Graph Theory and Complex Networks: An Introduction

Maarten van Steen

VU Amsterdam, Dept. Computer Science
Room R4.20, steen@cs.vu.nl

Chapter 05: Trees

Version: April 21, 2014

vrije Universiteit amsterdam

Contents

Introduction

Definition

A connected graph without cycles is a **tree**.

Connector problem: Set up a communication infrastructure such that the total costs are minimized.

Communication network: Set up an overlay network such that the total costs from a source to all destinations are minimized.

1. Formalities
2. Spanning trees
3. Routing in communication networks

Fundamentals: characterization (1)

Theorem

For any connected (simple) graph G with n vertices and m edges, $n \leq m+1$.

Proof by induction on m

- $m = 1 \Rightarrow n = 2 \Rightarrow$ OK. Consider G with $k > 1$ edges.
- Assume G has a cycle C. Let $e \in E(C)$ and $G^* = G - e$.
  - G^* is still connected
  - $n = |V(G^*)| \leq |E(G^*)| + 1 = k - 1 + 1 = k \leq k + 1$.
- Assume G is acyclic. Let P be a longest path in G, connecting vertices u and w.
  - P is longest path $\Rightarrow \delta(u) = \delta(w) = 1$
  - Let $G^* = G - u \Rightarrow |E(G^*)| = |E(G)| - 1 = k - 1$
  - $|V(G^*)| = n - 1 \leq |E(G^*)| + 1 = k \Rightarrow n \leq k + 1$

Fundamentals: characterization (2)

Theorem

A connected graph G with n vertices and m edges for which $n = m+1$, is a tree.

Proof by contradiction

- Assume G contains a cycle C and let $e \in E(C)$.
- $G^* = G - e$ is connected
  $\Rightarrow n = |V(G^*)| \leq |E(G^*)| + 1 = (m-1) + 1 = m$.
  Contradicts fact that $n = m+1$. G must be acyclic, i.e., a tree.

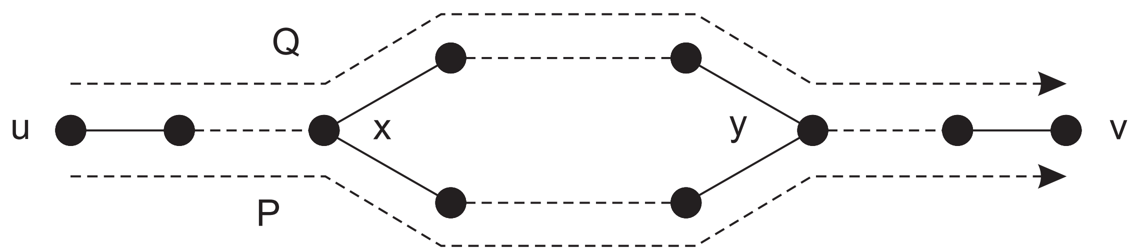
Fundamentals: characterization (3)

Theorem

A graph G is a tree iff $\forall u, v \in V(G) : \exists!(u, v)$-path.
(Notation: $\exists!$ means exists exactly one.)

Proof G tree $\Rightarrow \forall u, v \in V(G) : \exists!(u, v)$-path

- Let $u, v \in V(G)$ and (u, v)-path P.
- Assume another distinct (u, v)-path Q.
- Let x be last vertex common to P and Q, and y first common one succeeding $x \Rightarrow$ have identified a cycle:



Fundamentals: characterization (3)

Proof $\forall u, v \in V(G) : \exists!(u, v)$-path $\Rightarrow$ G is a tree

- By contradiction: assume G is not a tree.
- **Note:** G is connected.
- G is connected, not a tree $\Rightarrow$ there exists a cycle $C = [v_1, v_2, \ldots, v_n = v_1]$.
- $\forall v_i, v_j \in V(C)$: there are *two* distinct paths:
  - $P_{i \to j} = [v_i, v_{i+1}, \ldots, v_{j-1}, v_j]$
  - $P_{j \to i} = [v_i, v_{i-1}, \ldots, v_{j+1}, v_j]$

Fundamentals

Theorem

An edge e of a graph G is a cut edge if and only if e is not part of any cycle of G.

Proof e **is not part of a cycle** $\Rightarrow$ e **is a cut edge of** G

- By contradiction: assume that $e = \langle u, v \rangle$ is not a cut edge $\Rightarrow$ u, v in the same component in $G - e$.
- $\exists(u, v)$-path P in $G - e$.
- But: $P + e$ is a cycle in G. Contradiction.

Fundamentals

Proof e is cut edge $\Rightarrow$ e is not in any cycle of G

- By contradiction: assume $e = \langle u, v \rangle$ was part of a cycle C.
- Let x and y be in different components of $G - e$.
- e is cut edge $\Rightarrow \exists(x, y)$-path P in G and $e \in E(P)$.
- Assume u precedes v when traversing from x to y.
  $P_1 = (x, u)$-part of P, $P_2 = (v, y)$-part of P.
- **Note:** $C - e$ is (u, v)-path in $G - e$.
- u^* is first vertex common to P_1 and $C - e$;
  v^* is first vertex common to P_2 and $C - e$.
- $x \xrightarrow{P_1} u^* \xrightarrow{C-e} v^* \xrightarrow{P_2} y$ is an (x, y)-path in $G - e$, contradicting that x and y are in different components.

Fundamentals: characterization (4)

Theorem

A connected graph G is a tree if and only if every edge is a cut edge.

Proof

G is tree $\Rightarrow \forall e \in E(G) : e$ is cut edge: Let G be a tree and $e \in E(G)$.
G contains no cycles $\Rightarrow e$ not contained in any cycle $\Rightarrow e$ is cut edge.

$\forall e \in E(G) : e$ is cut edge $\Rightarrow G$ is tree: Assume G contains a cycle $C \Rightarrow \forall e \in E(C) : e$ is not a cut edge $\Rightarrow$ not every edge in G is a cut edge, contradicting our starting-point.

Spanning tree

Definition

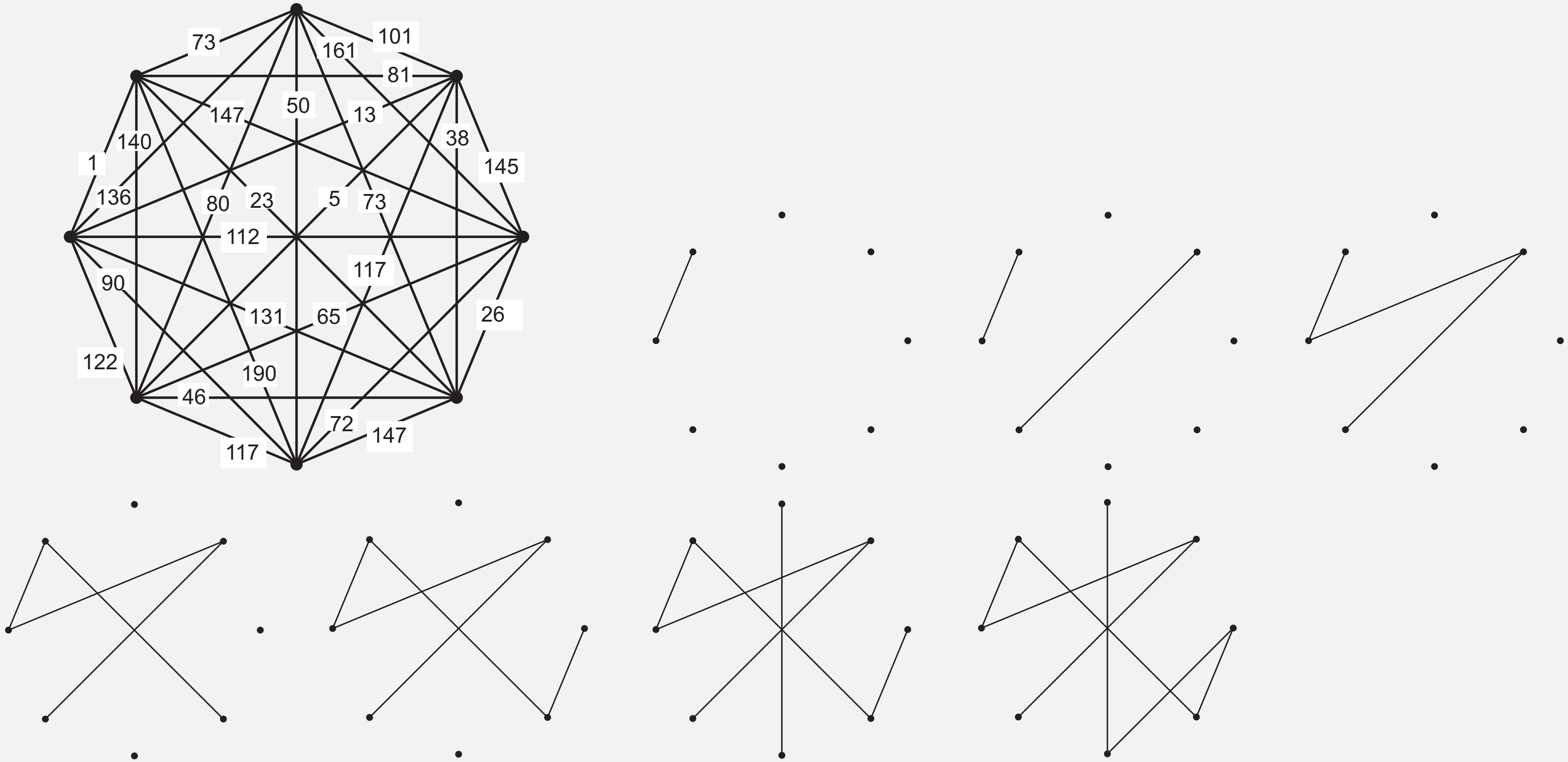
$T \subseteq G$ is a minimal spanning tree of G iff $V(T) = V(G)$ and $\sum_{e \in E(T)} w(e)$ is minimal.

Algorithm (Kruskal)

G is connected, weighted graph. $\forall e \in E(G) : w(e) \in \mathbb{R}$. Choose edge e_1 with minimal weight.

1. *Assume edges $E_k = \{e_1, e_2, \ldots, e_k\}$ have been chosen so far. Choose next edge $e_{k+1} \in E(G) \setminus E_k$ such that:*
   (1) *$G_{k+1} = G[\{e_1, e_2, \ldots, e_k, e_{k+1}\}]$ is acyclic (but not necessarily connected).*
   (2) *$\forall e \in E(G) \setminus E_k : w(e) \geq w(e_{k+1})$.*
2. *Stop when no such edge e_{k+1} can be selected.*

Example Kruskal’s algorithm



Correctness Kruskal's algorithm

Theorem

Any spanning tree T_{opt} of a weighted connected graph G constructed by Kruskal's algorithm has minimal weight.

Proof by construction and contradiction

- Notation: $\forall$spanning $T \neq T_{opt}$, $\imath(T)$ smallest index $i : e_i \notin E(T)$.
- Assume T_{opt} is not optimal. Let T be spanning with maximal $\imath(T)$.
- $\imath(T) = k \Rightarrow e_1, e_2, \ldots, e_{k-1} \in E(T) \cap E(T_{opt})$.
- **Note:** $T + e_k$ contains a unique cycle C (**Why?**)

Correctness Kruskal's algorithm

Proof by construction and contradiction (cntd)

- Let $\hat{e} \in \{E(C) \cap E(T)\} \backslash E(T_{opt})$.
- $\hat{e} \in E(C) \Rightarrow \hat{T} = (T + e_k) - \hat{e}$ is connected and spanning tree of G.
- $w(\hat{T}) = w(T) + w(e_k) - w(\hat{e})$ with $w(\hat{e}) \geq w(e_k)$ (**Why?**)
- Implication: $\hat{T}$ must be optimal.
- However: $e_k \in E(\hat{T}) \Rightarrow \imath(\hat{T}) > \imath(T)$. Contradiction.

Routing

Basics

In a communication network, each node u maintains a routing table $\mathbf{R}_u$ with $\mathbf{R}_u[i,j] = k$ meaning that messages from i to j should be forwarded to neighbor k.

Issue

Messages to destination u should follow a path along a spanning tree **rooted at u**.

Technically

We need to construct a spanning tree optimized for all (v,u)-paths, called a **sink tree**.

Dijkstra's algorithm

Algorithm (Dijkstra, sink tree construction)

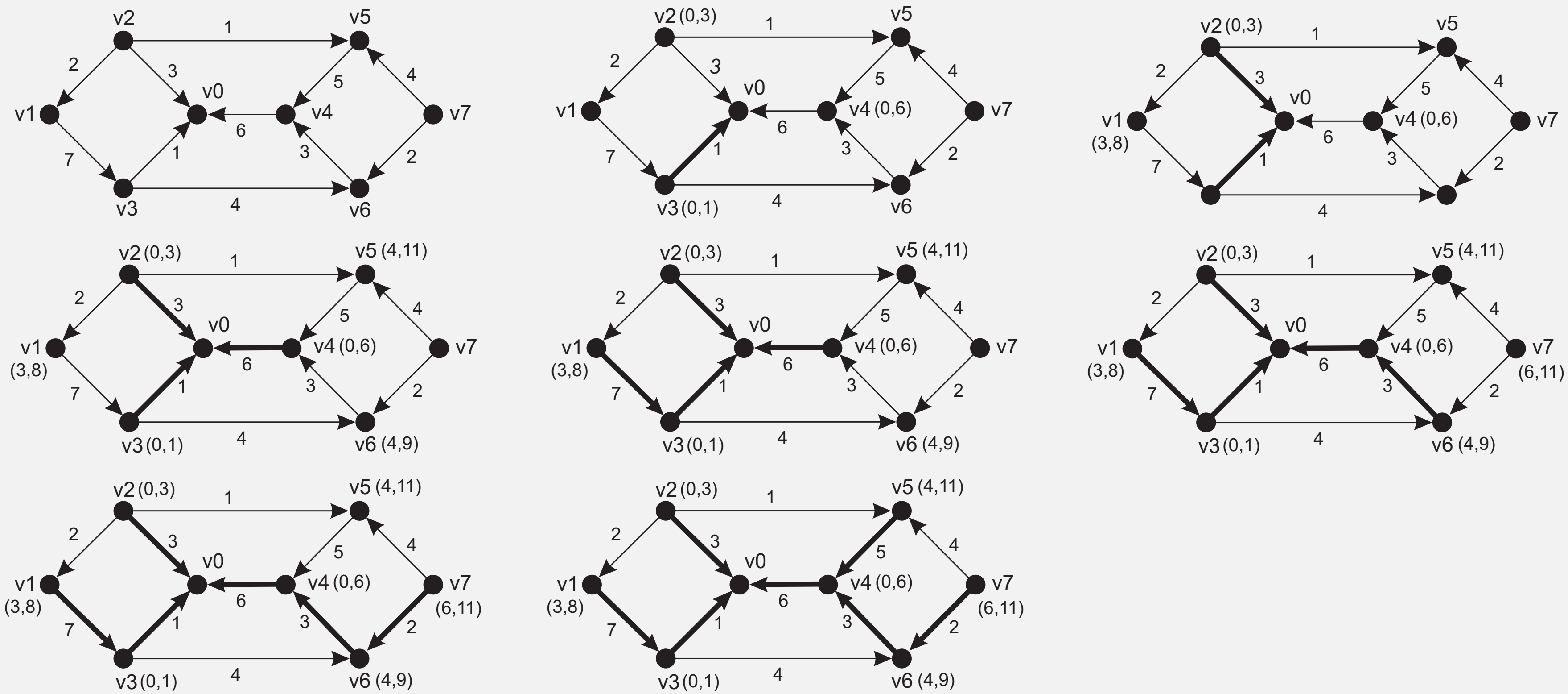
D is directed, weighted graph with nonnegative weights.
$\forall u : v \in S_t(u) \Rightarrow$ *shortest* (v,u)*-path found.*
$\forall v : \mathbf{L}(v) = (L_1(v), L_2(v))$ *with*

- $L_1(v)$: *vertex succeeding v in shortest* (v,u)*-path so far.*
- $L_2(v)$: *total weight (length) of that path.*

Let $R_t(u) = S_t(u) \cup_{v \in S_t(u)} N(v)$, *with* $N(v) = \{w | \exists$ *arc* $\langle \overrightarrow{w,v} \rangle \}$.

1. *Initialize* $t \leftarrow 0$; $\mathbf{L}(u) \leftarrow (u,0)$; $\forall v \neq u : \mathbf{L}(v) \leftarrow (-,\infty)$; $S_0(u) \leftarrow \{u\}$.
2. $\forall y \in R_t(u) \backslash S_t(u)$, *select* $x \in S_t(u) : L_2(x) + w(\langle \overrightarrow{y,x} \rangle)$ *is minimal.* *Set* $\mathbf{L}(y) \leftarrow \left(x, L_2(x) + w(\langle \overrightarrow{y,x} \rangle)\right)$.
3. *Let* $z \in R_t(u) \backslash S_t(u) : L_2(z)$ *is minimal. Set* $S_{t+1}(u) \leftarrow S_t(u) \cup \{z\}$. *If* $S_{t+1}(u) = V(G)$, *stop. Otherwise,* $t \leftarrow t+1$, *recompute* $R_t(u)$ *and repeat previous step.*

Example: Dijkstra’s shortest path algorithm



Correctness of Dijkstra's algorithm

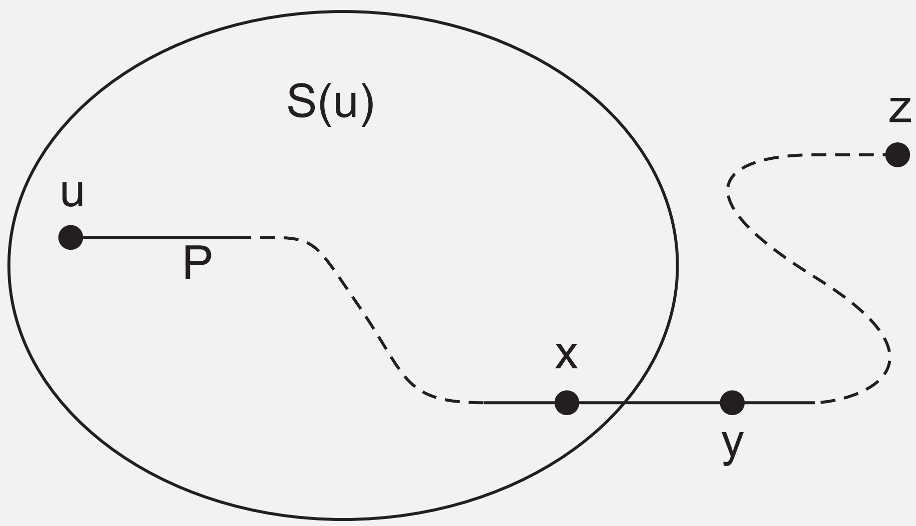
Theorem

Applying Dijkstra's algorithm vertex $u \in V(D)$, each time a vertex z is added to $S_t(u)$, $L_2(z)$ corresponds to the shortest (z, u)-path in D.

Proof by contradiction

- Let $d(w, u)$ be total weight of shortest (w, u)-path.
- Let z be first vertex added to $S_t(u)$, such that $L_2(z) > d(z, u)$.
- **Note:** $L_2(z) < \infty$ (otherwise z would never have been added).
- Let P be shortest (z, u)-path.
- Let y be last vertex on P not in $S_t(u)$, and x its successor (and thus in $S_t(u)$).

Correctness of Dijkstra's algorithm



- **Note:** $L_2(x) = d(x,u)$ and $L_2(y) \leq L_2(x) + w(\langle\overrightarrow{y,x}\rangle)$.
- **Also:** y is on shortest (z,u)-path $\Rightarrow L_2(y) = d(y,u)$.
- y was not selected at step $t \Rightarrow L_2(z) \leq L_2(y)$.
- **Note:** $d(z,y) + d(y,u) = d(z,u)$
- $L_2(z) \leq L_2(y) = d(y,u) \leq d(y,u) + d(z,y) = d(z,u)$.
  Contradiction.

Decentralized routing

Observation

In order to execute Dijkstra's algorithm, each vertex should know the **topology** of the entire network.

Alternative

Let nodes tell their neighbors on shortest paths to other nodes discovered so far.

Observation

If a neighbor v of u knows about a path to w, and tells u, then u discovers a path to w (namely via v).

Bellman-Ford routing

Algorithm (Bellman-Ford)

- *Consider node v_i. We proceed in* ***rounds****: in every round t, each node evaluates its routing table $\mathbf{R}_i[j] = d^t(i,j)$ with:*

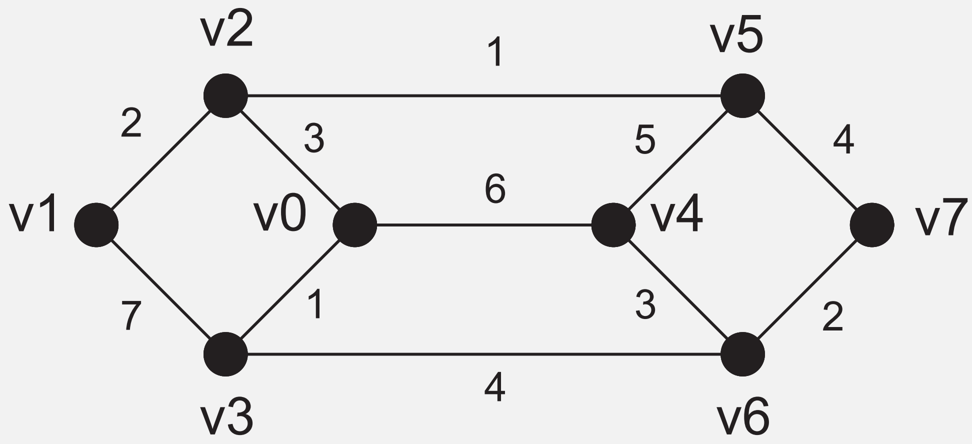
$$d^0(i,j) \leftarrow \begin{cases} 0 & \text{if } i = j \\ \infty & \text{otherwise} \end{cases}$$

- *Every round,* ***adjust*** *$d^t(i,j)$ to:*

$$d^{t+1}(i,j) \leftarrow \min_{k \in N(v_i)} w(\langle v_i, v_k \rangle) + d^t(k,j)$$

- *With $d^t(i,j)$ thus denoting the total weight of optimal (v_i, v_j)-path, found by v_i after t rounds.*

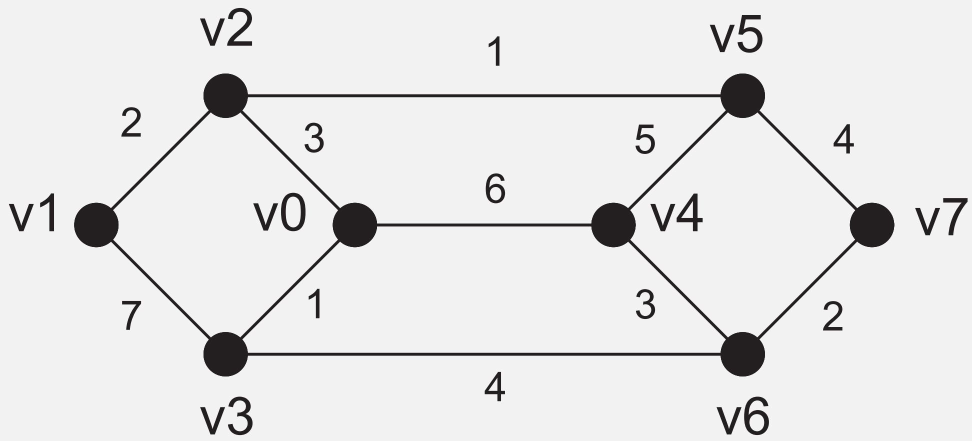
Example: Bellman-Ford



Destination

	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7
v_0 :	$(0, v_0)$		$(3, v_2)$	$(1, v_3)$	$(6, v_4)$			
v_1 :		$(0, v_1)$	$(2, v_2)$	$(7, v_3)$				
v_2 :	$(3, v_0)$	$(2, v_1)$	$(0, v_2)$			$(1, v_5)$		
v_3 :	$(1, v_0)$	$(7, v_1)$		$(0, v_3)$			$(4, v_6)$	
v_4 :	$(6, v_0)$				$(0, v_4)$	$(5, v_5)$	$(3, v_6)$	
v_5 :			$(1, v_2)$		$(5, v_4)$	$(0, v_5)$		$(4, v_7)$
v_6 :				$(4, v_3)$	$(3, v_4)$		$(0, v_6)$	$(2, v_7)$
v_7 :						$(4, v_5)$	$(2, v_6)$	$(0, v_7)$

Example: Bellman-Ford after 2 rounds



Destination

	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7
v_0 :	$(0, v_0)$	$(5, v_2)$	$(3, v_2)$	$(1, v_3)$	$(6, v_4)$	$(4, v_2)$	$(5, v_3)$	
v_1 :	$(5, v_2)$	$(0, v_1)$	$(2, v_2)$	$(7, v_3)$		$(3, v_2)$	$(11, v_3)$	
v_2 :	$(3, v_0)$	$(2, v_1)$	$(0, v_2)$	$(4, v_0)$	$(6, v_5)$	$(1, v_5)$		$(5, v_5)$
v_3 :	$(1, v_0)$	$(7, v_1)$	$(4, v_0)$	$(0, v_3)$	$(7, v_0)$		$(4, v_6)$	$(6, v_6)$
v_4 :	$(6, v_0)$		$(6, v_5)$	$(7, v_6)$	$(0, v_4)$	$(5, v_5)$	$(3, v_6)$	$(5, v_6)$
v_5 :	$(4, v_2)$	$(3, v_2)$	$(1, v_2)$		$(5, v_4)$	$(0, v_5)$	$(6, v_7)$	$(4, v_7)$
v_6 :	$(5, v_3)$	$(11, v_3)$		$(4, v_3)$	$(3, v_4)$	$(6, v_7)$	$(0, v_6)$	$(2, v_7)$
v_7 :			$(5, v_5)$	$(6, v_6)$	$(5, v_6)$	$(4, v_5)$	$(2, v_6)$	$(0, v_7)$

A note on efficiency

Observation

Dijkstra's algorithm roughly requires each node to inspect every other node once, implying a total of approximately n^2 steps.

The Bellman-Ford algorithm requires that for each node we inspect exactly the tables of each of its neighbors. Because we have $\sum \delta(v) = 2m$ with m the number of edges, there are a total of roughly $n \cdot m$ steps.

A note on efficiency