



La Sapienza

Università degli Studi di Roma

Dipartimento di Informatica e Sistemistica

RETI DI CALCOLATORI II

Teoria dei grafi ed algoritmi di routing

Emiliano Trevisani

trevisani@dis.uniroma1.it

A.A. 2008/2009

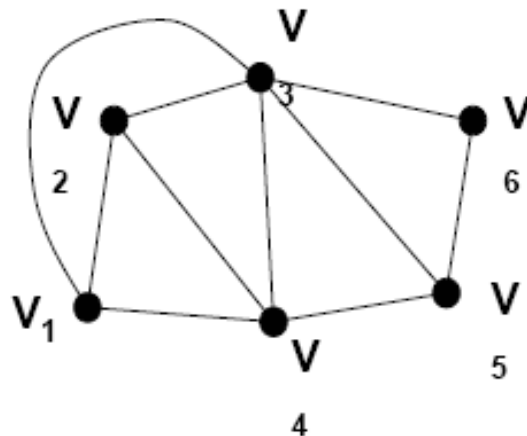
Teoria dei grafi

- Un grafo $G(V,E)$ è definito da:
 - un insieme V di nodi
 - un insieme E di rami; ciascun ramo connette una coppia di nodi
- I due vertici i e j sono detti **adiacenti** se il ramo $(i,j) \in E$ [i nodi i e j sono connessi]
- Il ramo (i,j) è detto **incidente** ai nodi i e j
- La cardinalità $|V|$ dell'insieme dei nodi è detta **ordine** del grafo G
- La cardinalità $|E|$ dell'insieme dei rami è detta **dimensione** del grafo G

Teoria dei grafi

- Un grafo può essere rappresentato utilizzando la **matrice delle adiacenze** A di dimensioni $|V| \times |V|$

$$A = [a_{ij}] \quad a_{ij} = \begin{cases} 1 & \text{if } (i, j) \in E \\ 0 & \text{otherwise} \end{cases}$$



$$\begin{array}{c} V_1 \\ V_2 \\ V_3 \\ V_4 \\ V_5 \\ V_6 \end{array} \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \end{pmatrix}$$

Teoria dei grafi

- ❑ Un **cammino** tra due nodi i e j è una sequenza di nodi e rami a partire dal nodo i al nodo j tale che ogni ramo è incidente al nodo precedente ed al successivo
- ❑ Un cammino in cui ogni nodo e ogni ramo appare una sola volta è detto **cammino semplice (simple path)**
- ❑ Il minimo numero di rami che compone un cammino tra due nodi i e j è detta **distanza** tra i due nodi
- ❑ Un **ciclo** è un cammino semplice in cui il nodo di partenza coincide con il nodo di arrivo
- ❑ Un grafo G è detto **connesso** se esiste un cammino tra una qualsiasi coppia di nodi
- ❑ Un grafo **orientato** è un grafo in cui i rami hanno un verso di percorrenza
- ❑ Un grafo **pesato** è un grafo in cui a ciascun ramo (i,j) è associato un numero w_{ij} (**peso o metrica** del ramo)

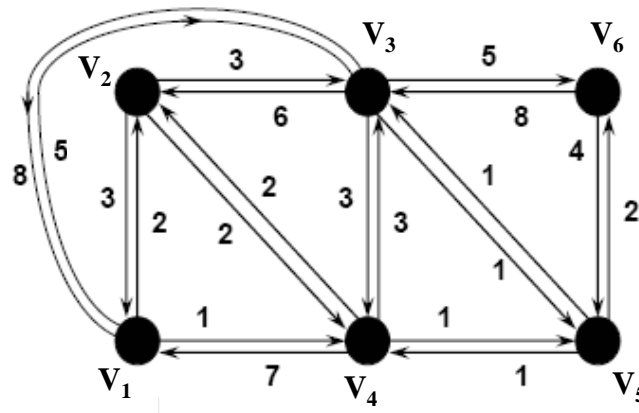
Teoria dei grafi

- La matrice delle adiacenze di un grafo orientato pesato è definita:

$$A = [a_{ij}] \quad a_{ij} = \begin{cases} w_{ij} & \text{if } (i, j) \in E \\ 0 & \text{otherwise} \end{cases}$$

- La lunghezza di un cammino in un grafo pesato è data dalla somma dei pesi associati ai rami del cammino

- Esempio:



	V ₁	V ₂	V ₃	V ₄	V ₅	V ₆
V ₁	0	2	5	1	0	0
V ₂	3	0	3	2	0	0
V ₃	8	6	0	3	1	5
V ₄	7	2	3	0	1	0
V ₅	0	0	1	1	0	2
V ₆	0	0	8	0	4	0

Grafo orientato e pesato

Teoria dei grafi

- Una qualsiasi rete a pacchetto può essere modellata come un grafo orientato e pesato:
 - i nodi sono i router
 - i rami sono le sottoreti
- La funzione di instradamento di un pacchetto in una rete equivale alla ricerca di un cammino nel grafo associato della rete
 - Ricerca del cammino a minima distanza: grafo non pesato
 - Ricerca del cammino a minima lunghezza: grafo pesato

Alberi

- ❑ Un grafo T è detto albero (tree) se:
 - tra ogni coppia di nodi i e j esiste un solo cammino semplice
 - detto N è il numero di nodi, il numero di rami del grafo è $N-1$ ed esso risulta connesso senza cicli
- ❑ Ogni nodo di un albero può essere denominato radice
- ❑ Un albero può essere sempre raffigurato disponendo i nodi in livelli successivi a partire dalla radice
- ❑ In una rappresentazione a livelli di un albero
 - ogni nodo, tranne la radice, ha un solo nodo padre
 - ogni nodo ha zero o più nodi figli
 - se un nodo non ha figli è detto foglia dell'albero

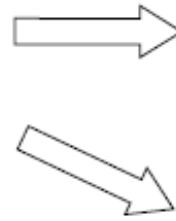
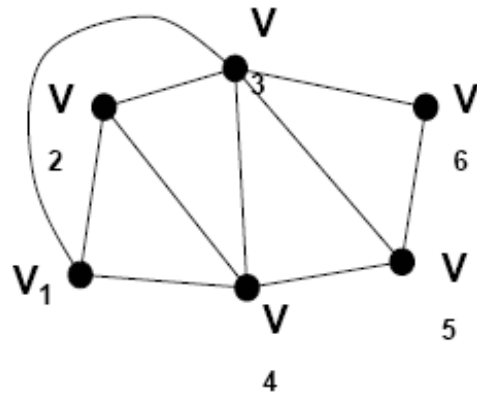
Spanning tree

- Un sottografo di un grafo $G(V,E)$ è un grafo ottenuto dal grafo G :
 - scegliendo un sotto insieme di rami e di nodi appartenenti a G
 - per ogni ramo scelto devono essere compresi, nel sottoinsieme dei nodi, i nodi in cui il ramo è incidente
- Un sottografo T di un grafo G è chiamato **spanning tree** di G se:
 - T è un albero
 - T include tutti i nodi di G
- Uno spanning tree di un grafo G [detto anche albero ricoprente] è un sottografo connesso in cui sono stati rimossi tutti i possibili cicli tra due nodi
- Lo Spanning Tree di un grafo non è, in generale, unico

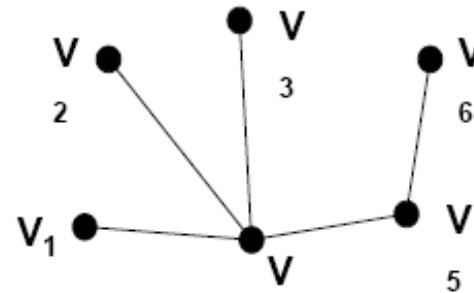
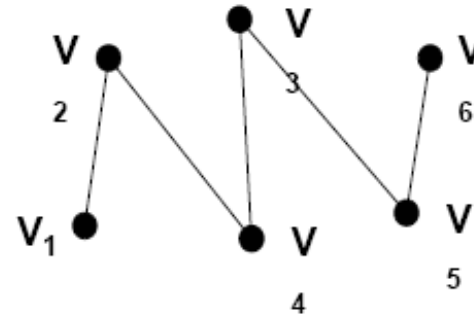
Spanning tree

□ Esempio:

Grafo G



Spanning Tree 1



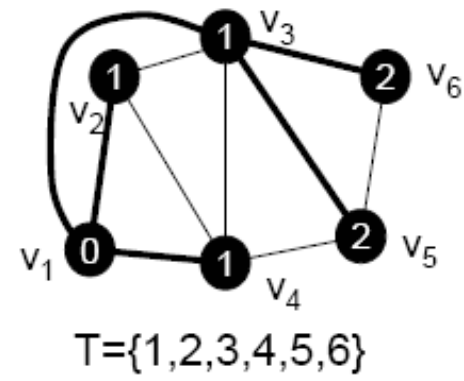
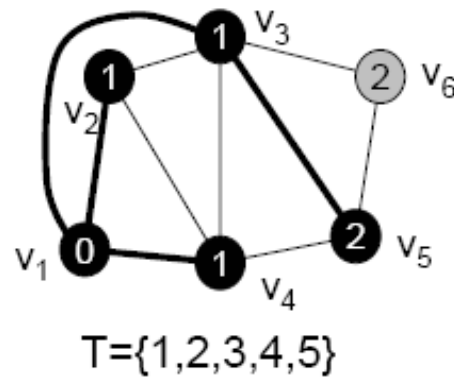
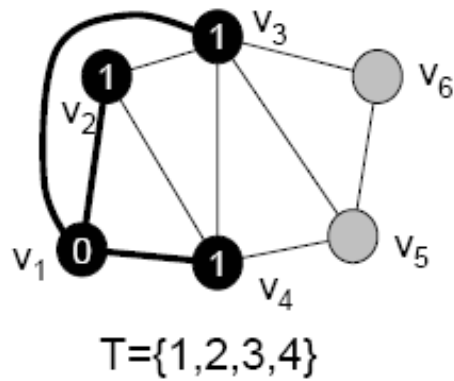
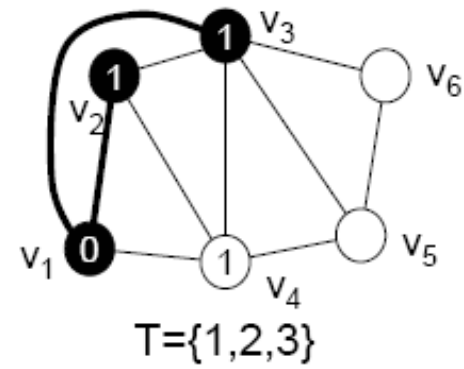
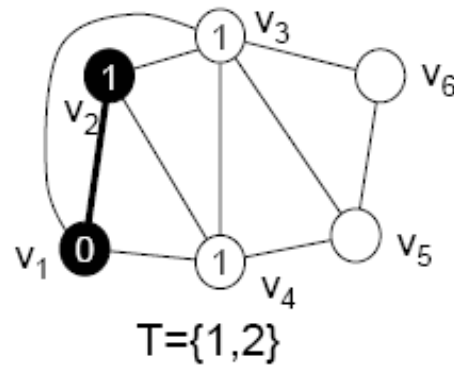
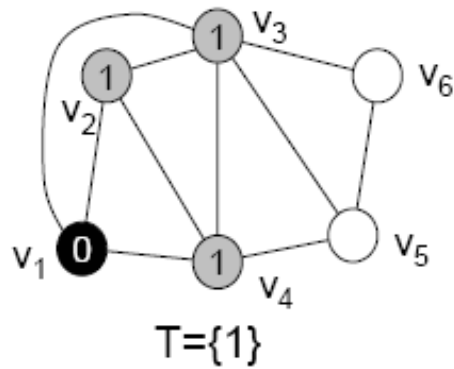
Spanning Tree 2

Spanning tree

- Ricerca di un spanning tree: **Breadth-First Search (BFS) algorithm**
 - Idea base: classificare i nodi in livelli a partire da un nodo radice
 - Algoritmo:
 - si individua un nodo radice (nodo x)
 - si individuano tutti i nodi adiacenti a x (nodi di livello 1)
 - per ogni nodo appartenente al livello 1 si individuano i nodi adiacenti non precedentemente raggiunti (nodi di livello 2)
 - si itera sino a che tutti i nodi del grafo sono raggiunti

Breadth-First Search algorithm

□ Esempio:



Breadth-First Search algorithm

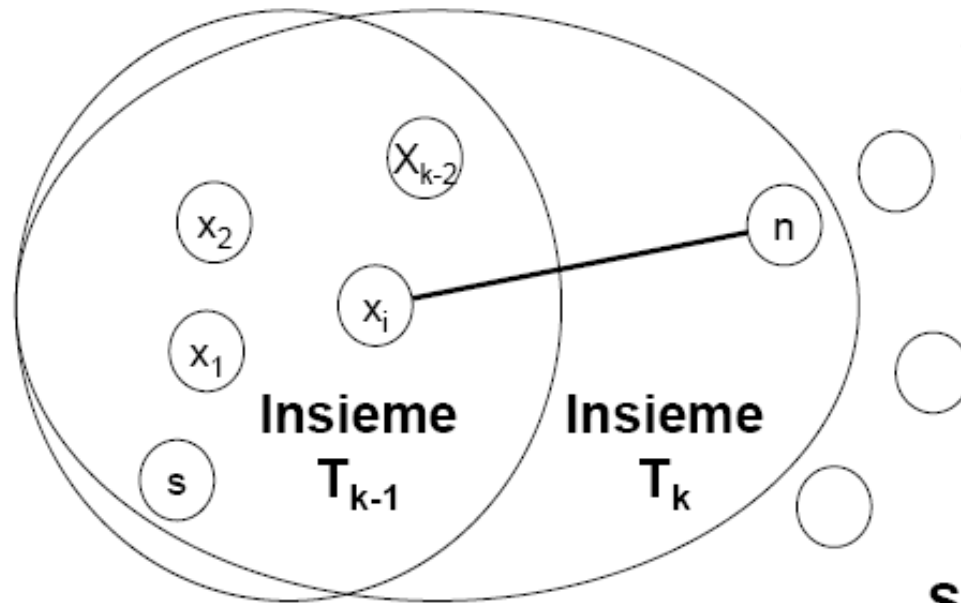
- L'algoritmo BFS, oltre a produrre uno spanning tree, **individua i cammini a distanza minima** tra il nodo radice e tutti gli altri nodi del grafo
 - Infatti, tutti i nodi di livello 1 hanno distanza 1 dalla radice; quelli di livello 2 hanno distanza 2 (e non possono averla inferiore altrimenti sarebbero di livello 1)
- Un cammino a distanza minima (shortest path length) $\delta(s,v)$ tra un nodo s ed un nodo v è dato dal cammino con il minimo numero di rami tra i due nodi
- La complessità computazionale dell'algoritmo BFS è proporzionale a $|V|$ ed $|E|$ [$O(|V| \cdot |E|)$]
 - nel caso peggiore di grafo completo [un ramo fra ogni coppia di nodi] si avrebbe:
 - o posto $|V|=n \Rightarrow |E| = n(n-1) \Rightarrow$ la complessità è $O(|V|^3)$

Algoritmo di Dijkstra

- Individua il cammino a **lunghezza minima** tra un nodo s [sorgente] e tutti gli altri nodi di un grafo G procedendo in modo da aumentare progressivamente la distanza
- L'algoritmo procede per step successivi:
 - al passo k -mo sono individuati k nodi raggiungibili dal nodo sorgente tramite i cammini a costo più basso; tali k nodi formano l'insieme T_k
 - al passo $k+1$ -mo si individua il nodo n che è caratterizzato dal cammino dal costo più basso dal nodo s che transita esclusivamente nei nodi dell'insieme T_k
 - viene formato l'insieme T_{k+1} aggiungendo il nodo n all'insieme T_k
 - l'algoritmo termina quando sono stati esplorati tutti i nodi

Algoritmo di Dijkstra

Al passo k viene aggiunto all'insieme T_{k-1} il nodo n caratterizzato dal cammino a costo minimo con il nodo sorgente s che transita esclusivamente in nodi dell'insieme T_{k-1}



Situazione al passo k

Algoritmo di Dijkstra

□ Notazioni:

- V : insieme dei nodi del grafo
 - $N=|V|$
 - s : nodo sorgente
 - T_k : insieme dei nodi raggiunti dall'algoritmo al passo k
 - $w(i,j)$: peso (costo) del ramo (i,j)
 - $w(i,i) = 0$
 - $w(i,j) \geq 0$ se i vertici i e j sono connessi direttamente
 - $w(i,j) = \infty$ se i vertici i e j non sono connessi direttamente
- $L_k(n)$: costo del cammino a costo minimo, individuato dall'algoritmo fino al passo k , tra il nodo s ed un generico nodo n
- è calcolato anche per i nodi non appartenenti all'insieme T_k

Algoritmo di Dijkstra

□ Inizializzazione ($k=1$)

- $T_1 = \{s\}$
- $L_1(n) = w(s,n)$ per $n \neq s$

□ Aggiunta di un nodo (passo $1 \leq k \leq N$)

- trovare $x \notin T_{k-1}$ tale che:

$$L_{k-1}(x) = \min_{j \notin T_{k-1}} L_{k-1}(j)$$

- aggiungere all'insieme T_{k-1} il nodo x ed il ramo incidente a x
- Oss: per come è costruito l'algoritmo, il nodo x aggiunto è connesso o direttamente ad s oppure ad un nodo $\in T_{k-1}$; solo in questi 2 casi, infatti, si ha $L_{k-1}(x) \neq \infty$

□ Aggiornamento dei cammini minimi correnti:

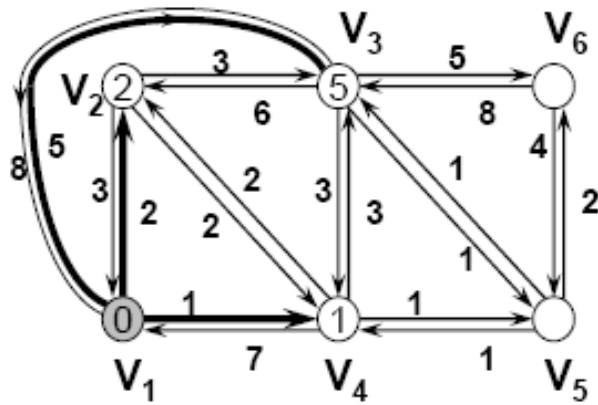
- $L_k(n) = \min [L_{k-1}(n), L_{k-1}(x) + w(x,n)]$ per tutti i valori di $n \notin T_k$

Algoritmo di Dijkstra

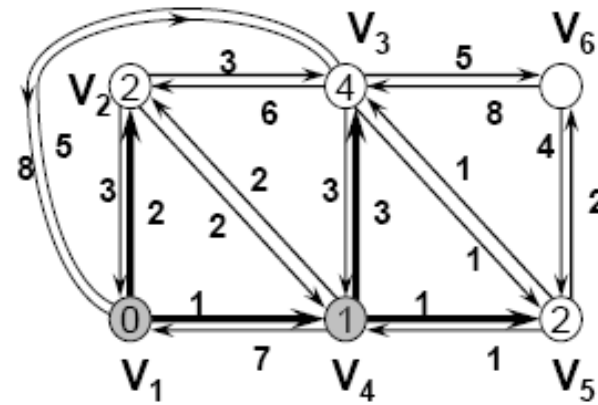
- La terminazione dell'algoritmo determina:
 - l'insieme T_N che risulta essere uno spanning tree del grafo di partenza contenente i cammini a costo minimo tra il nodo sorgente e tutti gli altri nodi del grafo
 - $L_N(n) \forall n \in V$ indica il costo del cammino a costo minimo tra il nodo s ed il generico nodo n
- Si noti che:
 - al passo k -mo viene aggiunto all'insieme T_{k-1} il k -mo nodo ed è individuato il cammino a costo minimo tra tale nodo ed il nodo sorgente
 - questo cammino transita esclusivamente attraverso i nodi sinora compresi nell'insieme T_{k-1}
- La complessità dell'algoritmo è $o(|V|^2)$; implementazioni alternative più efficienti possono arrivare a $o(|V| \cdot \log|V|)$.

Algoritmo di Dijkstra

□ Esempio 1/2:

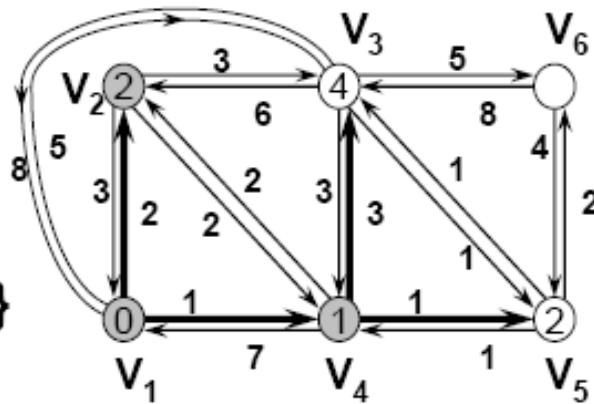


$T_1 = \{1\}$



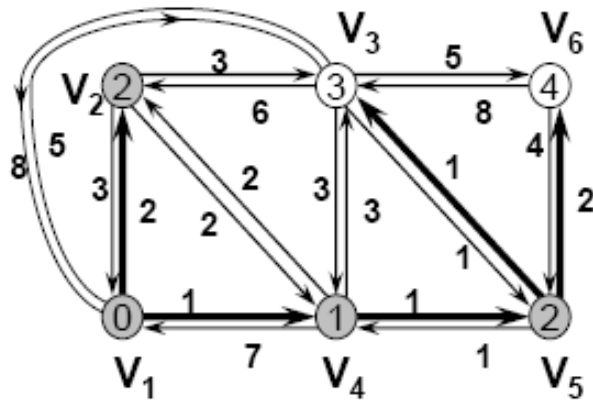
$T_2 = \{1,4\}$

$T_3 = \{1,2,4\}$

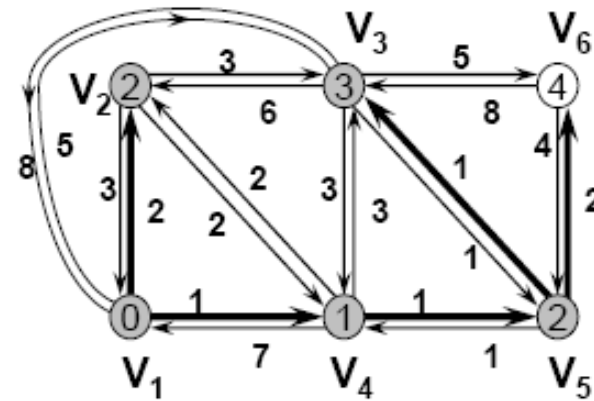


Algoritmo di Dijkstra

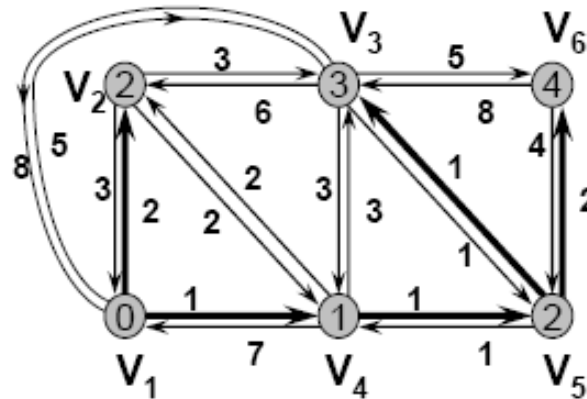
□ Esempio 2/2:



$T_4 = \{1,2,4,5\}$



$T_5 = \{1,2,3,4,5\}$



$T_6 = \{1,2,3,4,5,6\}$

Algoritmo di Bellman-Ford

- ❑ Individua il cammino a lunghezza minima tra un nodo s [sorgente] e tutti gli altri nodi di un grafo G
- ❑ L'algoritmo procede per step successivi:
 - al primo step si individuano i cammini minimi tra il nodo sorgente s e gli altri nodi con il vincolo che i cammini devono essere composti al massimo da 1 ramo
 - al secondo step si trovano i cammini minimi tra il nodo sorgente e gli altri nodi con il vincolo che i cammini devono essere composti al più da 2 rami
 - si itera il procedimento sino al **valore massimo di rami in un cammino**

Algoritmo di Bellman-Ford

□ Notazioni

- V : insieme dei nodi del grafo
- $N=|V|$
- s : nodo sorgente
- h : massimo numero di rami in un cammino correntemente consentito dall'algoritmo (\equiv step dell'algoritmo)
- $w(i,j)$: peso (costo) del ramo (i,j)
 - $w(i,i) = 0$
 - $w(i,j) \geq 0$ se i vertici i e j sono connessi direttamente
 - $w(i,j) = \infty$ se i vertici i e j non sono connessi direttamente
- $L_h(s,n)$: costo del cammino a costo minimo individuato dall'algoritmo fino al passo h , tra il nodo s ed il nodo n , con il vincolo che il numero di rami sia al più pari ad h

Algoritmo di Bellman-Ford

□ Inizializzazione:

- $L_0(n) = \infty \quad \forall n \neq s$
- $L_h(s) = 0 \quad \forall h$

□ Aggiornamento:

- per ogni valore $h \geq 0$ e per ogni nodo calcolare:

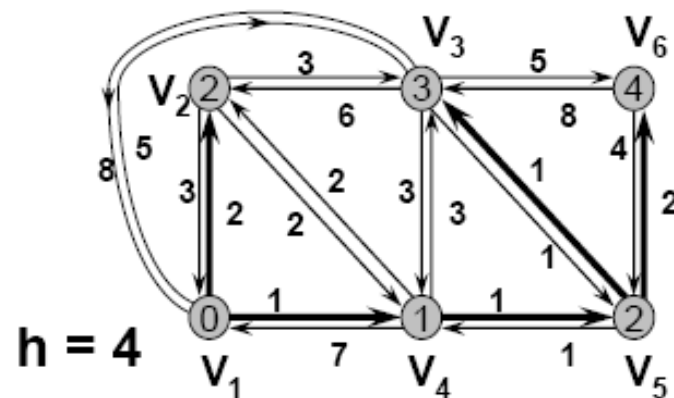
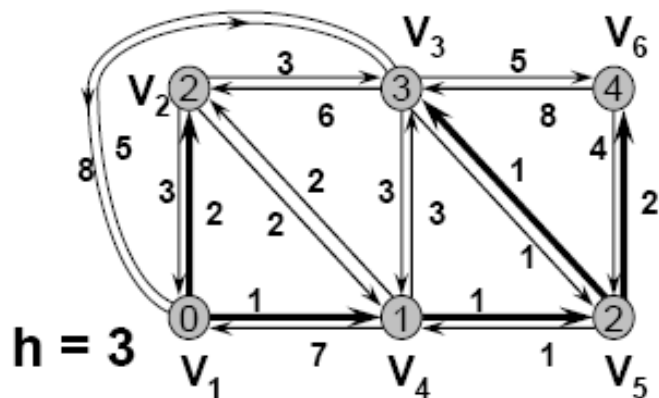
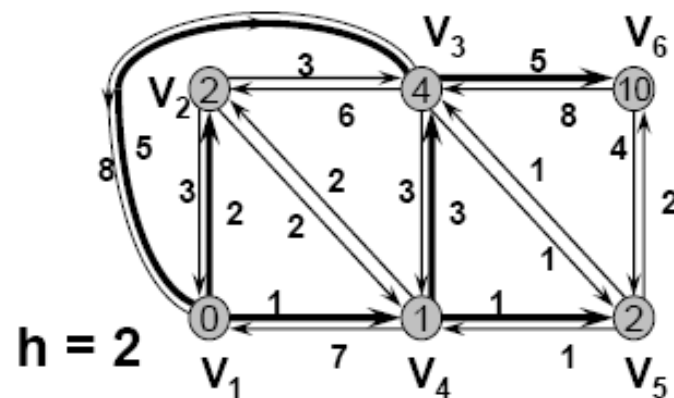
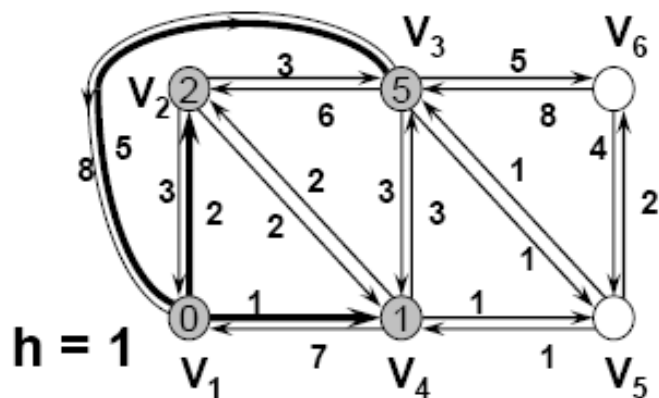
$$L_{h+1}(s, n) = \min_j [L_h(s, j) + w(j, n)]$$

- connettere il nodo n con il nodo predecessore j che raggiunge il minimo ed eliminare le connessioni di n con altri nodi predecessori individuati in precedenti iterazioni

- La complessità dell'algoritmo è $o(|V| \cdot |E|)$, ovvero, nel caso peggiore in cui $|E|=|V|^2$, si ha $o(|V|^3)$

Algoritmo di Bellman-Ford

□ Esempio:



Algoritmo di Bellman-Ford

- La soluzione di Bellman-Ford può essere scritta:

$$L(s,n) = \min_{j \in A_n} [L(s,j) + w(j,n)] \quad A_n: \text{insieme dei nodi adiacenti al nodo } n$$

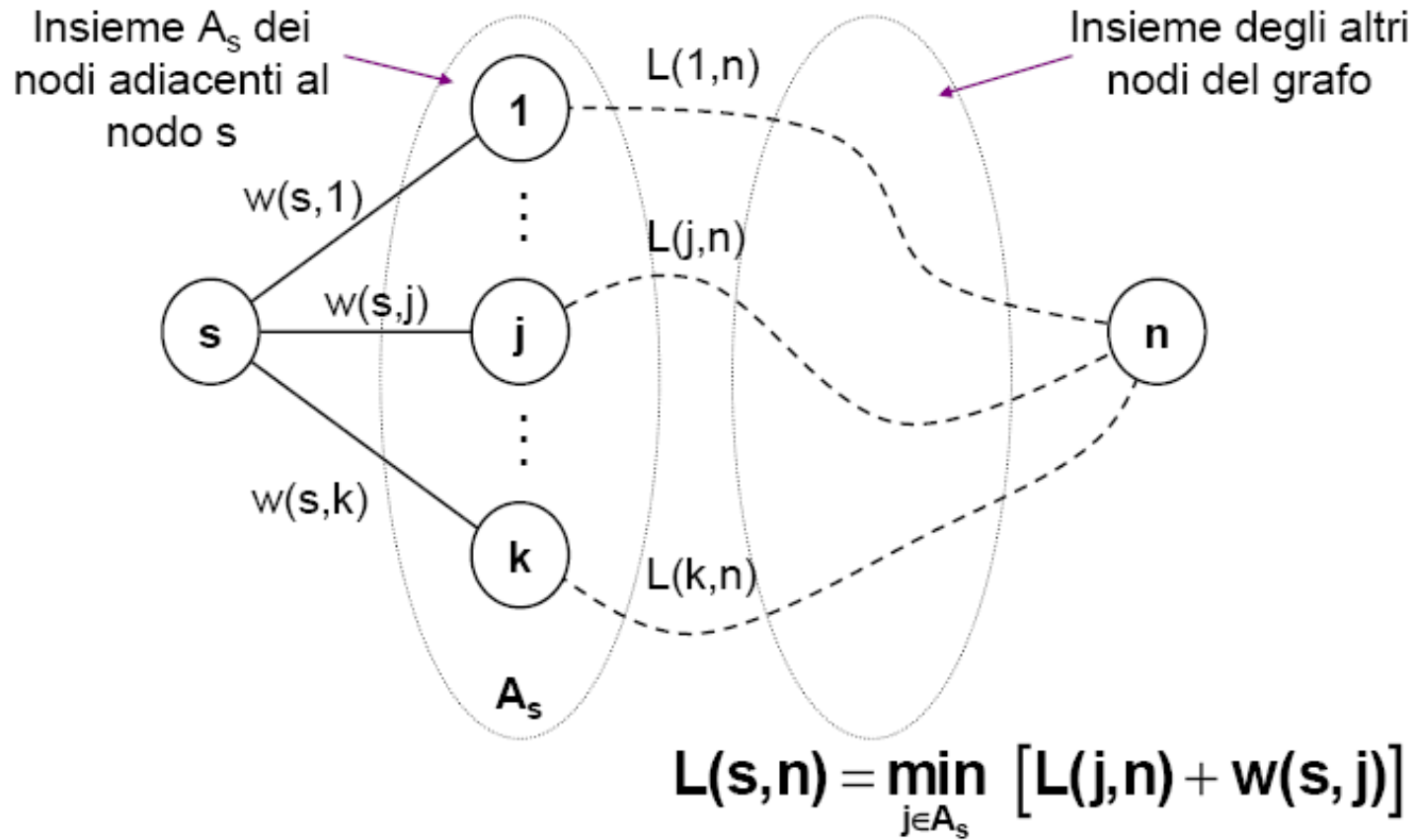
- Il cammino a lunghezza minima tra il nodo s ed il nodo n è dato dalla composizione a costo minimo del cammino a lunghezza minima tra il nodo s ed un nodo j adiacente al nodo n e il ramo incidente ai nodi j e n

- Oppure:

$$L(s,n) = \min_{j \in A_s} [w(s,j) + L(j,n)] \quad A_s: \text{insieme dei nodi adiacenti al nodo } s$$

- **Il cammino a lunghezza minima tra il nodo s ed il nodo n è dato dalla composizione a costo minimo del cammino a lunghezza minima tra il nodo n ed un nodo j adiacente al nodo s e il ramo incidente ai nodi s e j**

Algoritmo di Bellman-Ford



Dijkstra vs. Bellman-Ford

- I due algoritmi convergono alla stessa soluzione
- L'algoritmo di Bellman-Ford, nel caso peggiore, è più complesso; in molti casi pratici i due algoritmi si equivalgono
- L'algoritmo di Dijkstra richiede **che un nodo conosca l'intera topologia della rete ed i pesi (stato) di tutti i rami**
 - necessità di colloquio tra tutti i nodi
- L'algoritmo di Bellman-Ford richiede **la conoscenza dello stato dei rami uscenti da un nodo e le informazioni sui percorsi minimi a partire dai nodi vicini**
 - possibilità di colloquio solo tra nodi adiacenti (implementazione distribuita)

Algoritmo di Bellman-Ford Distribuito

- Ogni nodo s esegue, **in modo asincrono**, il seguente calcolo:

$$L(s,n) = \min_{j \in A_s} [L(j,n) + w(s,j)]$$

- Utilizzando:
 - i valori di $w(s,j)$ ($j \in A_s$) che **conosce direttamente**
 - i valori di $L(j,n)$ ($j \in A_s$) **ricevuti dai nodi adiacenti** [utilizza le ultime stime ricevute]
- A sua volta, il nodo s comunica il risultato ai nodi vicini
- L'algoritmo distribuito converge a soluzione se la dinamica delle variazioni è più lenta della velocità di convergenza dell'algoritmo
 - Eventualità di **“bad news phenomenon”**
 - in casi particolari la convergenza può essere molto lenta
 - il numero di iterazioni può essere molto elevato