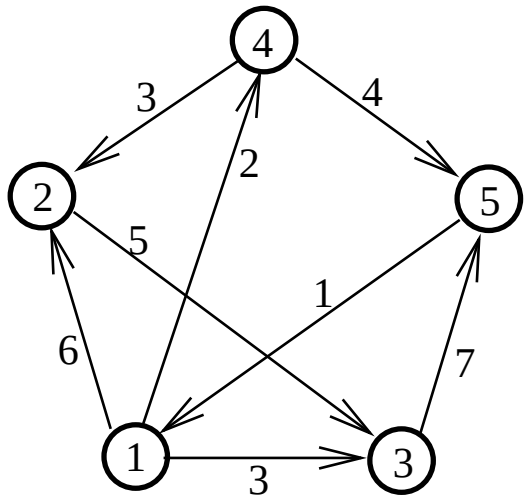
Matrix **C**
after step 2 with $k = 5$

	1	2	3	4	5
1	0	5	3	2	6
2	13	0	5	15	12
3	6	13	0	10	7
4	5	3	8	0	4
5	1	6	4	3	0

Matrix **X**
after step 2 with $k = 5$

	1	2	3	4	5
1	1	4	1	1	4
2	5	2	2	1	3
3	5	4	3	1	3
4	5	4	2	4	4
5	5	4	1	1	5

Floyd's distance matrix algorithm – Example – Results



Matrix **C**
after step 2 with $k = 5$

	1	2	3	4	5
1	0	5	3	2	6
2	13	0	5	15	12
3	6	13	0	**10**	7
4	5	3	8	0	4
5	1	6	4	3	0

i-th row of resultin matrix **X** contain pointers for construction all shortest *i*-*j* paths.

Matrix **X**
after step 2 with $k = 5$

	1	2	3	4	5
1	1	4	1	1	4
2	5	2	2	1	3
3	**5**	4	3	**1**	**3**
4	5	4	2	4	4
5	5	4	1	1	5

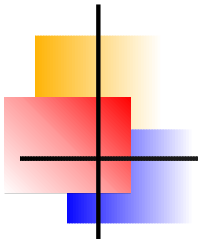
Le us search for the shortest 3-4 paths.

$x_{3,4} = 1$ – last but one vertex of searched path is vertex 1

$x_{3,1} = 5$ – last but two vertex is vertex 5

$x_{3,5} = 3$ – vertex 3 is the starting vertex of searched path

Shortest 3-4 path is $(\mathbf{3}, (3,5), \mathbf{5}, (5,1), \mathbf{1}, (1,4), \mathbf{4})$ and its length is $c_{3,4} = \mathbf{10}$.



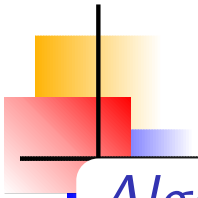
Floyd's distance matrix algorithm – Example – Programm

Example

Floyd's algorithm:

```
for(k=1; k <= n; k++)
   for(i=1; i <= n; i++)
      for(j=1; j <= n; j++)
         if(c[i,j] > c[i,k] + c[k,j])
         {  c[i,j] = c[i,k] + c[k,j];
            x[i,j] = x[k,j]; }
```

The number of elementary operations following statment **if** *in inner cycle* **for***(j=1; ...) (printed in blue) can be limited from above by a fixed number K. This statement will be executed n^3 times. The number of elementry steps of Floyd's algorithm can be limited from above by Kn^3. Hence Floyd's algorithm is an $O(n^3)$ algorithm.*



Label-set a Label-correct implementation

Algorithm

Label-set a Label-correct implemention of point-to all shortest path algorithm *from a fixed starting vertex $u \in V$ into all vertices $v \in V$ in an arc weighted digraph $\overrightarrow{G} = (V, H, c)$ with $c(h) \geq 0$. Suppose $0 \notin V$.*

- **Step 1: Initializtion.**
  Set $t(u) := 0$, $t(i) := \infty$ for $i \in V$, $i \neq u$ and $x(i) := 0$ for all $i \in V$.

  Set $\mathcal{E} := \{u\}$.

- **Step 2:** *Extract a vertex $r \in \mathcal{E}$, set $\mathcal{E} := \mathcal{E} - \{r\}$.*
  For all arcs (r, j) such that $(r, j) \in H^+(r)$ do:
  If $t(j) > t(r) + c(r, j)$ then

  $$t(j) := t(r) + c(r, j),\ x(j) := r,\ \mathcal{E} := \mathcal{E} \cup \{j\}.$$

- **Step 3:** *If $\mathcal{E} \neq \emptyset$ GOTO Step 2.*

  If $\mathcal{E} = \emptyset$ then $t(i)$ is equal to the length of the shortest u–i path for every vertex i reachable from u.

  Corresponding shortest u–i path can be created making use of pointers $x(\,)$ in the same way as in prior shortest path algorithms.

Label-set a Label-correct implementation

Algorithm

Label-set a Label-correct implemention of point-to all shortest path algorithm *from a fixed starting vertex $u \in V$ into all vertices $v \in V$ in an arc weighted digraph $\overrightarrow{G} = (V, H, c)$ with $c(h) \geq 0$. Suppose $0 \notin V$.*

- **Step 1: Initializtion.**
  Set $t(u) := 0$, $t(i) := \infty$ for $i \in V$, $i \neq u$ and $x(i) := 0$ for all $i \in V$.

  Set $\mathcal{E} := \{u\}$.

- **Step 2:** *Extract a vertex $r \in \mathcal{E}$, set $\mathcal{E} := \mathcal{E} - \{r\}$.*
  For all arcs (r, j) such that $(r, j) \in H^+(r)$ do:
  If $t(j) > t(r) + c(r, j)$ then

  $$t(j) := t(r) + c(r, j),\ x(j) := r,\ \mathcal{E} := \mathcal{E} \cup \{j\}.$$

- **Step 3:** *If $\mathcal{E} \neq \emptyset$ GOTO Step 2.*

  If $\mathcal{E} = \emptyset$ then $t(i)$ is equal to the length of the shortest u–i path for every vertex i reachable from u.

  Corresponding shortest u–i path can be created making use of pointers $x(\,)$ in the same way as in prior shortest path algorithms.

Algorithm

Label-set a Label-correct implemention of point-to all shortest path algorithm *from a fixed starting vertex $u \in V$ into all vertices $v \in V$ in an arc weighted digraph $\overrightarrow{G} = (V, H, c)$ with $c(h) \geq 0$. Suppose $0 \notin V$.*

- **Step 1: Initializtion.**
  Set $t(u) := 0$, $t(i) := \infty$ for $i \in V$, $i \neq u$ and $x(i) := 0$ for all $i \in V$.

  Set $\mathcal{E} := \{u\}$.

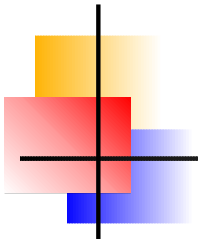
- **Step 2:** *Extract a vertex $r \in \mathcal{E}$, set $\mathcal{E} := \mathcal{E} - \{r\}$.*
  For all arcs (r, j) such that $(r, j) \in H^+(r)$ do:
  If $t(j) > t(r) + c(r, j)$ then

  $$t(j) := t(r) + c(r, j),\ x(j) := r,\ \mathcal{E} := \mathcal{E} \cup \{j\}.$$

- **Step 3:** *If $\mathcal{E} \neq \emptyset$ GOTO Step 2.*

  If $\mathcal{E} = \emptyset$ then $t(i)$ is equal to the length of the shortest u–i path for every vertex i reachable from u.

  Corresponding shortest u–i path can be created making use of pointers $x(\ )$ in the same way as in prior shortest path algorithms.



Label-set a Label-correct implementation

If we extract in the second step of last algorithm pivot $r \in \mathcal{E}$ arbitrarily without any rule we get implementation of fundamental algorithm called **label correct algorithm**.

If we extract in the second step of last algorithm pivot $r \in \mathcal{E}$ with the least value of label $t(\ )$ we get implementation of Dijkstra's algorithm called **label set algorithm**.

It is convenient to organize the set $\mathcal{E}$ as priority queue for label set algorithm.

Label-set a Label-correct implementation - Example

h	$(1,3)$	$(1,7)$	$(2,4)$	$(3,2)$	$(3,5)$	$(4,3)$	$(4,6)$	$(5,1)$	$(5,2)$	$(5,6)$	$(6,1)$	$(7,3)$
$c(h)$	20	120	30	10	60	80	10	30	90	150	20	40

	r	$t(r)$	1	2	3	4	5	6	7	
			$t(v)\|x(v)$							the set $\mathcal{E}$
	-		$\infty\|0$	$\infty\|0$	$0\|0$	$\infty\|0$	$\infty\|0$	$\infty\|0$	$\infty\|0$	3
1.	3	0		$10\|3$			$60\|3$			2 5
2.	2	10				$40\|2$				5 4
3.	5	60	$90\|5$					$210\|5$		4 1 6
4.	4	40						$50\|4$		1 6
5.	1	90							$210\|1$	6 7
6.	6	50	$70\|6$							7 1
7.	7	210								1
8.	1	70							$190\|1$	7
9.	7	190								

Label-set a Label-correct implementation - Example

h	$(1,3)$	$(1,7)$	$(2,4)$	$(3,2)$	$(3,5)$	$(4,3)$	$(4,6)$	$(5,1)$	$(5,2)$	$(5,6)$	$(6,1)$	$(7,3)$
$c(h)$	20	120	30	10	60	80	10	30	90	150	20	40

	r	$t(r)$	1	2	3	4	5	6	7	
							$t(v)\|x(v)$			the set $\mathcal{E}$
	-		$\infty\|0$	$\infty\|0$	$0\|0$	$\infty\|0$	$\infty\|0$	$\infty\|0$	$\infty\|0$	3
1.	3	0		$10\|3$			$60\|3$			2 5
2.	2	10				$40\|2$				5 4
3.	5	60	$90\|5$					$210\|5$		4 1 6
4.	4	40						$50\|4$		1 6
5.	1	90							$210\|1$	6 7
6.	6	50	$70\|6$							7 1
7.	7	210								1
8.	1	70							$190\|1$	7
9.	7	190								

Label-set a Label-correct implementation - Example

h	$(1,3)$	$(1,7)$	$(2,4)$	$(3,2)$	$(3,5)$	$(4,3)$	$(4,6)$	$(5,1)$	$(5,2)$	$(5,6)$	$(6,1)$	$(7,3)$
$c(h)$	20	120	30	10	60	80	10	30	90	150	20	40

	r	$t(r)$	1	2	3	4	5	6	7	
							$t(v)\|x(v)$			the set $\mathcal{E}$
	-		$\infty\|0$	$\infty\|0$	$0\|0$	$\infty\|0$	$\infty\|0$	$\infty\|0$	$\infty\|0$	3
1.	3	0		$10\|3$			$60\|3$			2 5
2.	2	10				$40\|2$				5 4
3.	5	60	$90\|5$					$210\|5$		4 1 6
4.	4	40						$50\|4$		1 6
5.	1	90							$210\|1$	6 7
6.	6	50	$70\|6$							7 1
7.	7	210								1
8.	1	70							$190\|1$	7
9.	7	190								

Label-set a Label-correct implementation - Example

h	$(1,3)$	$(1,7)$	$(2,4)$	$(3,2)$	$(3,5)$	$(4,3)$	$(4,6)$	$(5,1)$	$(5,2)$	$(5,6)$	$(6,1)$	$(7,3)$
$c(h)$	20	120	30	10	60	80	10	30	90	150	20	40

	r	$t(r)$	1	2	3	4	5	6	7	
							$t(v)\|x(v)$			the set $\mathcal{E}$
	-		$\infty\|0$	$\infty\|0$	$0\|0$	$\infty\|0$	$\infty\|0$	$\infty\|0$	$\infty\|0$	3
1.	3	0		$10\|3$			$60\|3$			2 5
2.	2	10				$40\|2$				5 4
3.	5	60	$90\|5$					$210\|5$		4 1 6
4.	4	40						$50\|4$		1 6
5.	1	90							$210\|1$	6 7
6.	6	50	$70\|6$							7 1
7.	7	210								1
8.	1	70							$190\|1$	7
9.	7	190								

Label-set a Label-correct implementation - Example

h	$(1,3)$	$(1,7)$	$(2,4)$	$(3,2)$	$(3,5)$	$(4,3)$	$(4,6)$	$(5,1)$	$(5,2)$	$(5,6)$	$(6,1)$	$(7,3)$
$c(h)$	20	120	30	10	60	80	10	30	90	150	20	40

	r	$t(r)$	1	2	3	4	5	6	7	
							$t(v)\|x(v)$			the set $\mathcal{E}$
	-		$\infty\|0$	$\infty\|0$	$0\|0$	$\infty\|0$	$\infty\|0$	$\infty\|0$	$\infty\|0$	3
1.	3	0		$10\|3$			$60\|3$			2 5
2.	2	10				$40\|2$				5 4
3.	5	60	$90\|5$					$210\|5$		4 1 6
4.	4	40						$50\|4$		1 6
5.	1	90							$210\|1$	6 7
6.	6	50	$70\|6$							7 1
7.	7	210								1
8.	1	70							$190\|1$	7
9.	7	190								

Label-set a Label-correct implementation - Example

h	$(1,3)$	$(1,7)$	$(2,4)$	$(3,2)$	$(3,5)$	$(4,3)$	$(4,6)$	$(5,1)$	$(5,2)$	$(5,6)$	$(6,1)$	$(7,3)$
$c(h)$	20	120	30	10	60	80	10	30	90	150	20	40

	r	$t(r)$	1	2	3	4	5	6	7	
			$t(v)\|x(v)$							the set $\mathcal{E}$
	-		$\infty\|0$	$\infty\|0$	$0\|0$	$\infty\|0$	$\infty\|0$	$\infty\|0$	$\infty\|0$	3
1.	3	0		$10\|3$			$60\|3$			2 5
2.	2	10				$40\|2$				5 4
3.	5	60	$90\|5$					$210\|5$		4 1 6
4.	4	40						$50\|4$		1 6
5.	1	90							$210\|1$	6 7
6.	6	50	$70\|6$							7 1
7.	7	210								1
8.	1	70							$190\|1$	7
9.	7	190								

Label-set a Label-correct implementation - Example

h	$(1,3)$	$(1,7)$	$(2,4)$	$(3,2)$	$(3,5)$	$(4,3)$	$(4,6)$	$(5,1)$	$(5,2)$	$(5,6)$	$(6,1)$	$(7,3)$
$c(h)$	20	120	30	10	60	80	10	30	90	150	20	40

	r	$t(r)$	1	2	3	4	5	6	7	
			$t(v)\|x(v)$							the set $\mathcal{E}$
	-		$\infty\|0$	$\infty\|0$	$0\|0$	$\infty\|0$	$\infty\|0$	$\infty\|0$	$\infty\|0$	3
1.	3	0		$10\|3$			$60\|3$			2 5
2.	2	10				$40\|2$				5 4
3.	5	60	$90\|5$					$210\|5$		4 1 6
4.	4	40						$50\|4$		1 6
5.	1	90							$210\|1$	6 7
6.	6	50	$70\|6$							7 1
7.	7	210								1
8.	1	70							$190\|1$	7
9.	7	190								

Label-set a Label-correct implementation - Example

h	$(1,3)$	$(1,7)$	$(2,4)$	$(3,2)$	$(3,5)$	$(4,3)$	$(4,6)$	$(5,1)$	$(5,2)$	$(5,6)$	$(6,1)$	$(7,3)$
$c(h)$	20	120	30	10	60	80	10	30	90	150	20	40

	r	$t(r)$	1	2	3	4	5	6	7	
			$t(v)\|x(v)$							the set $\mathcal{E}$
	-		$\infty\|0$	$\infty\|0$	$0\|0$	$\infty\|0$	$\infty\|0$	$\infty\|0$	$\infty\|0$	3
1.	3	0		$10\|3$			$60\|3$			2 5
2.	2	10				$40\|2$				5 4
3.	5	60	$90\|5$					$210\|5$		4 1 6
4.	4	40						$50\|4$		1 6
5.	1	90							$210\|1$	6 7
6.	6	50	$70\|6$							7 1
7.	7	210								1
8.	1	70							$190\|1$	7
9.	7	190								

Label-set a Label-correct implementation - Example

h	$(1,3)$	$(1,7)$	$(2,4)$	$(3,2)$	$(3,5)$	$(4,3)$	$(4,6)$	$(5,1)$	$(5,2)$	$(5,6)$	$(6,1)$	$(7,3)$
$c(h)$	20	120	30	10	60	80	10	30	90	150	20	40

	r	$t(r)$	1	2	3	4	5	6	7	
			$t(v)\|x(v)$							the set $\mathcal{E}$
	-		$\infty\|0$	$\infty\|0$	$0\|0$	$\infty\|0$	$\infty\|0$	$\infty\|0$	$\infty\|0$	3
1.	3	0		$10\|3$			$60\|3$			2 5
2.	2	10				$40\|2$				5 4
3.	5	60	$90\|5$					$210\|5$		4 1 6
4.	4	40						$50\|4$		1 6
5.	1	90							$210\|1$	6 7
6.	6	50	$70\|6$							7 1
7.	7	210								1
8.	1	70							$190\|1$	7
9.	7	190								

Label-set a Label-correct implementation - Example

h	$(1,3)$	$(1,7)$	$(2,4)$	$(3,2)$	$(3,5)$	$(4,3)$	$(4,6)$	$(5,1)$	$(5,2)$	$(5,6)$	$(6,1)$	$(7,3)$
$c(h)$	20	120	30	10	60	80	10	30	90	150	20	40

	r	$t(r)$	1	2	3	4	5	6	7	
			$t(v)\|x(v)$							the set $\mathcal{E}$
	-		$\infty\|0$	$\infty\|0$	$0\|0$	$\infty\|0$	$\infty\|0$	$\infty\|0$	$\infty\|0$	3
1.	3	0		$10\|3$			$60\|3$			2 5
2.	2	10				$40\|2$				5 4
3.	5	60	$90\|5$					$210\|5$		4 1 6
4.	4	40						$50\|4$		1 6
5.	1	90							$210\|1$	6 7
6.	6	50	$70\|6$							7 1
7.	7	210								1
8.	1	70							$190\|1$	7
9.	7	190								

Cycle with negative cost in a digraph

Algorithm

Algorithm for searching a cycle with negative cost *in an arc weighted digraph* $\overrightarrow{G} = (V, H, c)$ *with general arc cost.*

- **Step 1: Initialization.**
  Set $t(i) := 0$ *and* $x(i) := 0$ *for all* $i \in V$.
  Set $\mathcal{E} := V$.
- **Step 2:** *Extract* $r \in \mathcal{E}$, *set* $\mathcal{E} := \mathcal{E} - \{r\}$.
  For every arc (r, j), $(r, j) \in H^+(r)$ *do:*
  If $t(j) > t(r) + c(r, j)$ *then*
  set $t(j) := t(r) + c(r, j)$, $x(j) := r$, $\mathcal{E} := \mathcal{E} \cup \{j\}$ *and try whether vertex* j *is an element of sequence:*

$$x(j), x(x(j)), x(x(x(j))), \ldots$$

  If yes you have just found a cycle with negative cost containing vertex j. *STOP.*
  If no then GOTO Step 3.
- **Step 3:** *If* $\mathcal{E} \neq \emptyset$ *then GOTO Step 2.*
  If $\mathcal{E} = \emptyset$ *then STOP. Digraph* $\overrightarrow{G}$ *does not contain a cycle with negative cost.*

Cycle with negative cost in a digraph

Algorithm

Algorithm for searching a cycle with negative cost *in an arc weighted digraph* $\overrightarrow{G} = (V, H, c)$ *with general arc cost.*

- **Step 1: Initialization.**
  Set $t(i) := 0$ *and* $x(i) := 0$ *for all* $i \in V$.
  Set $\mathcal{E} := V$.
- **Step 2:** *Extract* $r \in \mathcal{E}$, *set* $\mathcal{E} := \mathcal{E} - \{r\}$.
  For every arc (r, j), $(r, j) \in H^+(r)$ *do:*
  If $t(j) > t(r) + c(r, j)$ *then*
  set $t(j) := t(r) + c(r, j)$, $x(j) := r$, $\mathcal{E} := \mathcal{E} \cup \{j\}$ *and try whether vertex* j *is an element of sequence:*

$$x(j), x(x(j)), x(x(x(j))), \ldots$$

  If yes you have just found a cycle with negative cost containing vertex j.
  STOP.
  If no then GOTO Step 3.
- **Step 3:** *If* $\mathcal{E} \neq \emptyset$ *then GOTO Step 2.*
  If $\mathcal{E} = \emptyset$ *then STOP. Digraph* $\overrightarrow{G}$ *does not contain a cycle with negative cost.*

Cycle with negative cost in a digraph

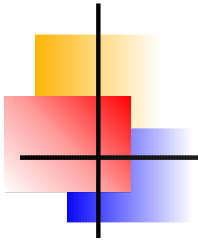
Algorithm

Algorithm for searching a cycle with negative cost *in an arc weighted digraph* $\overrightarrow{G} = (V, H, c)$ *with general arc cost.*

- **Step 1: Initialization.**
  Set $t(i) := 0$ *and* $x(i) := 0$ *for all* $i \in V$.
  Set $\mathcal{E} := V$.
- **Step 2:** *Extract* $r \in \mathcal{E}$, *set* $\mathcal{E} := \mathcal{E} - \{r\}$.
  For every arc (r, j), $(r, j) \in H^+(r)$ *do:*
  If $t(j) > t(r) + c(r, j)$ *then*
  set $t(j) := t(r) + c(r, j)$, $x(j) := r$, $\mathcal{E} := \mathcal{E} \cup \{j\}$ *and try whether vertex* j *is an element of sequence:*

$$x(j), x(x(j)), x(x(x(j))), \ldots$$

  If yes you have just found a cycle with negative cost containing vertex j. *STOP.*
  If no then GOTO Step 3.
- **Step 3:** *If* $\mathcal{E} \neq \emptyset$ *then GOTO Step 2.*
  If $\mathcal{E} = \emptyset$ *then STOP. Digraph* $\overrightarrow{G}$ *does not contain a cycle with negative cost.*



Cycle with negative cost in a digraph

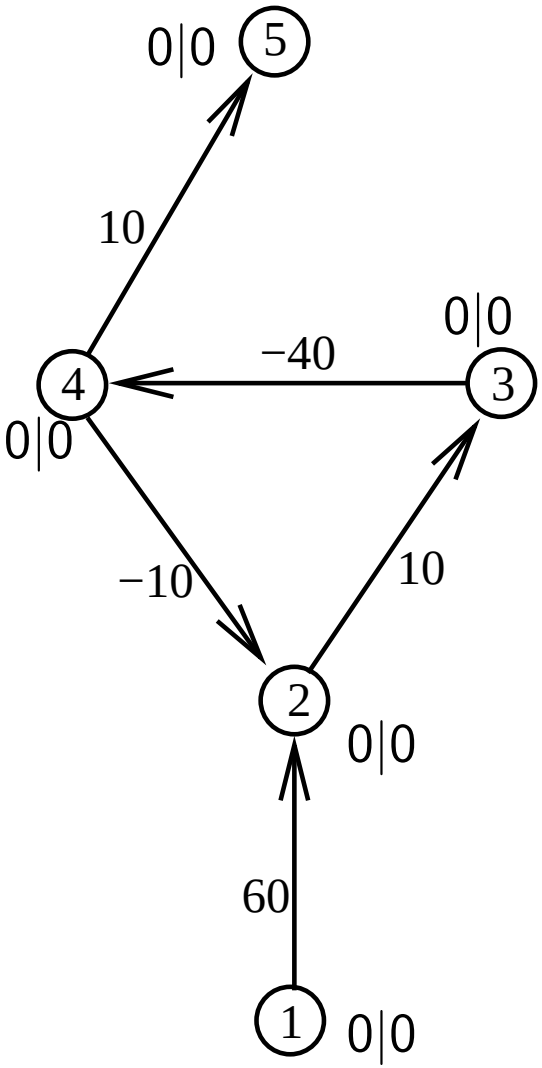
Remark

The sequence

$$x(j), x(x(j)), x(x(x(j))), ...$$

in the second step of last algorithm is necessary to enumerate until it contains vertex j or such vertex $k = x(x(\ldots x(j)\ldots))$ for which it holds $x(k) = 0$.

Cycle with negative cost in a digraph – Example

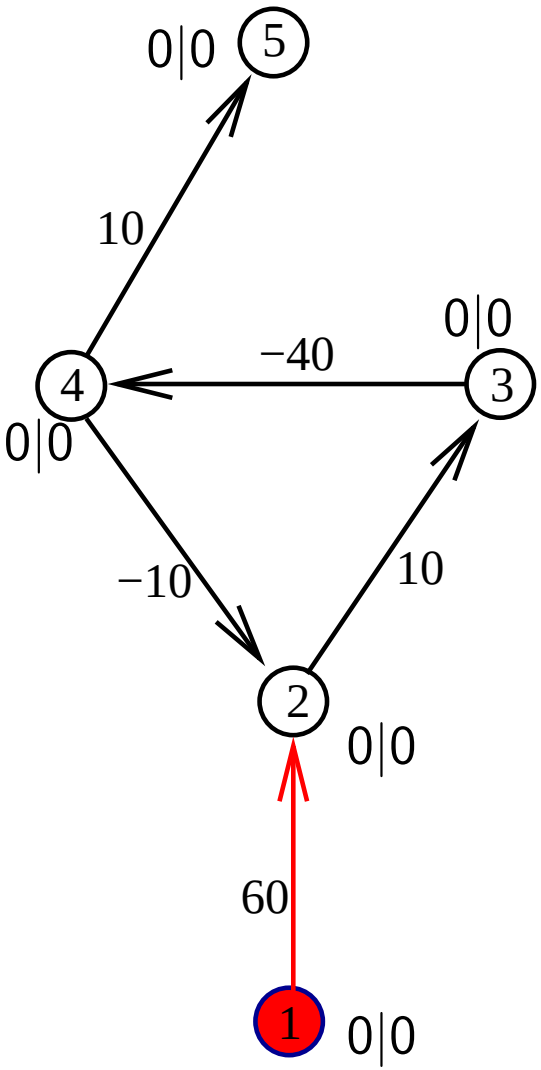


	r	$t(r)$	1	2	3	4	5	
			$t(v)\|x(v)$					the set $\mathcal{E}$
	-		0|0	0|0	0|0	0|0	0|0	1 2 3 4 5
1.	1	0						2 3 4 5
2.	2	0						3 4 5
3.	3	0				−40|3		4 5
4.	4	−40		−50|4			−30|4	5 2
5.	5	0						2
6.	2	−50			−40|2			3

$$x^{(k)}(j) = \underbrace{x(x(\ldots x(}_{k\text{-times}} j)\ldots x(j)\ldots))$$

	j	$x(j)$	$x^{(2)}(j)$	$x^{(3)}(j)$	$x^{(4)}(j)$	$x^{(5)}(j)$
1.	-					
2.	-					
3.	4	3	0			
4.	2	4	3	0		
4.	5	4	3	0		
5.	-					
6.	3	2	4	3		

Cycle with negative cost in a digraph – Example

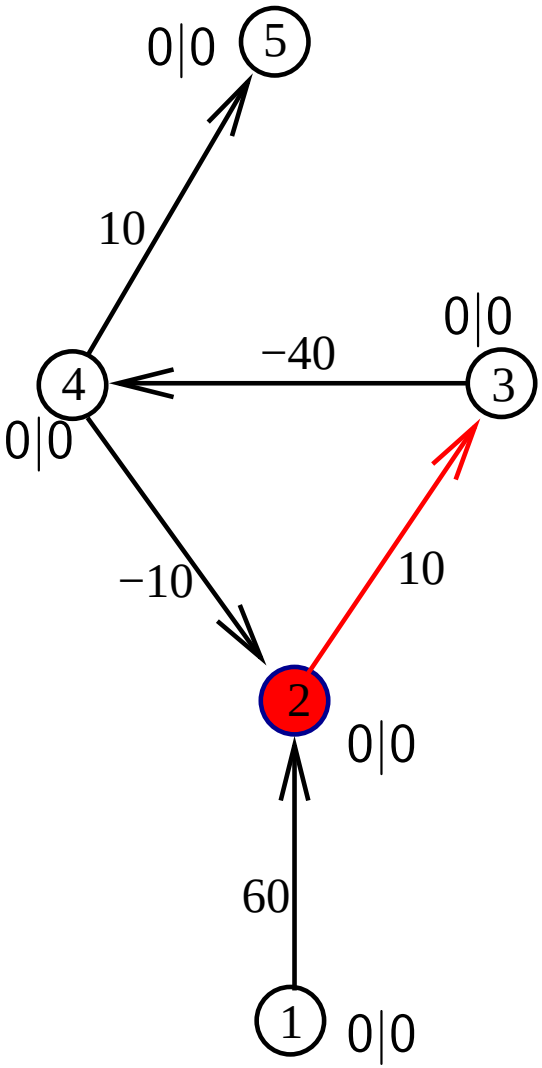


	r	$t(r)$	1	2	3	4	5	
			$t(v)\|x(v)$					the set $\mathcal{E}$
	-		0|0	0|0	0|0	0|0	0|0	1 2 3 4 5
1.	1	0						2 3 4 5
2.	2	0						3 4 5
3.	3	0				−40|3		4 5
4.	4	−40		−50|4			−30|4	5 2
5.	5	0						2
6.	2	−50			−40|2			3

$$x^{(k)}(j) = \underbrace{x(x(\ldots x(}_{k\text{-times}} j)\ldots x(j)\ldots))$$

	j	$x(j)$	$x^{(2)}(j)$	$x^{(3)}(j)$	$x^{(4)}(j)$	$x^{(5)}(j)$
1.	-					
2.	-					
3.	4	3	0			
4.	2	4	3	0		
4.	5	4	3	0		
5.	-					
6.	3	2	4	3		

Cycle with negative cost in a digraph – Example

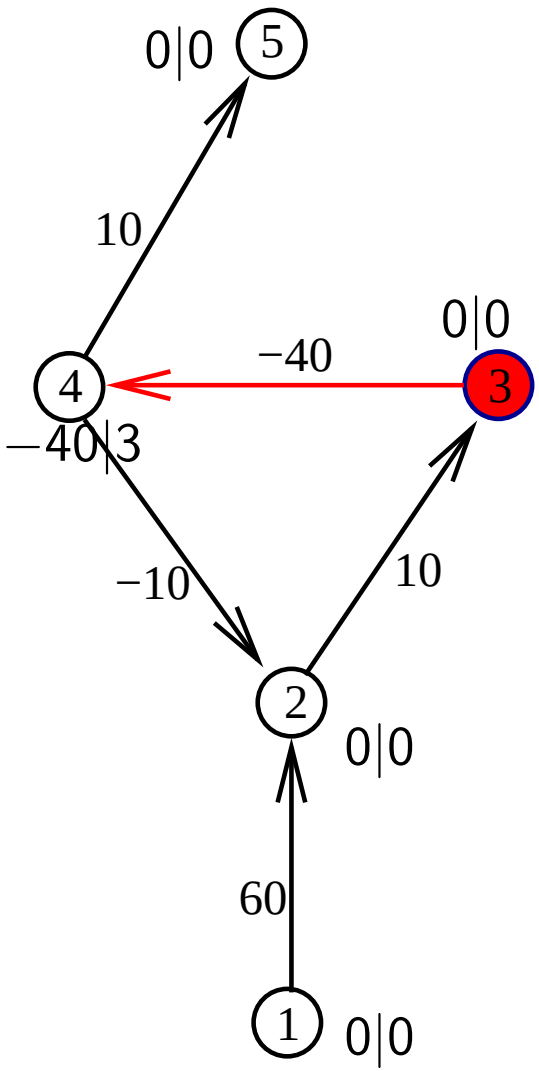


	r	$t(r)$	1	2	3	4	5	
					$t(v)\|x(v)$			the set $\mathcal{E}$
	-		0|0	0|0	0|0	0|0	0|0	1 2 3 4 5
1.	1	0						2 3 4 5
2.	2	0						3 4 5
3.	3	0				−40|3		4 5
4.	4	−40		−50|4			−30|4	5 2
5.	5	0						2
6.	2	−50			−40|2			3

$$x^{(k)}(j) = \underbrace{x(x(\ldots x(}_{k\text{-times}} j) \ldots x(j) \ldots))$$

	j	$x(j)$	$x^{(2)}(j)$	$x^{(3)}(j)$	$x^{(4)}(j)$	$x^{(5)}(j)$
1.	-					
2.	-					
3.	4	3	0			
4.	2	4	3	0		
4.	5	4	3	0		
5.	-					
6.	3	2	4	3		

Cycle with negative cost in a digraph – Example

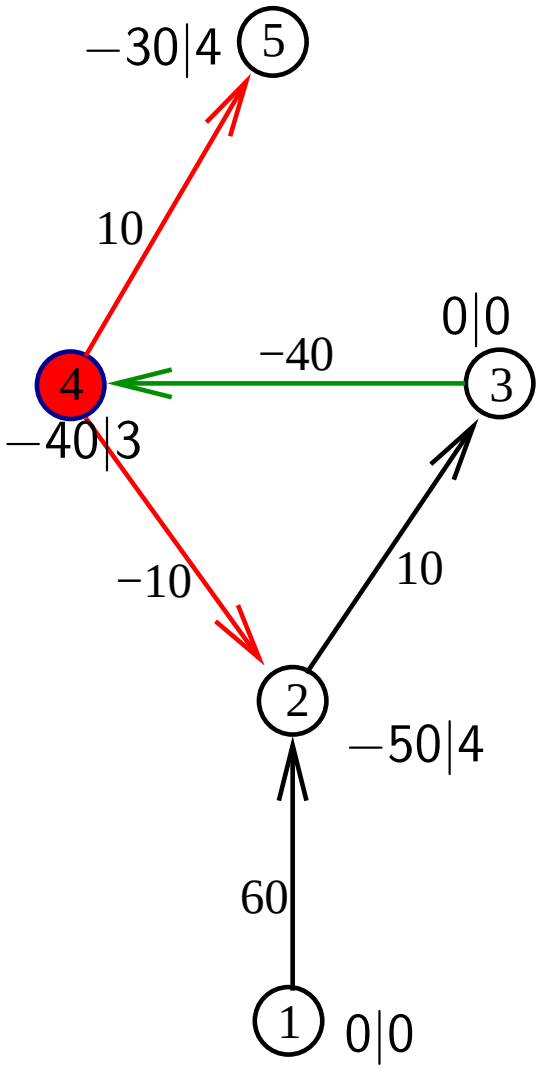


	r	$t(r)$	1	2	3	4	5	
					$t(v)\|x(v)$			the set $\mathcal{E}$
	-		0|0	0|0	0|0	0|0	0|0	1 2 3 4 5
1.	1	0						2 3 4 5
2.	2	0						3 4 5
3.	3	0				−40|3		4 5
4.	4	−40		−50|4			−30|4	5 2
5.	5	0						2
6.	2	−50			−40|2			3

$$x^{(k)}(j) = \underbrace{x(x(\ldots x(}_{k\text{-times}} j)\ldots x(j)\ldots))$$

	j	$x(j)$	$x^{(2)}(j)$	$x^{(3)}(j)$	$x^{(4)}(j)$	$x^{(5)}(j)$
1.	-					
2.	-					
3.	4	3	0			
4.	2	4	3	0		
4.	5	4	3	0		
5.	-					
6.	3	2	4	3		

Cycle with negative cost in a digraph – Example

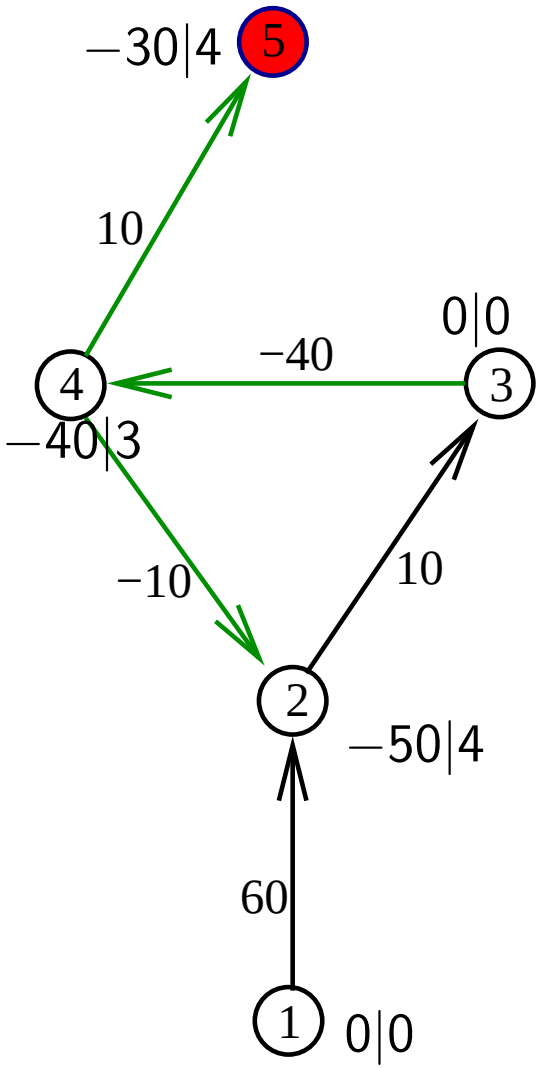


	r	$t(r)$	1	2	3	4	5	
					$t(v)\|x(v)$			the set $\mathcal{E}$
	-		0|0	0|0	0|0	0|0	0|0	1 2 3 4 5
1.	1	0						2 3 4 5
2.	2	0						3 4 5
3.	3	0				−40|3		4 5
4.	4	−40		−50|4			−30|4	5 2
5.	5	0						2
6.	2	−50			−40|2			3

$$x^{(k)}(j) = \underbrace{x(x(\ldots x(}_{k\text{-times}} j)\ldots x(j)\ldots))$$

	j	$x(j)$	$x^{(2)}(j)$	$x^{(3)}(j)$	$x^{(4)}(j)$	$x^{(5)}(j)$
1.	-					
2.	-					
3.	4	3	0			
4.	2	4	3	0		
4.	5	4	3	0		
5.	-					
6.	3	2	4	3		

Cycle with negative cost in a digraph – Example

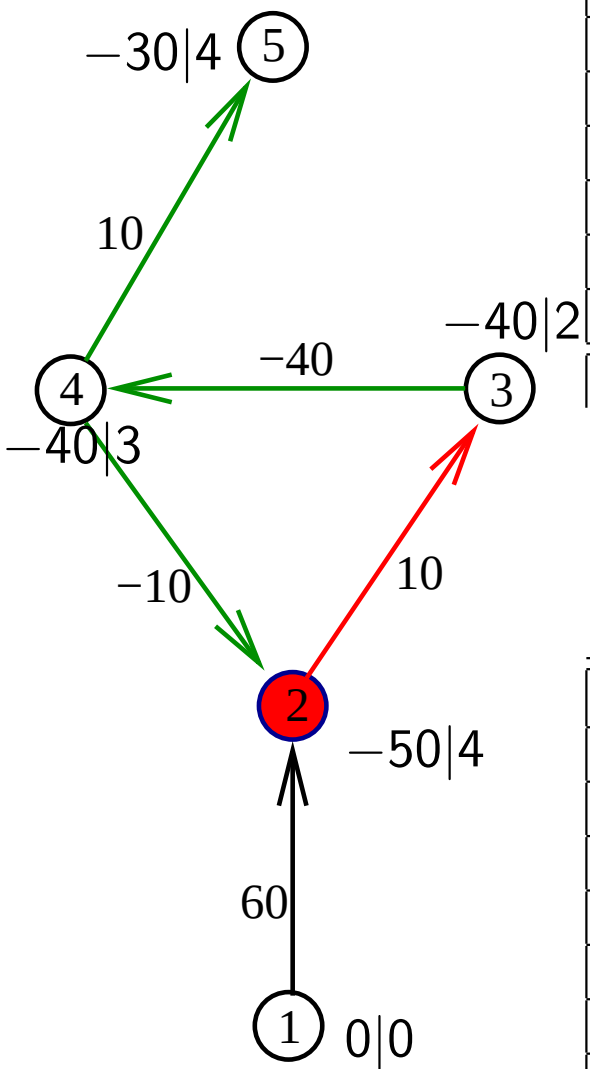


	r	$t(r)$	1	2	3	4	5	
					$t(v)\|x(v)$			the set $\mathcal{E}$
	-		0|0	0|0	0|0	0|0	0|0	1 2 3 4 5
1.	1	0						2 3 4 5
2.	2	0						3 4 5
3.	3	0				−40|3		4 5
4.	4	−40		−50|4			−30|4	5 2
5.	5	0						2
6.	2	−50			−40|2			3

$$x^{(k)}(j) = \underbrace{x(x(\ldots x(}_{k\text{-times}} j) \ldots x(j) \ldots))$$

	j	$x(j)$	$x^{(2)}(j)$	$x^{(3)}(j)$	$x^{(4)}(j)$	$x^{(5)}(j)$
1.	-					
2.	-					
3.	4	3	0			
4.	2	4	3	0		
4.	5	4	3	0		
5.	-					
6.	3	2	4	3		

Cycle with negative cost in a digraph – Example



	r	$t(r)$	1	2	3	4	5	
					$t(v)\|x(v)$			the set $\mathcal{E}$
	-		0|0	0|0	0|0	0|0	0|0	1 2 3 4 5
1.	1	0						2 3 4 5
2.	2	0						3 4 5
3.	3	0				−40|3		4 5
4.	4	−40		−50|4			−30|4	5 2
5.	5	0						2
6.	2	−50			−40|2			3

$$x^{(k)}(j) = \underbrace{x(x(\ldots x(}_{k\text{-times}} j)\ldots x(j)\ldots))$$

	j	$x(j)$	$x^{(2)}(j)$	$x^{(3)}(j)$	$x^{(4)}(j)$	$x^{(5)}(j)$
1.	-					
2.	-					
3.	4	3	0			
4.	2	4	3	0		
4.	5	4	3	0		
5.	-					
6.	3	2	4	**3**		

Cycle with negative cost in a digraph

Floyd's algorithm 6 (page. 117) can be modified so that it will find a negative cycle in the case of a digraph with general arc weights.
It suffices to define initial matrix **C** in step 1. as follows:

$$c_{ij} = \begin{cases} c(i,j), & \text{ak } \{i,j\} \in H, \quad \text{resp.} \quad (i,j) \in H \\ \infty, & \text{ak } \{i,j\} \notin H, \quad \text{resp.} \quad (i,j) \notin H \end{cases}$$

Matrix **C** contains (by contrast to standard Floyd's algorithm) ∞ on main diagonal.
Matrix **X** stays without change.
Step 2. is the same.
Attention! It is necessary to change elements on main diagonal, too!!!

As soon as a negative number c_{jj} appears on main diagonal we have recovered a negative cycle containing vertex j

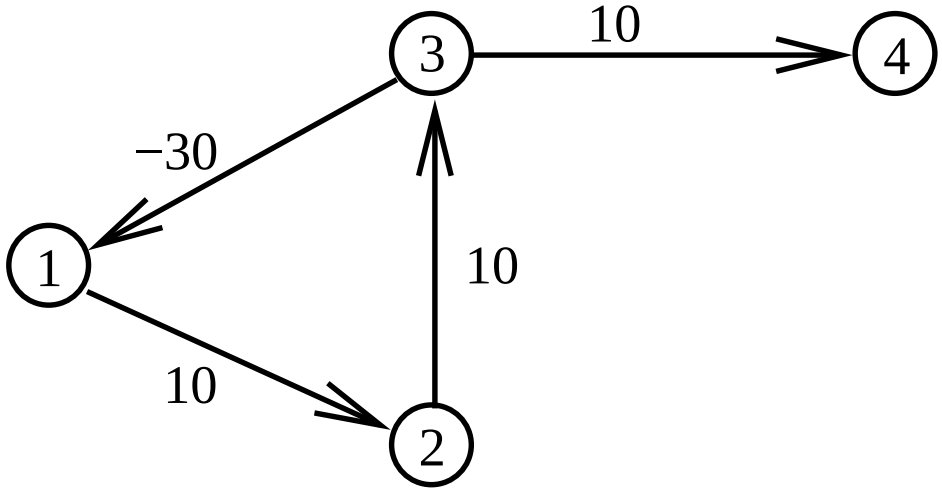
This cycle can be constructed using matrix $\mathbb{X}$.

Example

It is necessary to find out whether digraph represented by following diagram contains a cycle with negative cost.

We write matrices **C** *and* **X** *and execute several times step 2. of Floyd's algorithm for* $k = 1, 2, 3$.

Development of matrices **C** *a* **X** *is illustrated on following tables.*



Obr.: Digraf k príkladu 2.

Príklad

C

∞	10	∞	∞
∞	∞	10	∞
-30	∞	∞	10
∞	∞	∞	∞

C po $k = 1$

∞	10	∞	∞
∞	∞	10	∞
-30	-20	∞	10
∞	∞	∞	∞

C po $k = 2$

∞	10	20	∞
∞	∞	10	∞
-30	-20	**-10**	10
∞	∞	∞	∞

C po $k = 3$

-10	0	20	30
-20	**-10**	10	20
-30	-20	**-10**	10
∞	∞	∞	∞

X

-	1	-	-
-	-	2	-
3	-	-	3
-	-	-	-

X po $k = 1$

-	1	-	-
-	-	2	-
3	1	-	3
-	-	-	-

X po $k = 2$

-	1	2	-
-	-	2	-
3	**1**	**2**	3
-	-	-	-

X po $k = 3$

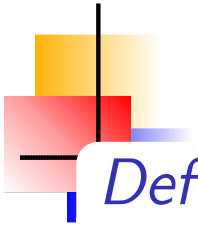
3	1	2	3
3	1	2	3
3	1	2	3
-	-	-	-

After computing third couple of matrices **C**, **X** negative number -10 appears on entry c_{33} what suffices for constatation that examined digraph contains a negative cycle containing vertex 3.

Third row of pointer matrix **X** says that last but one vertex of this cycle is $x_{33} = \mathbf{2}$ last but two vertex is $x_{32} = \mathbf{1}$ and in front of it is vertex $x_{31} = \mathbf{3}$, which is first and also last vertex of cycle.

Searched cycle with negative cost is:

$$3, (3,1), 1, (1,2), 2, (2,3), 3.$$



Path with largest reliability

Definition

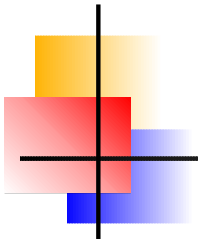
Let $G = (V, H, c)$ be a graph where edge weight $c(h)$ represents the reliability of edge h (reliability – probability of successful travel along the edge h), i.e. $0 \leq c(h) \leq 1$.
Let $\mu(u, v)$ be a u–v path.
The realiability *$s(\mu(u, v))$* **of path** *$\mu(u, v)$ is defined as follows:*

$$s(\mu(u,v)) = \prod_{h \in \mu(u,v)} c(h) \ .$$

u–v u–v **path with maximal probability** *is that u–v path $\mu(u, v)$, which has maximum relibility from all u–v paths.*

Theorem

Let $G = (V, H, c)$ be a graph where $c(h) > 0$ is the reliability of edge $h \in H$.
u–v path $\mu(u, v)$ is the u–v path with maximum reliability in $G = (V, H, c)$ if and only if $\mu(u, v)$ is the shortest path in graph $\overline{G} = (V, H, \overline{c})$ with edge cost $\overline{c}$ fulfilling $\overline{c}(i, j) = -\log_z (c(i, j))$ (where $z > 1$).[a]



Path with largest reliability

Most reliable u-v path

The goal: to maximize

$$s(\mu(u,v)) = \prod_{h \in \mu(u,v)} c(h)$$

$d(\mu(u,v))$ is maximal $\Leftrightarrow$

$\Leftrightarrow \log d(\mu(u,v))$ is maximal.

The goal: to maximize

$$\log s(\mu(u,v)) = \log \prod_{h \in \mu(u,v)} c(h) = \sum_{h \in \mu(u,v)} \log c(h)$$

$\sum_{h \in \mu(u,v)} \log c(h)$ is maximal $\Leftrightarrow$

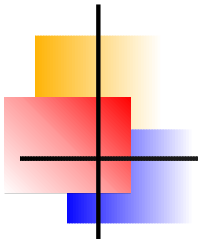
$\Leftrightarrow \sum_{h \in \mu(u,v)} -\log c(h)$ je minimal.

The goal: to minimize $\sum_{h \in \mu(u,v)} -\log c(h)$

The shortest u-v path

The goal: to minimize

$$d(\mu(u,v)) = \sum_{h \in \mu(u,v)} c(h)$$



Path with largest reliability

Most reliable u-v path

The goal: to maximize

$$s(\mu(u,v)) = \prod_{h \in \mu(u,v)} c(h)$$

$d(\mu(u,v))$ is maximal $\Leftrightarrow$

$\Leftrightarrow \log d(\mu(u,v))$ is maximal.

The goal: to maximize

$$\log s(\mu(u,v)) = \log \prod_{h \in \mu(u,v)} c(h) = \sum_{h \in \mu(u,v)} \log c(h)$$

$\sum_{h \in \mu(u,v)} \log c(h)$ is maximal $\Leftrightarrow$

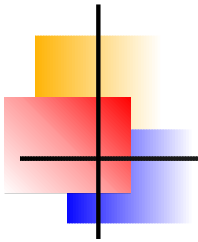
$\Leftrightarrow \sum_{h \in \mu(u,v)} -\log c(h)$ je minimal.

The goal: to minimize $\sum_{h \in \mu(u,v)} -\log c(h)$

The shortest u-v path

The goal: to minimize

$$d(\mu(u,v)) = \sum_{h \in \mu(u,v)} c(h)$$



Path with largest reliability

Most reliable u-v path

The goal: to maximize

$$s(\mu(u,v)) = \prod_{h \in \mu(u,v)} c(h)$$

$d(\mu(u,v))$ is maximal $\Leftrightarrow$

$\Leftrightarrow \log d(\mu(u,v))$ is maximal.

The goal: to maximize

$$\log s(\mu(u,v)) = \log \prod_{h \in \mu(u,v)} c(h) = \sum_{h \in \mu(u,v)} \log c(h)$$

$\sum_{h \in \mu(u,v)} \log c(h)$ is maximal $\Leftrightarrow$

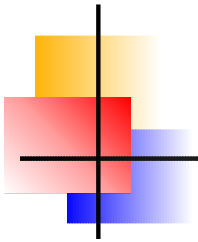
$\Leftrightarrow \sum_{h \in \mu(u,v)} -\log c(h)$ je minimal.

The goal: to minimize $\sum_{h \in \mu(u,v)} -\log c(h)$

The shortest u-v path

The goal: to minimize

$$d(\mu(u,v)) = \sum_{h \in \mu(u,v)} c(h)$$



Path with largest reliability

Most reliable u-v path

The goal: to maximize

$$s(\mu(u,v)) = \prod_{h \in \mu(u,v)} c(h)$$

$d(\mu(u,v))$ is maximal $\Leftrightarrow$

$\Leftrightarrow \log d(\mu(u,v))$ is maximal.

The goal: to maximize

$$\log s(\mu(u,v)) = \log \prod_{h \in \mu(u,v)} c(h) = \sum_{h \in \mu(u,v)} \log c(h)$$

$\sum_{h \in \mu(u,v)} \log c(h)$ is maximal $\Leftrightarrow$

$\Leftrightarrow \sum_{h \in \mu(u,v)} -\log c(h)$ je minimal.

The goal: to minimize $\sum_{h \in \mu(u,v)} -\log c(h)$

The shortest u-v path

The goal: to minimize

$$d(\mu(u,v)) = \sum_{h \in \mu(u,v)} c(h)$$

Path with largest reliability

Most reliable u-v path

The goal: to maximize

$$s(\boldsymbol{\mu}(u,v)) = \prod_{h \in \boldsymbol{\mu}(u,v)} c(h)$$

$d(\boldsymbol{\mu}(u,v))$ is maximal $\Leftrightarrow$
$\Leftrightarrow \log d(\boldsymbol{\mu}(u,v))$ is maximal.

The goal: to maximize

$$\log s(\boldsymbol{\mu}(u,v)) = \log \prod_{h \in \boldsymbol{\mu}(u,v)} c(h) = \sum_{h \in \boldsymbol{\mu}(u,v)} \log c(h)$$

$\sum_{h \in \boldsymbol{\mu}(u,v)} \log c(h)$ is maximal $\Leftrightarrow$
$\Leftrightarrow \sum_{h \in \boldsymbol{\mu}(u,v)} -\log c(h)$ je minimal.

The goal: to minimize $\sum_{h \in \boldsymbol{\mu}(u,v)} -\log c(h)$

The shortest u-v path

The goal: to minimize

$$d(\boldsymbol{\mu}(u,v)) = \sum_{h \in \boldsymbol{\mu}(u,v)} c(h)$$