



Universitatea “Constatin Brâncuși” din Târgu-Jiu
Facultatea de Inginerie
Departamentul de Automatică, Energie și Mediu

Rețele de calculatoare

Lect. dr. Adrian Runceanu

An universitar 2013-2014

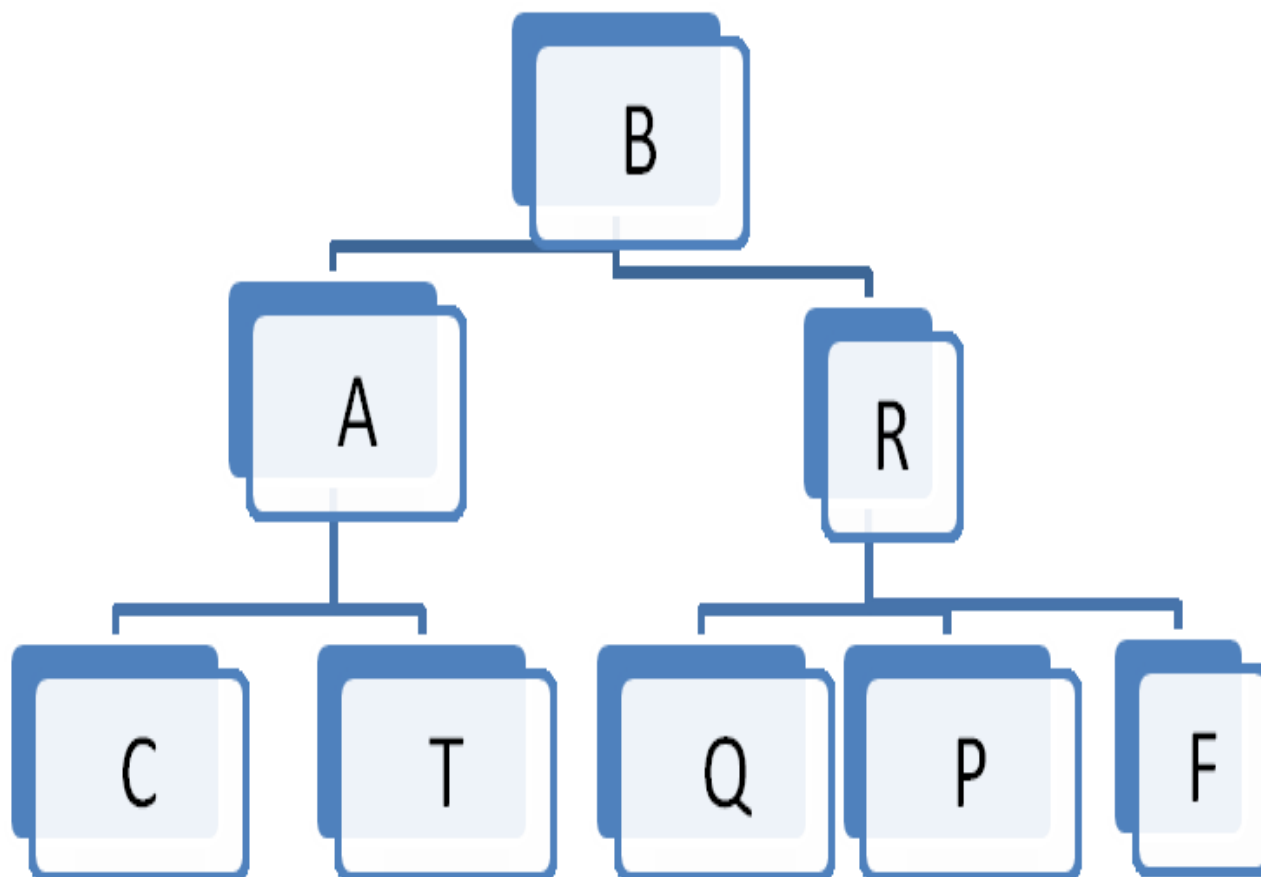
Curs 9

Algoritmi de dirijare

- 1. Algoritmul Dijkstra**
- 2. Algoritmul Belmann-Ford**

Principiul optimalității

- *Principiul optimalității presupune că dacă o rută optimă între nodul A și nodul B trece prin nodul C atunci ruta optimă între B și C este inclusă în ruta A-B.*
- O consecință a principiului optimalității este formarea **arborelui de scufundare** care semnifică *mulțimea rutelor optime către un nod destinație*, reprezentat de rădăcina arborelui.
- Scopul **algoritmilor de dirijare** constă în *descoperirea arborilor de scufundare pentru fiecare rută.*



Arbore de scufundare pentru nodul B

Algoritmi de dirijare

- *Algoritmul de dirijare este acea parte a software-lui nivelului rețea care răspunde de alegerea liniei de ieșire pe care un pachet recepționat trebuie trimis mai departe.*
- În cazul **datagramelor** această decizie trebuie luată din nou pentru fiecare pachet recepționat, deoarece e posibil ca cea mai bună rută să se fi modificat între timp.
- În cazul **circuitelor virtuale**, decizia de dirijare se ia doar la stabilirea unui nou circuit virtual.
- După aceea pachetele vor urma doar calea stabilită anterior.
- Acest ultim caz este numit uneori **sesiune de dirijare** deoarece calea rămâne în funcțiune pentru o întreagă sesiune utilizator (ex. transfer de fișiere, login prin terminal).

Clasificare

Algoritmii pot fi:

1. neadaptivi - *alegerea căii se calculează în avans (of line) și parvine ruterului la inițializarea rețelei*

2. adaptivi - *își modifică deciziile de dirijare pentru a reflecta modificările de topologie și pe cele de trafic*

Algoritmii adaptivi diferă prin:

1. *locul de unde își iau informația:*

- a) local
- b) de la un vecin
- c) de la toate ruterele

2. *momentul în care schimbă rutele:*

- a) când se schimbă încărcarea
- b) când se schimbă topologia
- c) la ΔT secunde

3. *metrica folosită pentru optimizare:*

- a) distanța
- b) timpul de tranzit
- c) numărul de salturi

Algoritmi statici de dirijare

1. Dirijarea pe calea cea mai scurtă (shortest path)

- *Calea cea mai scurtă este dictată de metrica de măsurare a distanței.*
- Aceasta poate fi:
 - ❑ distanța în km
 - ❑ numărul de salturi
 - ❑ rata de transfer
 - ❑ traficul mediu
 - ❑ costul comunicației
 - ❑ lungimea minimă a cozilor de așteptare
 - ❑ întârzieri măsurate

2. Algoritmul de inundare - generează foarte multe pachete.

Pentru a limita numărul acestora se procedează astfel:

1. contor de salturi în antetul fiecărui pachet decrementat la fiecare salt și care face ca pachetul să fie distrus atunci când contorul atinge valoarea 0.

2. ruterul sursă să plaseze un număr de secvență în fiecare pachet pe care îl primește de la calculatorul gazdă asociat.

2. Algoritmul de inundare

- Fiecare ruter necesită menținerea unei liste pentru fiecare ruter sursă, cu numere de secvență inițiate de acel ruter sursă și care au fost deja trimis mai departe.
- Dacă sosește un pachet care deja se află în listă el nu se mai trimite.
- Se poate însoți lista de un contor k care semnifică faptul că toate numerele de secvență până la k au fost deja tratate.
- Inundarea se utilizează în aplicații militare, aplicații cu baze de date distribuite care este necesar să fie actualizate concurent.

3. Dirijarea bazată pe flux

- În condițiile în care traficul mediu de la i la j este cunoscut în avans și într-o aproximare rezonabilă, constant în timp, este posibilă analiza matematică a fluxurilor pentru a optimiza dirijarea.
- Ideea care stă la baza analizei este aceea că pentru o anumită linie dacă se cunosc capacitatea și fluxul mediu, este posibil să se calculeze întârzierea medie a unui pachet pe linia respectivă, folosind teoria cozilor.

- Pe baza întârzierilor medii ale tuturor liniilor se poate calcula imediat, ca medie ponderată după flux, întârzierea medie a unui pachet pentru întreaga subrețea.
- *Problema dirijării se reduce apoi la găsirea algoritmului de dirijare care produce întârzierea medie minimă pentru subrețea.*
- Folosirea acestei tehnici presupune:
 - cunoașterea topologiei subrețelei
 - matricea traficului
 - matricea capacităților liniilor, C_i , măsurată în < Kbps.>

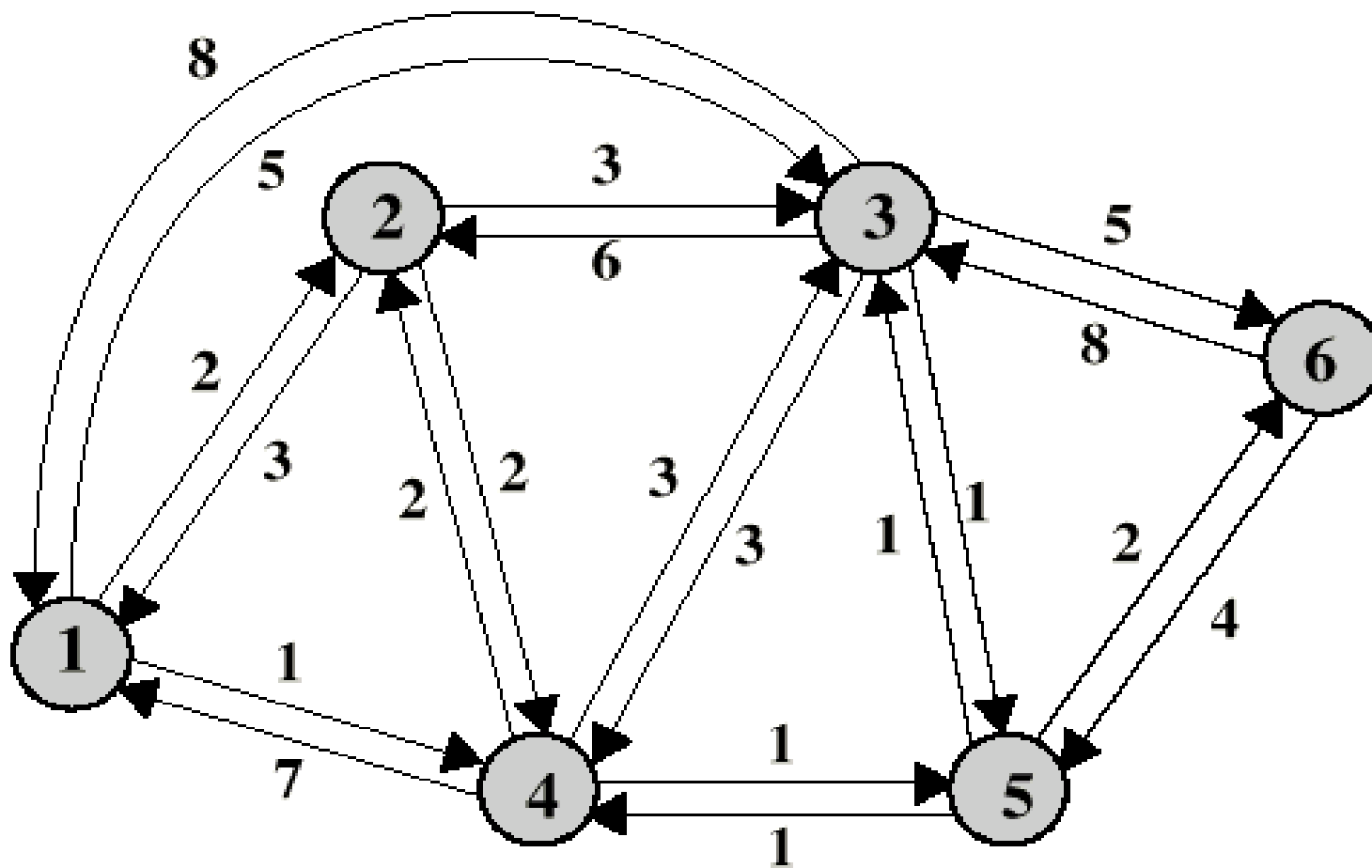
Algoritmi dinamici de dirijare

Cei mai folosiți sunt:

1. Algoritmul de dirijare cu vectori distanță
2. Algoritmul de dirijare bazat pe starea legăturilor

Algoritmul de dirijare cu vectori distanță
(Bellman - Ford - Fulkerson)

Costurile rutelor



Strategii de Rutare

1. Rutare fixă
2. Rutare bazata pe inundare
3. Rutare aleatoare
4. Rutare adaptivă

1. Rutare Fixă

- O singură cale pentru fiecare pereche **sursă-destinație**
- Rutele sunt determinate printr-un *algorithm de cost minim*
- Rutele rămân fixe, până la schimbarea topologiei rețelei

Tabele de Rutare Fixe

CENTRAL ROUTING DIRECTORY

From Node

To Node

	1	2	3	4	5	6
1	—	1	5	2	4	5
2	2	—	5	2	4	5
3	4	3	—	5	3	5
4	4	4	5	—	4	5
5	4	4	5	5	—	5
6	4	4	5	5	6	—

Node 1 Directory

Destination	Next Node
2	2
3	4
4	4
5	4
6	4

Node 2 Directory

Destination	Next Node
1	1
3	3
4	4
5	4
6	4

Node 3 Directory

Destination	Next Node
1	5
2	5
4	5
5	5
6	5

Node 4 Directory

Destination	Next Node
1	2
2	2
3	5
5	5
6	5

Node 5 Directory

Destination	Next Node
1	4
2	4
3	3
4	4
6	6

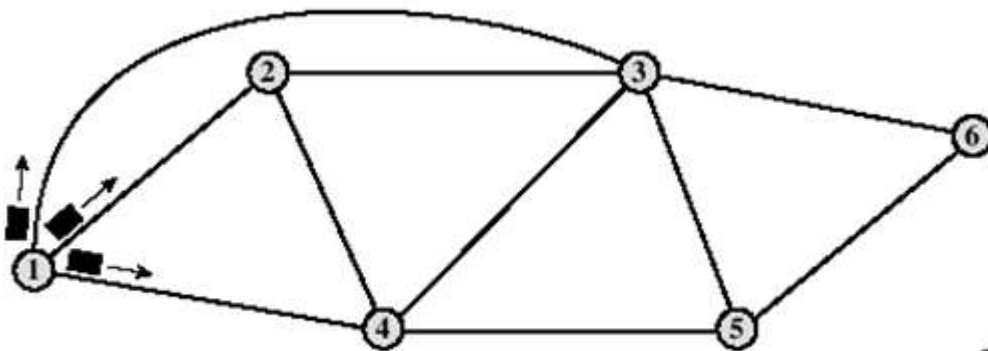
Node 6 Directory

Destination	Next Node
1	5
2	5
3	5
4	5
5	5

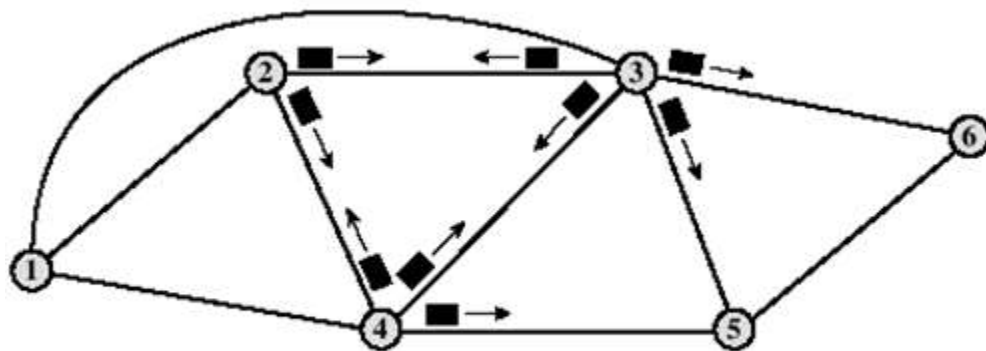
2. Rutare bazata pe inundare

- Nu sunt necesare informatii despre rețea
- *Pachetul este trimis la toți vecinii*
- *Sau la toți în afară de unde a venit*
- Un număr de copii ajung după un timp la destinație
- Fiecare pachet are un număr unic, duplicatele se ignoră
- Nodurile pot reține identitatea pachetelor pentru a nu le ruta din nou
- Se poate defini un timp de viață a pachetelor

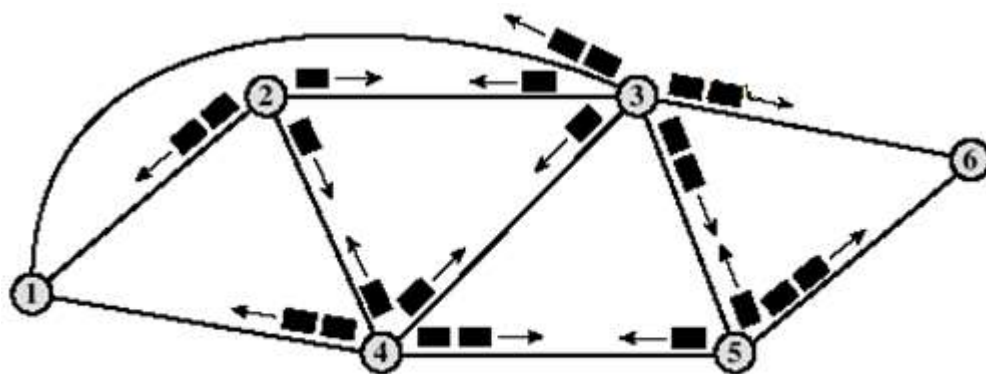
Inundare Exemplu



(a) Primul salt



(b) Al doilea salt



(c) Al treilea salt

Proprietăți ale Inundării

- TOATE rutele posibile sunt încercate
 - foarte robust
- Cel puțin un pachet va ajunge pe calea de cost minim
 - Se poate folosi pt. stabilirea unui circuit virtual
- Toate nodurile sunt atinse
 - Util pt. distribuirea de informații (ex. rutare)

3. Rutare Aleatoare

- *Nodul selectează o cale de ieșire pt. transmiterea unui pachet primit*
- Selecția poate fi aleatoare sau round-robin (fiecare nod este analizat)
- Se pot utiliza și probabilități
- Nu sunt necesare informații despre rețea
- Ruta nu este în general optimă
- Trafic inutil mai mic ca la inundare

4. Rutare Adaptivă - caracteristici

- **Rutarea adaptivă** este cel mai des utilizată
- Decizia de rutare se adaptează condițiilor din rețea:
 - Defecte de linie sau noduri
 - Congestie
- Necesită informații despre rețea
- Decizia este mai complexă
- Compromis între calitatea rețelei și overhead (supraîncărcare)
- Reacție prea rapidă produce oscilații
- Prea încet pentru a fi relevant

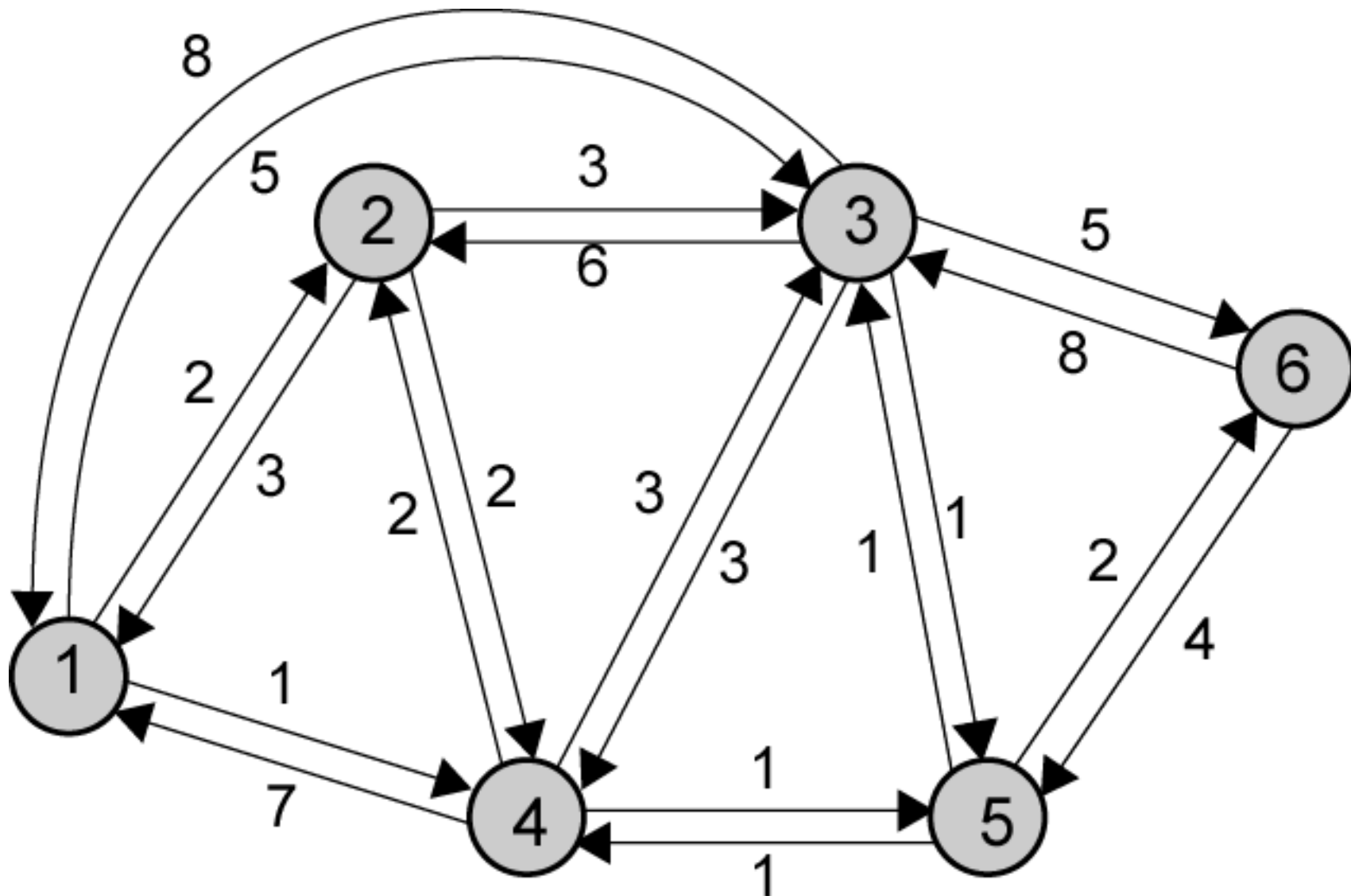
Rutare Adaptivă – Avantaje

- Creșterea performanței
- Ajută la controlul congestiei
- Sistem Complex
 - Poate să nu ajungă la beneficiile teoretice

Criterii de selectie a rutelor

- Minimum hop count (numărul de salturi)
- Minimum cost

Retea



Algoritm de cost minim

- Decizia de rutare este data de:
 - Numar de hopuri(salturi)
 - Cost link invers proportional cu capacitatea
- Retea cu link-uri bidirectionale
- Cost asociat in ambele directii
- Costul caii de rutare = suma costurilor pe link
- Se cauta *calea de cost minim pentru toate perechile sursa si destinatie*
- Costurile pot fi diferite pe cele doua sensuri

Algoritm Dijkstra - definiții

- *Caută calea cea mai scurtă de la un nod sursă la toate nodurile prin dezvoltarea de căi de lungime din ce în ce mai mare*
- N = setul de noduri în rețea
- s = nod sursă
- T = setul de noduri încorporate în algoritm
- $w(i, j)$ = costul legăturii de la nod i la nod j
 - $w(i, i) = 0$
 - $w(i, j) = \infty$ dacă nu sunt direct conectate
 - $w(i, j) \geq 0$ dacă sunt direct conectate
- $L(n)$ = costul căii cunoscute cea mai ieftină de la nod s la nodul n
- la terminare, $L(n)$ este costul căii celei mai ieftine de la s la n

Algoritm Dijkstra - metodă

■ Pas 1 [Inițializare]

- $T = \{s\}$ Nod sursă
- $L(n) = w(s, n)$ pentru $n \neq s$
- Initial doar costuri link

■ Pas 2 [Următorul Nod]

- Caută nod vecin necuprins în T cu cost minim față de s
- Incorporare nod în T

■ Pas 3 [Actualizare cale cea mai ieftină]

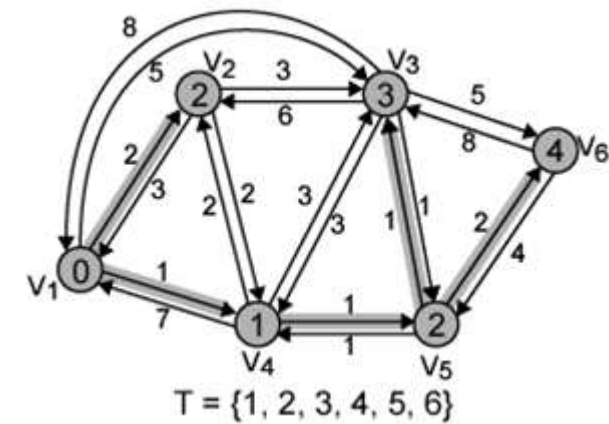
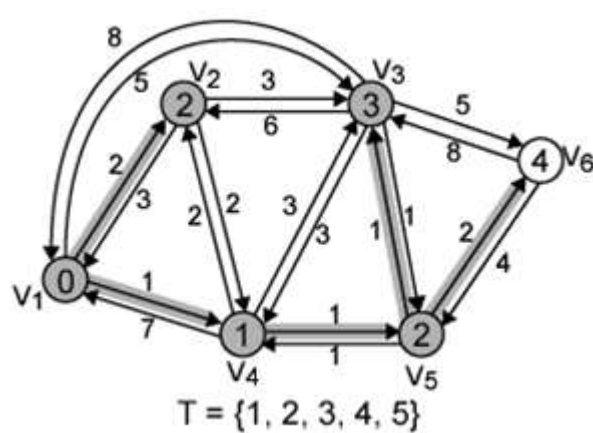
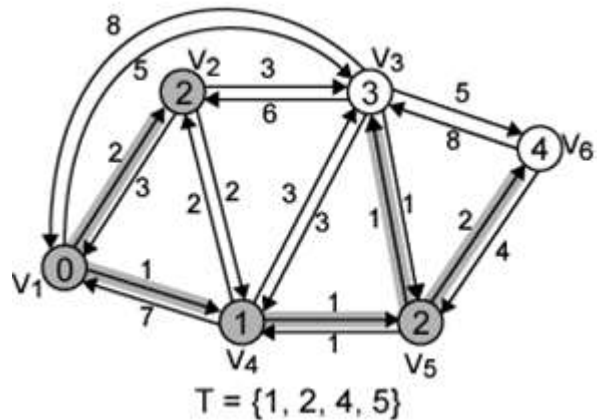
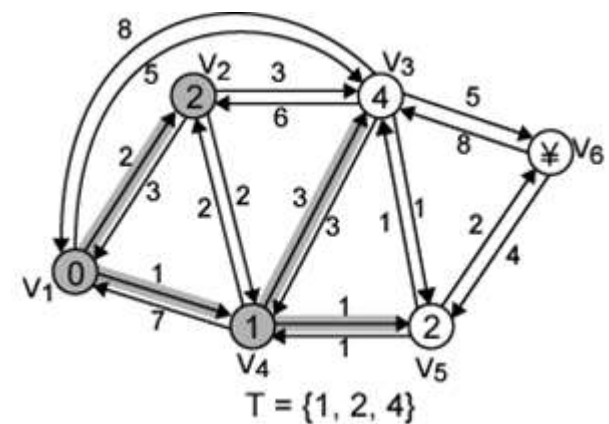
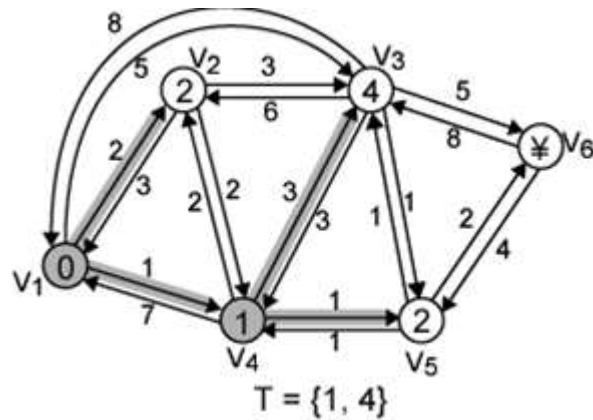
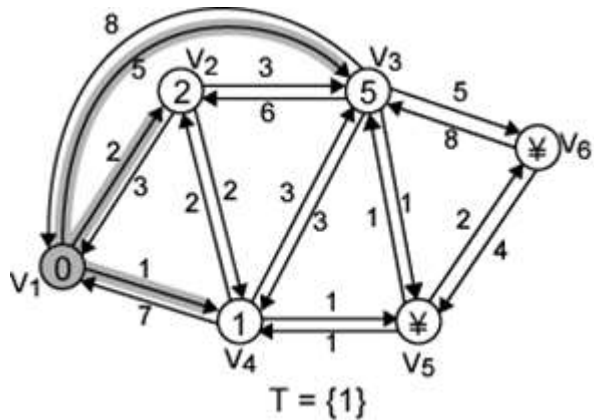
- $L(n) = \min[L(n), L(x) + w(x, n)]$ pentru toți $n \notin T$
- Dacă al doilea termen este minimul, actualizare cale prin concatenare

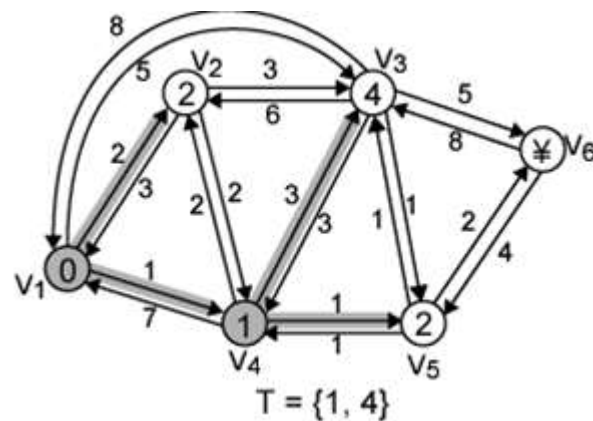
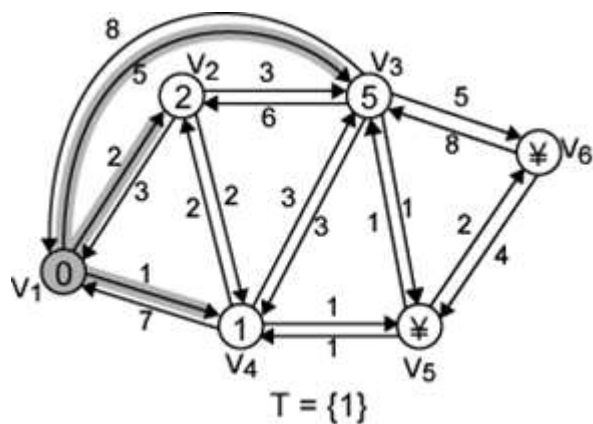
■ [Terminare] Algoritmul se termină la incorporarea tuturor nodurilor

Algoritm Dijkstra - observații

- La terminare, valoarea $L(x)$ asociată fiecărui nod x este costul (lungimea) căii de cost minim de la s la x .
- T definește calea cea mai ieftină de la s la fiecare alt nod
- O iterație a pașilor 2 și 3 adaugă un nou nod în T

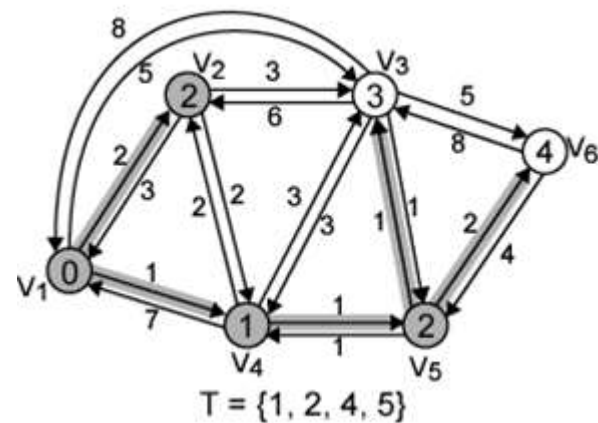
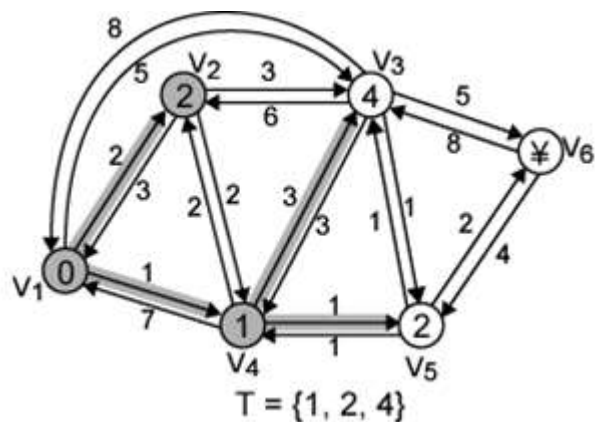
Algoritm Dijkstra - exemplu





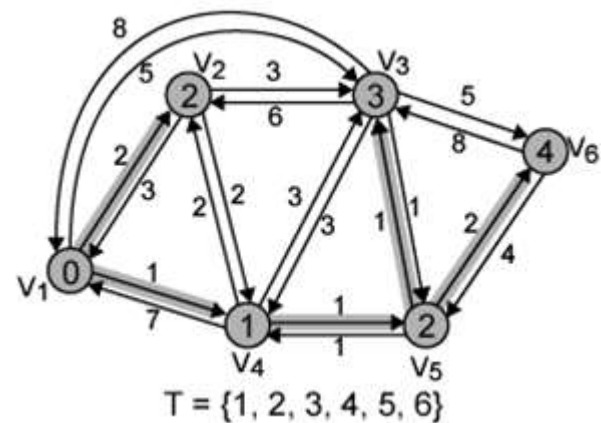
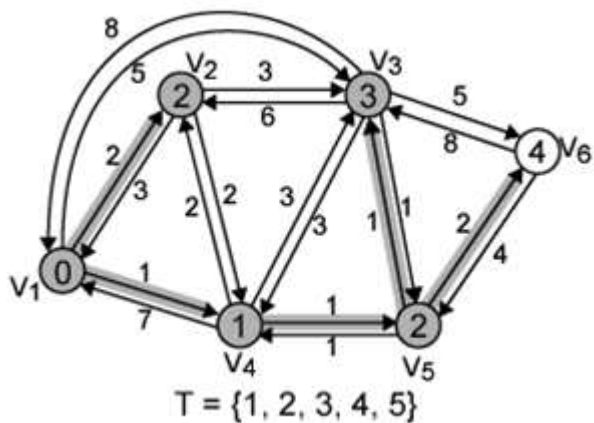
Iteratii	T	L(2)	Cale	L(3)	Cale	L(4)	Cale	L(5)	Cale	L(6)	Cale
1	{1}	2	1-2	5	1-3	1	1-4	∞	-	∞	-

Iteratii	T	L(2)	Cale	L(3)	Cale	L(4)	Cale	L(5)	Cale	L(6)	Cale
2	{1,4}	2	1-2	4	1-4-3	1	1-4	2	1-4-5	∞	-



Iteratii	T	L(2)	Cale	L(3)	Cale	L(4)	Cale	L(5)	Cale	L(6)	Cale
3	{1, 2, 4}	2	1-2	4	1-4-3	1	1-4	2	1-4-5	∞	-

Iteratii	T	L(2)	Cale	L(3)	Cale	L(4)	Cale	L(5)	Cale	L(6)	Cale
4	{1, 2, 4, 5}	2	1-2	3	1-4-5-3	1	1-4	2	1-4-5	4	1-4-5-6



Iteratii	T	L(2)	Cale	L(3)	Cale	L(4)	Cale	L(5)	Cale	L(6)	Cale
5	$\{1, 2, 3, 4, 5\}$	2	1-2	3	1-4-5-3	1	1-4	2	1-4-5	4	1-4-5-6

Iteratii	T	L(2)	Cale	L(3)	Cale	L(4)	Cale	L(5)	Cale	L(6)	Cale
6	$\{1, 2, 3, 4, 5, 6\}$	2	1-2	3	1-4-5-3	1	1-4	2	1-4-5	4	1-4-5-6

Rezultate la Exemplu

Algoritm Dijkstra

Iteratii	T	L(2)	Cale	L(3)	Cale	L(4)	Cale	L(5)	Cale	L(6)	Cale
1	{1}	2	1-2	5	1-3	1	1-4	∞	-	∞	-
2	{1,4}	2	1-2	4	1-4-3	1	1-4	2	1-4-5	∞	-
3	{1, 2, 4}	2	1-2	4	1-4-3	1	1-4	2	1-4-5	∞	-
4	{1, 2, 4, 5}	2	1-2	3	1-4-5-3	1	1-4	2	1-4-5	4	1-4-5-6
5	{1, 2, 3, 4, 5}	2	1-2	3	1-4-5-3	1	1-4	2	1-4-5	4	1-4-5-6
6	{1, 2, 3, 4, 5, 6}	2	1-2	3	1-4-5-3	1	1-4	2	1-4-5	4	1-4-5-6

Algoritm Bellman-Ford. Definiții

- Caută calea cea mai ieftină formată din cel mult un link
- Caută calea cea mai ieftină formată din cel mult două link-uri
- ș.a.m.d.
- s = nod sursă
- $w(i, j)$ = cost link de la nod i la nod j
 - $w(i, i) = 0$
 - $w(i, j) = \infty$ dacă cele două noduri nu sunt conectate direct
 - $w(i, j) \geq 0$ dacă cele două noduri sunt conectate direct
- h = număr maxim link-uri în cale în pasul curent al algoritmului
- $L_h(n)$ = costul căii cea mai ieftine de la s la n cu mai mult de h link-uri

Algoritm Bellman-Ford. Metodă

■ Pas 1 [Inițializare]

- $L_0(n) = \infty$, pentru toți $n \neq s$
- $L_h(s) = 0$, pentru toți h

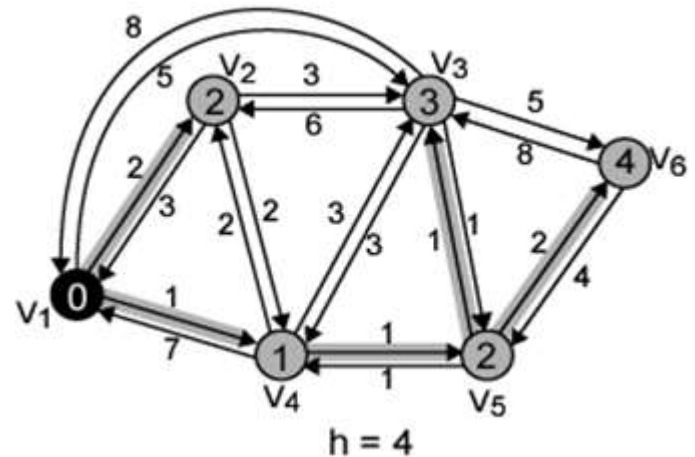
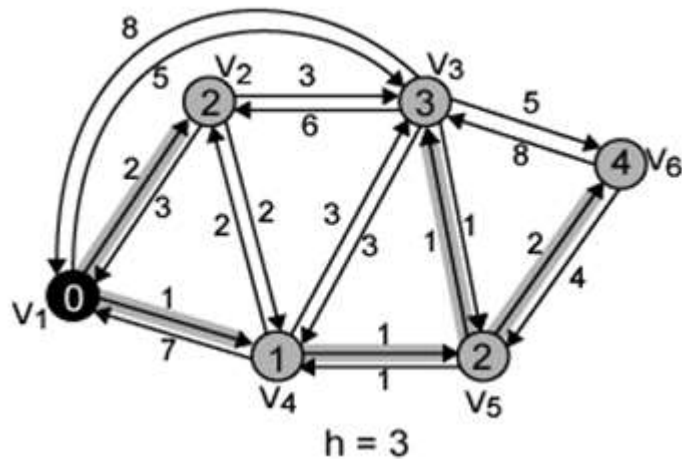
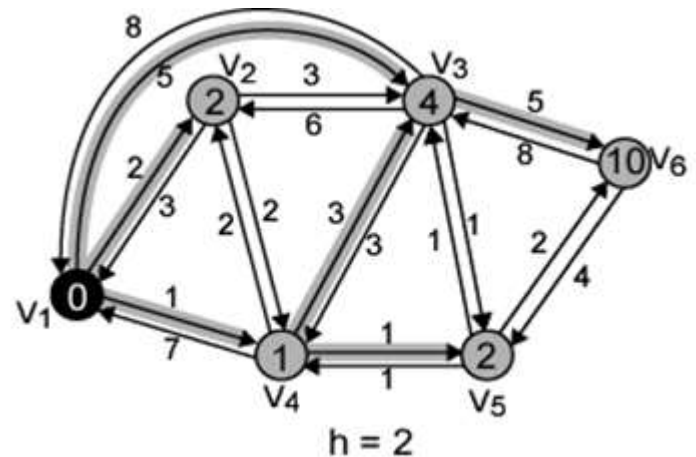
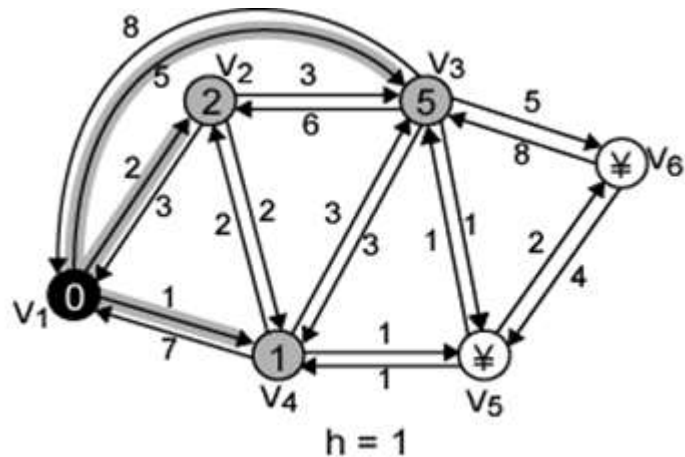
■ Pas 2 [Actualizare]

- Pentru fiecare $h \geq 0$ succesiv
- Pentru fiecare nod $n \neq s$, calculează
 - $L_{h+1}(n) = \min_j [L_h(j) + w(j, n)]$
- Conectează n cu nod predecesor j care atinge minimum
- *Elimină orice conexiune a unui nod n cu un nod predecesor diferit format în iterație anterioară*
- Calea de la s la n se termină cu link de la j la n

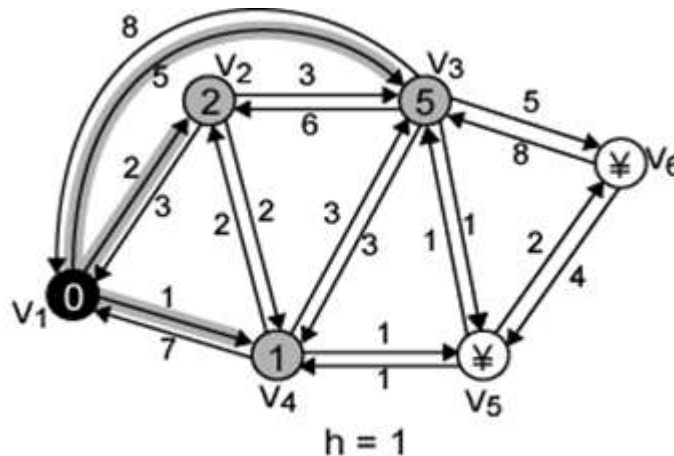
Observații Algoritm Bellman-Ford

- *La fiecare iterație a pasului 2 pentru $h=K$ și fiecare nod destinație n , algoritmul compară căile de la s la n de lungime $K=1$ cu calea de la iterația anterioară*
- În funcție de cost se menține calea anterioară sau se actualizează

Exemplu Algoritm Bellman-Ford

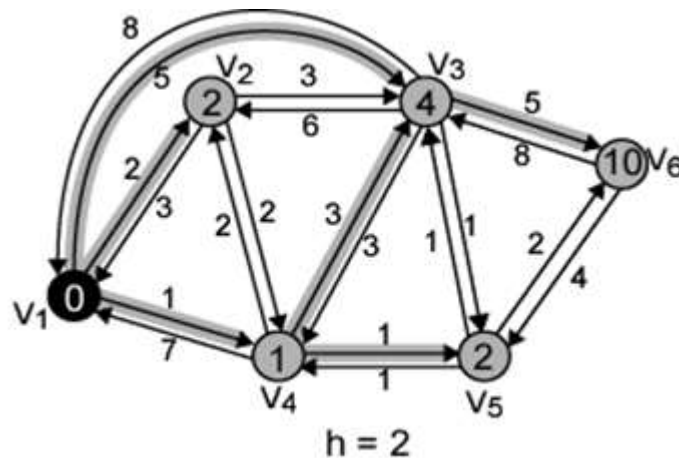


Exemplu Algoritm Bellman-Ford



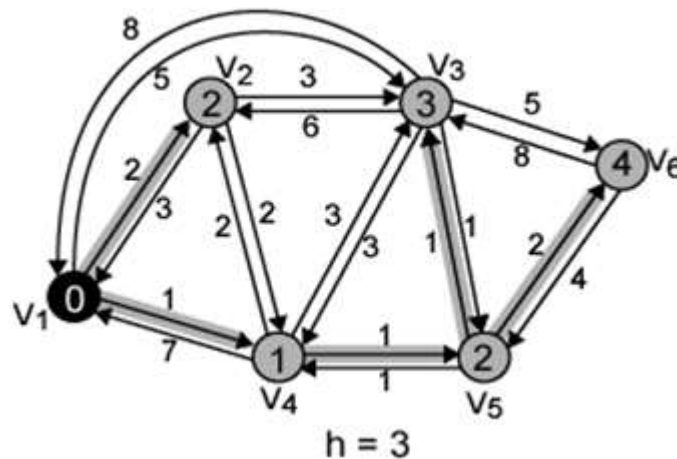
h	$L_h(2)$	Cale	$L_h(3)$	Cale	$L_h(4)$	Cale	$L_h(5)$	Cale	$L_h(6)$	Cale
0	∞	-	∞	-	∞	-	∞	-	∞	-

Exemplu Algoritm Bellman-Ford



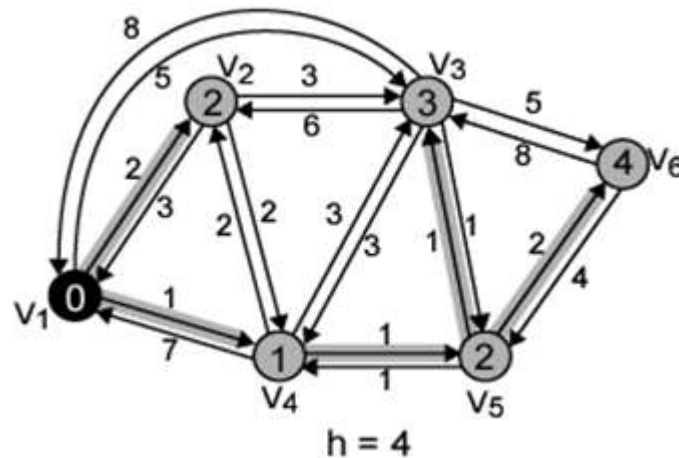
h	$L_h(2)$	Cale	$L_h(3)$	Cale	$L_h(4)$	Cale	$L_h(5)$	Cale	$L_h(6)$	Cale
1	2	1-2	5	1-3	1	1-4	∞	-	∞	-

Exemplu Algoritm Bellman-Ford



h	$L_h(2)$	Cale	$L_h(3)$	Cale	$L_h(4)$	Cale	$L_h(5)$	Cale	$L_h(6)$	Cale
2	2	1-2	4	1-4-3	1	1-4	2	1-4-5	10	1-3-6

Exemplu Algoritm Bellman-Ford



h	$L_h(2)$	Cale	$L_h(3)$	Cale	$L_h(4)$	Cale	$L_h(5)$	Cale	$L_h(6)$	Cale
3	2	1-2	3	1-4-5-3	1	1-4	2	1-4-5	4	1-4-5-6

Rezultate Exemplu Bellman-Ford

h	$L_h(2)$	Cale	$L_h(3)$	Cale	$L_h(4)$	Cale	$L_h(5)$	Cale	$L_h(6)$	Cale
0	∞	-	∞	-	∞	-	∞	-	∞	-
1	2	1-2	5	1-3	1	1-4	∞	-	∞	-
2	2	1-2	4	1-4-3	1	1-4	2	1-4-5	10	1-3-6
3	2	1-2	3	1-4-5-3	1	1-4	2	1-4-5	4	1-4-5-6
4	2	1-2	3	1-4-5-3	1	1-4	2	1-4-5	4	1-4-5-6

Comparație Bellman-Ford cu Dijkstra

- Rezultatele celor doi algoritmi coincid
- Informații colectate:
 - Bellman-Ford
 - Calculul în nodul n necesită cunoașterea costului link-ului la toți vecinii plus totalul costului de la fiecare vecin la s
 - Fiecare nod poate menține un set de costuri și căi pentru fiecare alt nod
 - Face schimb de informații cu vecinii direcți
 - Poate actualiza costurile și căile pe baza informațiilor de la vecini și costurile linkurilor locale
 - Dijkstra
 - Fiecare nod are nevoie de topologia completă
 - Trebuie să cunoască costul tuturor linkurilor din rețea
 - Schimbă informații cu toate celelalte noduri

Întrebări?