Università Roma Tre – Dipartimento di Matematica e Fisica
IN440 – Ottimizzazione Combinatoria
Prof. Marco Liverani

Cammini di costo minimo

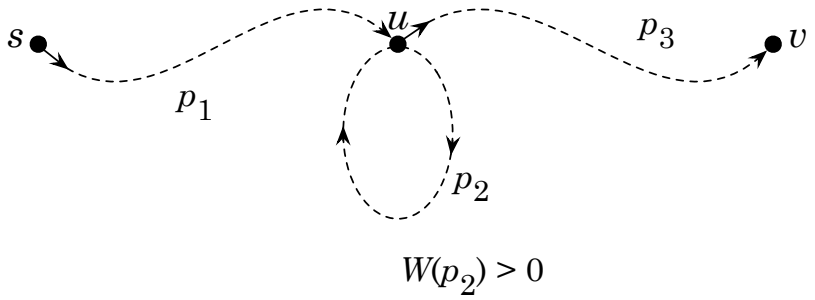
Cammini di costo minimo

- Dato un grafo $G = (V, E)$ la **lunghezza di un cammino** p su G è data dal numero di spigoli che lo compongono
- Un cammino di lunghezza minima tra due vertici u e v di G può essere trovato in $O(n + m)$ mediante una visita in ampiezza BFS da u o da v
- In diversi contesti applicativi è utile attribuire un costo, una capacità o un peso (in generale: un valore numerico non negativo) agli spigoli del grafo, ottenendo così un **grafo pesato** dato dalla coppia (G, w) con w: $E(G) \longrightarrow \mathbb{R}$ funzione costo/peso associata agli spigoli
- Possiamo definire il costo di un cammino p: $u \rightsquigarrow v$ come $W(p) = \sum_{(u,v)\in p} w(u, v)$
- Il **cammino di costo minimo** dal vertice u al vertice v è quindi il cammino per cui risulta $W(p)$ minimo; possiamo definire allora la **distanza pesata** $d(u, v)$ tra due vertici u e v di G come

$$d_{u,v} = \begin{cases} \min_{p:u\rightsquigarrow v} W(p) & \text{se esiste un cammino } p \text{ da } u \text{ a } v \\ \infty & \text{altrimenti} \end{cases}$$

Cammini di costo minimo

- La distanza pesata tra i vertici così definita soddisfa la **disuguaglianza triangolare** (è la principale caratteristica delle *distanze sugli spazi metrici*): $d_{u,v} \leq d_{u,z} + d_{z,v}$
- Un cammino di costo minimo è un *cammino semplice*: se tutti i pesi assegnati agli spigoli di G sono positivi, allora eventuali cicli presenti nel cammino ne aumentano inutilmente il costo



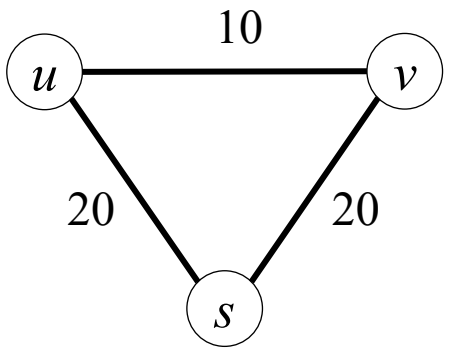
- **Prop.:** Ogni cammino di costo minimo gode della **proprietà di sottostruttura ottima**: se p: $u \rightsquigarrow v$ è un cammino di costo minimo da u a v e u', $v' \in p$, allora il sotto-cammino p' di p definito come p': $u' \rightsquigarrow v'$ è un cammino di costo minimo da u' a v'

  Dimostrazione: Se per assurdo p' non fosse un cammino minimo da u' a v' allora ne esisterebbe un altro costo inferiore: p^*: $u' \rightsquigarrow v'$ e quindi anche p non sarebbe un cammino di costo minimo da u a v, contraddicendo l'ipotesi ■

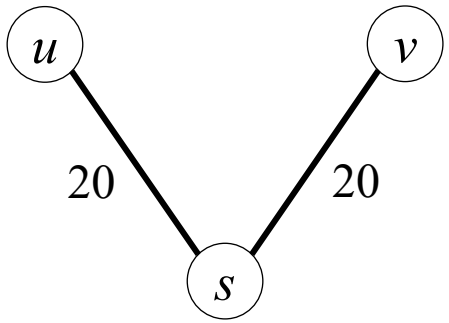
Cammini di costo minimo

- Fissato un vertice sorgente $s \in V(G)$ si può definire un albero orientato composto da cammini di costo minimo da s in ogni altro vertice $v \in V(G)$: per la proposizione precedente, infatti, è possibile scegliere fra i cammini di costo minimo quelli che non creano dei cicli

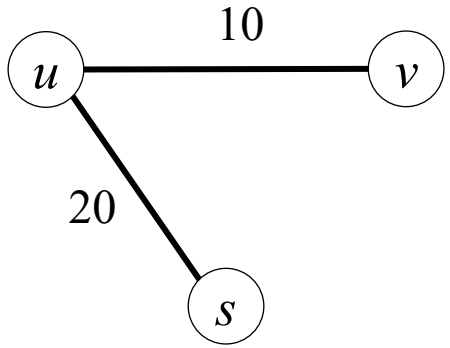
- È utile osservare che un albero di cammini di costo minimo non è necessariamente un albero ricoprente di costo minimo; per dimostrarlo basta esibire un esempio in cui si osserva la differenza



Un grafo G



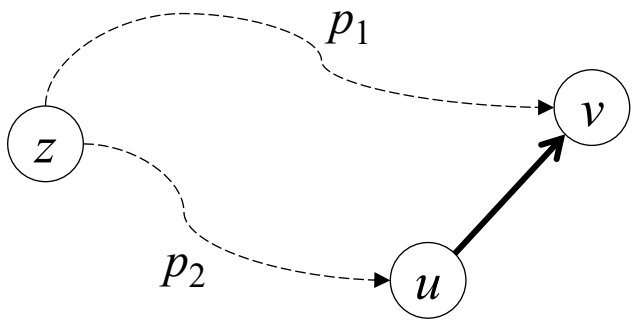
Un albero di cammini di costo minimo su G con sorgente in s



Un albero ricoprente di costo minimo su G

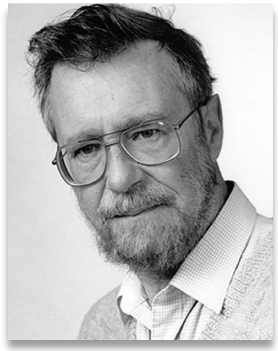
Cammini di costo minimo

- **Condizione di Bellmann**: Sia (G, w) un grafo orientato pesato con funzione costo w; allora per ogni $(u, v) \in E(G)$ e ogni vertice $z \in V(G)$ risulta $d(z, v) \leq d(z, u) + w(u, v)$



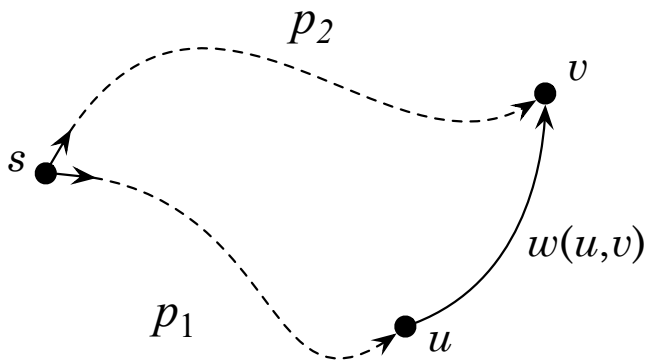
- Problemi sui cammini di costo minimo:
  - **Cammini minimi con sorgente singola**: fissata una singola sorgente $s \in V(G)$, trovare il cammino di costo minimo $p\colon s \longrightarrow v$ per ogni $v \in V(G)$
  - **Cammini minimi verso una singola destinazione**: fissato un singolo vertice di destinazione $t \in V(G)$, trovare il cammino minimo $p\colon v \longrightarrow t$ per ogni $v \in V(G)$
  - **Cammini minimi fra tutte le coppie**: per ogni coppia di vertici $u, v \in V(G)$ trovare il cammino di costo minimo che li collega
  - **Cammino di costo minimo fra una coppia di vertici**: fissati due vertici $u, v \in V(G)$ trovare un cammino di costo minimo che li collega

Algoritmo di Dijkstra



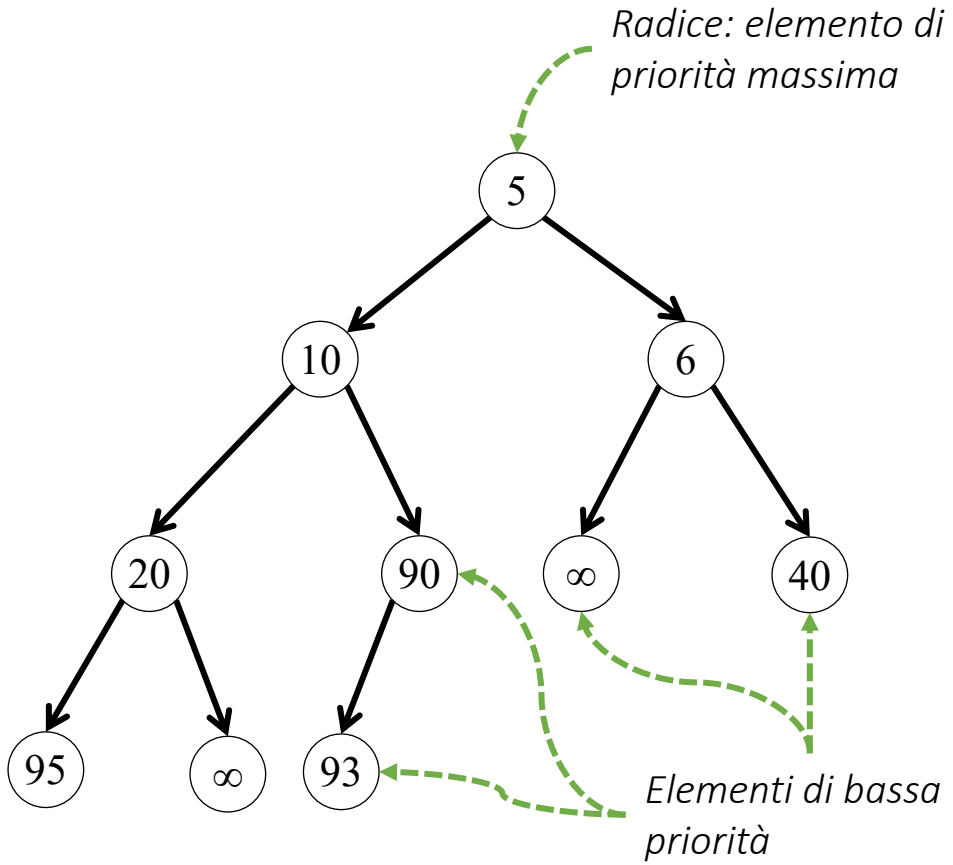
Edsger W. Dijkstra
(1930 – 2002)

- È forse l'algoritmo più noto per la soluzione del problema dei **cammini di costo minimo da una sorgente singola**
- L'autore, da cui prende il nome, **Edsger Wybe Dijkstra** è stato un celeberrimo informatico olandese, che ha lavorato a lungo presso l'Università del Texas negli Stati Uniti; nel 1972 ha vinto il ***Premio Turing*** per i suoi contributi fondamentali all'informatica teorica
  - Sul sito **https://www.cs.utexas.edu/users/EWD/** è disponibile una vasta raccolta delle sue dispense e dei suoi articoli, scritti con precisione ed eleganza con una macchina da scrivere
- La dinamica dell'algoritmo si basa sul **principio del rilassamento**:
  - sia p_1: $s \rightsquigarrow u$ un cammino da s a u con costo $W(p_1)$ e p_2: $s \rightsquigarrow v$ un cammino da s a v con costo $W(p_2)$; supponiamo che $(u, v) \in E(G)$ con costo $w(u, v)$
  - se $W(p_1) + w(u, v) < W(p_2)$ allora per andare da s a v conviene scegliere il cammino che passa per u, invece di usare il cammino p_2



Algoritmo di Dijkstra

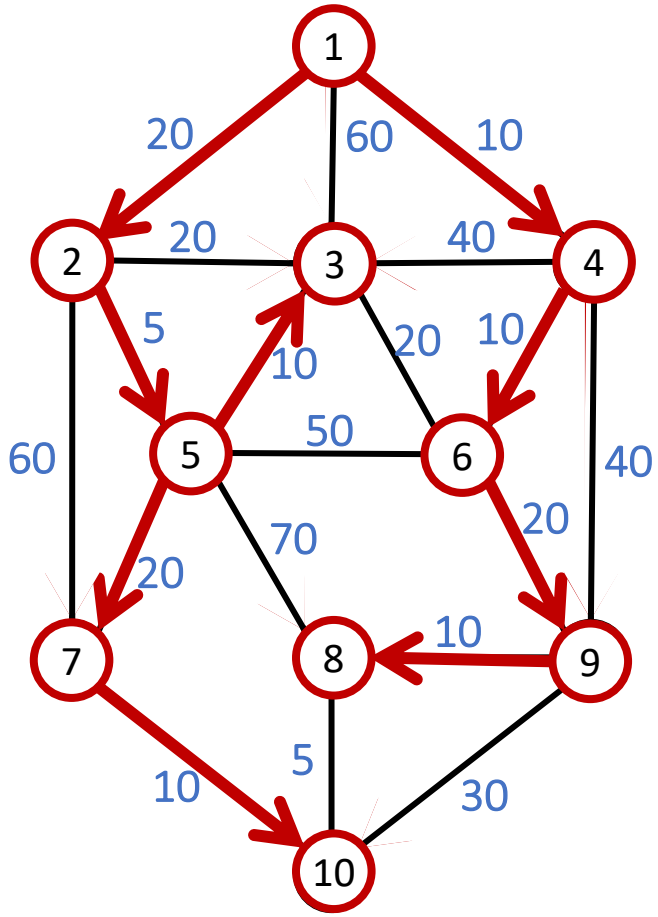
- Partendo dalla sorgente s fornita in input, l'algoritmo di volta in volta prende in esame il vertice più «vicino» alla sorgente tra quelli ancora da elaborare e applica il principio del rilassamento a tutti i vertici adiacenti
- L'algoritmo di Dijkstra utilizza una **coda di priorità** per scegliere il vertice di G da prendere in esame di volta in volta, utilizzando il costo del cammino minimo di ciascun vertice come valore su cui costruire la priorità degli elementi della coda: *a costo più basso corrisponde priorità più alta*
- Naturalmente, fino a quando non sono stati raggiunti ed elaborati, i vertici $v \in V(G)$ hanno una stima del costo del cammino minimo pari a $D_{s,v} = \infty$; solo la sorgente ha $D_{s,s} = 0$
- La coda di priorità viene implementata con una struttura dati di **heap binario**: un albero binario completo in cui l'elemento di priorità massima si trova sulla radice e ciascun vertice ha una priorità maggiore dei propri figli



Un heap binario H

- In un heap binario le operazioni di inserimento e di estrazione di un elemento e di modifica della priorità di un elemento hanno costo $O(\log_2 n)$

Algoritmo di Dijkstra



Q

$D_{s,v}$	0	20	35	10	25	20	45	50	40	55
	1	2	3	4	5	6	7	8	9	10

Algoritmo 18 DIJKSTRA(G, s)

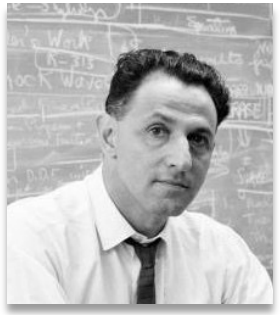
Input: Un grafo pesato (G,w) e un vertice $s \in V(G)$ scelto arbitrariamente come sorgente della visita

Output: Un albero di cammini minimi con radice in s costituito dai cammini di peso minimo che da s consentono di raggiungere ogni $v \in V(G)$

1: **per ogni** $v \in V(G)$ **ripeti**
2: $D_{s,v} := \infty$; $\pi(v) = NULL$
3: **fine-ciclo**
4: $D_{s,s} := 0$
5: sia Q una coda di priorità; $Q := V(G)$
6: **fintanto che** $Q \neq \emptyset$ **ripeti**
7: sia u il vertice di G estratto da Q con $D_{s,u}$ minimo in Q
8: **per ogni** $v \in N(u)$ **ripeti**
9: **se** $D_{s,v} \geq D_{s,u} + w(u,v)$ **allora**
10: $D_{s,v} := D_{s,u} + w(u,v)$, $\pi(v) = u$
11: **fine-condizione**
12: **fine-ciclo**
13: **fine-ciclo**

Complessità algoritmo di Dijkstra: $O(m \log_2 n)$, utilizzando un heap binario e tenendo conto che $n-1 \leq m < n^2$

Algoritmo di Bellman-Ford

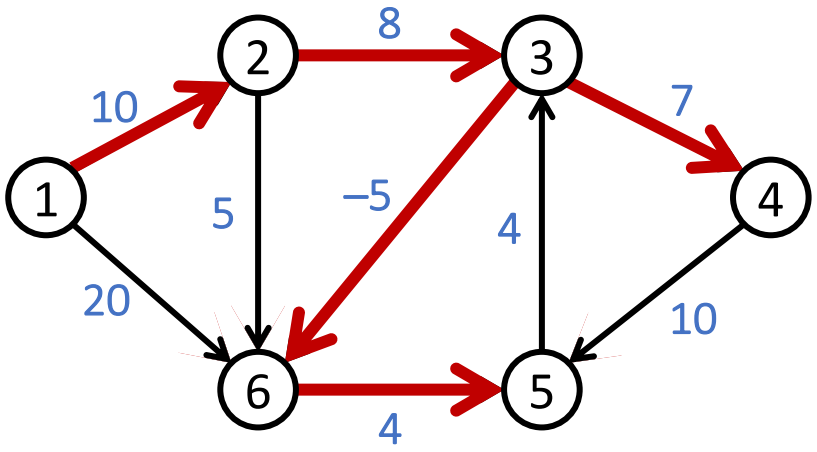


Richard Bellman
(1920 – 1984)

- Proposto indipendentemente da **Lester Ford jr.** nel 1954 e da **Richard Bellman** nel 1958, l'algoritmo di Bellman-Ford risolve il problema del cammino minimo da una sorgente singola
- Si basa sul principio del rilassamento e sulla considerazione che un qualsiasi cammino di costo minimo su un grafo G non può avere più di $n-1$ spigoli, se il grafo ha n vertici e non contiene *cicli* di peso negativo
- Ha una complessità computazionale superiore all'algoritmo di Dijkstra, ma può essere applicato anche su grafi con pesi negativi (purché non siano presenti cicli di costo negativo)
- Siccome ogni spigolo può comparire al massimo una volta nel cammino di costo minimo da s ad ogni vertice v del grafo, dopo aver valutato il contributo di ogni spigolo per $n-1$ volte, sarà stato certamente individuato il cammino di costo minimo per ciascun vertice del grafo
- Se invece, iterando il ciclo principale ancora una volta, qualche vertice riduce ulteriormente la sua stima di costo minimo da s, allora significa che nel grafo è presente un ciclo di costo negativo

Lester Ford jr.
(1927 – 2017)

Algoritmo di Bellman-Ford



$D_{s,v}$	0	10	18	25	17	13
	1	2	3	4	5	6

- Viene eseguito per $n-1$ volte un ciclo che itera su tutti gli m spigoli del grafo (righe 6-10): la complessità è $O(n \times m)$

Algoritmo 19 BELLMAN-FORD(G, s)

Input: Un grafo pesato (G,w) e un vertice $s \in V(G)$ scelto arbitrariamente come sorgente della visita

Output: Un albero con radice in s costituito dai cammini di peso minimo che da s consentono di raggiungere ogni $v \in V(G)$

```
1:  per ogni v ∈ V(G) ripeti
2:      D_{s,v} := ∞; π(v) = NULL
3:  fine-ciclo
4:  D_{s,s} := 0
5:  per i = 1,2,...,|V(G)| − 1 ripeti
6:      per ogni (u,v) ∈ E(G) ripeti
7:          se D_{s,v} > D_{s,u} + w(u,v) allora
8:              D_{s,v} := D_{s,u} + w(u,v), π(v) = u
9:          fine-condizione
10:     fine-ciclo
11: fine-ciclo
12: per ogni (u,v) ∈ E(G) ripeti
13:     se D_{s,v} > D_{s,u} + w(u,v) allora
14:         c'è un ciclo negativo!
15:     fine-condizione
16: fine-ciclo
```

Se esiste, identifica un ciclo di costo negativo

Ottimizzazione dell'algoritmo di Bellman-Ford

- L'algoritmo può essere reso più efficiente scegliendo opportunamente l'ordine con cui considerare gli spigoli
- Se il grafo è orientato e aciclico (DAG – *directed acyclic graph*) allora scegliendo i vertici nell'ordine topologico e gli spigoli uscenti da tali vertici in un ordine qualsiasi, si riduce drasticamente la complessità dell'algoritmo: non è necessario considerare più volte lo stesso spigolo
- In questo caso la complessità computazionale è $O(n+m)$

Algoritmo 20 BELLMAN-FORD BIS(G, s)

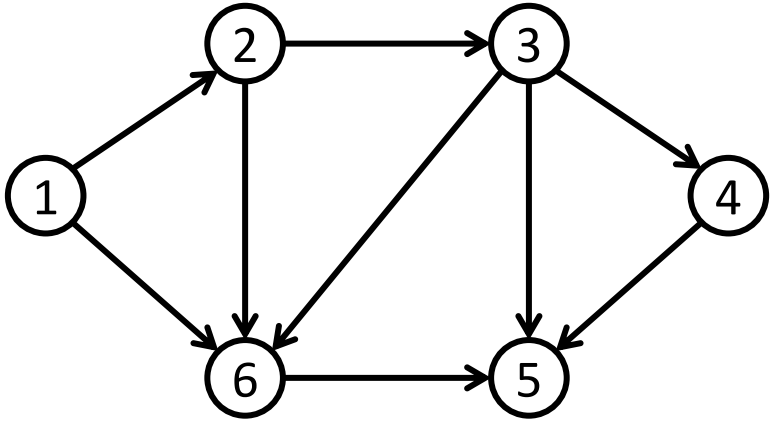
Input: Un grafo pesato (G, w) orientato e aciclico e un vertice $s \in V(G)$ scelto arbitrariamente come sorgente della visita

Output: Un albero di cammini minimi con radice in s costituito dai cammini di peso minimo che da s consentono di raggiungere ogni $v \in V(G)$

1: **per ogni** $v \in V(G)$ **ripeti**
2: $D_{s,v} := \infty$; $\pi(v) = NULL$
3: **fine-ciclo**
4: $D_{s,s} := 0$
5: esegui l'ordinamento topologico di $V(G)$
6: **per ogni** $u \in |V(G)|$, in ordine topologico **ripeti**
7: **per ogni** $v \in N(u)$ **ripeti**
8: **se** $D_{s,v} \geq D_{s,u} + w(u,v)$ **allora**
9: $D_{s,v} := D_{s,u} + w(u,v)$, $\pi(v) = u$
10: **fine-condizione**
11: **fine-ciclo**
12: **fine-ciclo**

Ordinamento topologico dei vertici di un DAG

- Un **ordinamento topologico** dei vertici di un grafo orientato aciclico (DAG) è una relazione d'ordine tale che, per ogni $u, v \in V(G)$ se $(u, v) \in E(G)$ allora $u \leq v$
- L'ordinamento topologico non può essere definito se il grafo contiene dei cicli



- Considerando il grafo in figura l'ordinamento topologico deve tenere conto delle seguenti disuguaglianze:

$$1 \leq 2,\ 1 \leq 6,\ 2 \leq 3,\ 2 \leq 6,\ 3 \leq 4,\ 3 \leq 5,\ 3 \leq 6,\ 4 \leq 5,\ 6 \leq 5$$

- Un possibile ordinamento topologico è quindi: $1, 2, 3, 6, 4, 5$
- L'ordinamento topologico è una ***relazione d'ordine parziale***: può non essere unico (ad esempio anche: $1, 2, 3, \mathbf{4}, \mathbf{6}, 5$)

Ordinamento topologico dei vertici di un DAG

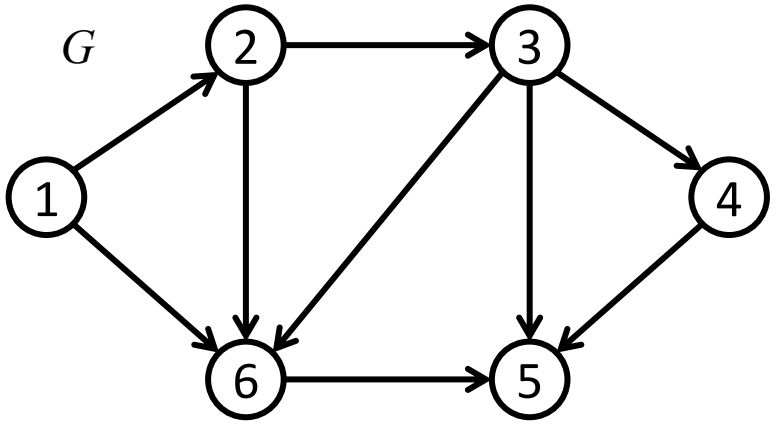
- Un ordinamento topologico dei vertici di un grafo G orientato aciclico lo si può ottenere con una visita in profondità di G (DFS), tenendo conto dei tempi di fine visita di ciascun vertice: l'ordinamento topologico è dato dai tempi di fine visita in ordine decrescente
- La complessità di tale procedimento è $O(n+m)$
- In alternativa si può utilizzare l'algoritmo in figura: si calcola il grado entrante di ogni vertice, quindi si costruisce una coda Q con cui si definisce l'ordinamento topologico di $V(G)$
- Si smantella progressivamente il grafo G (o una sua copia G') eliminando i vertici privi di spigoli entranti e accodandoli a Q, finché il grafo è vuoto e la coda Q contiene tutti i suoi vertici
- Complessità: $O(n \log_2 n + m)$ se si gestiscono i vertici di G' in una coda di priorità (*heap binario*) in base al numero di spigoli entranti

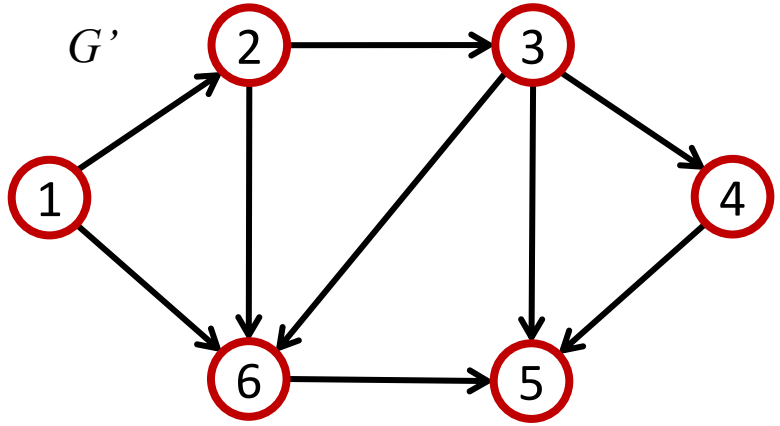
Algoritmo 21 SORT TOPOLOGICO(G)

Input: Un grafo (G) orientato e aciclico
Output: Un ordinamento topologico dei suoi vertici tale che $u < v$ se $(u,v) \in E(G)$

1: $G' = G$
2: $Q = \emptyset$
3: **fintanto che** esiste in G' un vertice u senza spigoli entranti **ripeti**
4: accoda u in Q
5: rimuovi da G' il vertice u e gli spigoli uscenti da u
6: **fine-ciclo**

Ordinamento topologico dei vertici di un DAG





Q | 1 2 3 6 4 5

Algoritmo 21 SORT TOPOLOGICO(G)

Input: Un grafo (G) orientato e aciclico

Output: Un ordinamento topologico dei suoi vertici tale che $u < v$ se $(u,v) \in E(G)$

1: $G' = G$
2: $Q = \emptyset$
3: **fintanto che** esiste in G' un vertice u senza spigoli entranti **ripeti**
4: accoda u in Q
5: rimuovi da G' il vertice u e gli spigoli uscenti da u
6: **fine-ciclo**

Programmazione dinamica

- La **programmazione dinamica** è una tecnica generale per la risoluzione di problemi di ottimizzazione combinatoria; la applicheremo al problema del cammino di costo minimo fra tutte le coppie di vertici di G
- Questa tecnica è stata introdotta da Bellman negli anni ‘40
- Si può applicare solo se il problema gode della proprietà di **sottostruttura ottima della soluzione** (come nel caso del problema del cammino di costo minimo)
- Il procedimento è di tipo ***bottom-up*** (*divide et impera* è *top-down*):
  - a partire dalla soluzione di sottoproblemi elementari (il cammino di costo minimo fra vertici adiacenti, ad esempio) si aggregano le soluzioni, calcolando la soluzione ottima di problemi sempre più grandi, fino a risolvere il problema originario
  - il procedimento si basa su un’**espressione ricorsiva** con cui si calcolano in successione le soluzioni dei problemi più grandi, basandosi sulle soluzioni di quelli più piccoli calcolate ai passi precedenti
  - in questo modo viene calcolata una **sequenza di matrici quadrate** di ordine n, $A^{(1)}$, $A^{(2)}$, ..., $A^{(n)}$ con un’espressione ricorsiva del tipo $A^{(k)} = f(A^{(k-1)})$, per $k > 1$ e costruendo a priori $A^{(1)}$ in modo diretto con dei valori costanti
  - Gli elementi $A_{i,j}{}^{(k)}$ delle matrici quadrate rappresentano, al passo k del procedimento risolutivo, il costo minimo, fin qui calcolato, per un cammino dal vertice v_i al vertice v_j

Programmazione dinamica

- Indichiamo con $w(v_i, v_j)$ il costo dello spigolo $(v_i, v_j) \in E(G)$; in questo modo si può definire una matrice quadrata W di ordine n:

$$W_{i,j} = \begin{cases} 0 & \text{se } v_i = v_j \\ w(v_i, v_j) & \text{se } (v_i, v_j) \in E(G) \\ \infty & \text{se } i \neq j \text{ e } (v_i, v_j) \notin E(G) \end{cases}$$

- Per applicare la tecnica della programmazione dinamica costruiremo una successione di matrici

$$L^{(0)}, L^{(1)}, \ldots, L^{(k)}, \ldots, L^{(n-1)}$$

- L'elemento generico della matrice $l_{i,j}^{(k)}$ rappresenta il costo del cammino di costo minimo da v_i a v_j definito utilizzando al più k spigoli (il costo del cammino di lunghezza minore o uguale a k)

$$l_{i,j}^{(0)} = \begin{cases} 0 & \text{se } i = j \\ \infty & \text{altrimenti} \end{cases} \qquad l_{i,j}^{(1)} = \begin{cases} 0 & \text{se } i = j \\ w(v_i, v_j) & \text{se } (v_i, v_j) \in E(G) \\ \infty & \text{se} i \neq j \text{ e } (v_i, v_j) \notin E(G) \end{cases}$$

Programmazione dinamica

- In generale si ha (espressione ricorsiva per l'algoritmo di programmazione dinamica):

$$l_{i,j}^{(0)} = \begin{cases} 0 & \text{se } i = j \\ \infty & \text{altrimenti} \end{cases}$$

$$l_{i,j}^{(1)} = \begin{cases} 0 & \text{se } i = j \\ w(v_i, v_j) & \text{se } (v_i, v_j) \in E(G) \\ \infty & \text{se} i \neq j \text{ e } (v_i, v_j) \notin E(G) \end{cases}$$

$$l_{u,v}^{(k)} = \min\left\{ l_{u,v}^{(k-1)}, \min_{1 \leq z \leq n} \left\{ l_{u,z}^{(k-1)} + w(z,v) \right\} \right\}$$

$$= \min_{1 \leq z \leq n} \left\{ l_{u,z}^{(k-1)} + w(z,v) \right\}$$

perché $w(v, v) = 0$ per ogni $v \in V(G)$

- Il procedimento parte da $l_{i,j}{}^{(0)}$ e rilassa man mano il vincolo sulla massima lunghezza di un cammino di costo minimo, fino a $n - 1$ (massima lunghezza di un cammino di costo minimo, quindi privo di cicli)

Cammini di costo minimo fra tutte le coppie

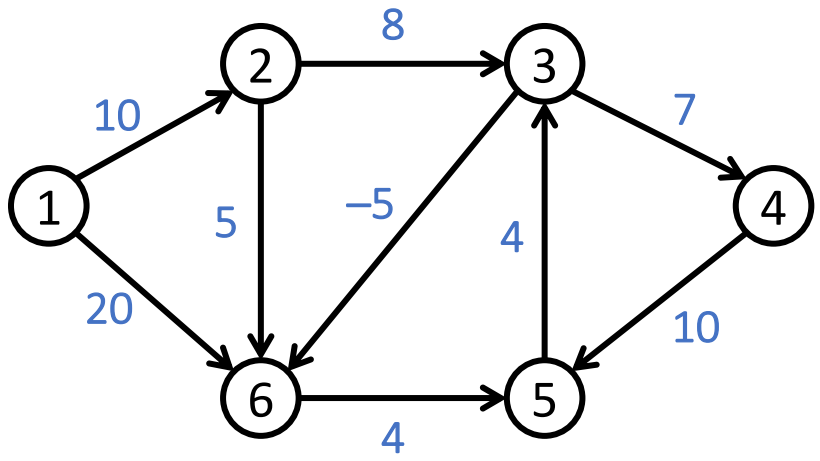
- Osserviamo che $L^{(1)} = W$
- Inoltre, un cammino di peso minimo tra u e v su un grafo $G = (V, E)$ con $|V(G)| = n$ non può avere più di $n - 1$ spigoli quindi il costo del cammino di costo minimo da u a v è $\delta(u, v) = l_{u,v}^{(n-1)}$
- Deve risultare $l_{u,v}^{(h)} = l_{u,v}^{(n-1)}$ per ogni $h \geq n$, perché il cammino minimo è semplice (in assenza di cicli negativi, come nel nostro caso)
- Se il cammino di costo minimo da u a v ha h spigoli, allora $l_{u,v}^{(h)}$ è il costo del cammino minimo e risulta $l_{u,v}^{(h)} = l_{u,v}^{(q)}$ per ogni $q > h$
- La complessità dell'algoritmo è $O(n^4)$

Algoritmo 22 ALLPAIRSSHORTESTPATH(G, w)

Input: Un grafo G orientato con funzione peso $w : E(G) \to \mathbb{R}$
Output: La matrice dei pesi dei cammini minimi per ciascuna coppia di vertici in $V(G)$

1: $W = (w_{ij})$ con $w_{ij} = w(i, j)$ per ogni $i, j = 1, 2, \ldots, n$
2: $L^{(1)} = W$
3: **per** $k = 2, \ldots, n-1$ **ripeti**
4: **per** $i = 1, 2, \ldots, n$ **ripeti**
5: **per** $j = 1, 2, \ldots, n$ **ripeti**
6: $l_{ij}^{(k)} = \infty$
7: **per** $h = 1, 2, \ldots, n$ **ripeti**
8: $l_{ij}^{(k)} = \min\{l_{ij}^{(k)}, l_{ih}^{(k-1)} + w_{hj}\}$
9: **fine-ciclo**
10: **fine-ciclo**
11: **fine-ciclo**
12: **fine-ciclo**

Cammini di costo minimo fra tutte le coppie



$L^{(1)}$

	1	2	3	4	5	6
1	0	10	∞	∞	∞	20
2	∞	0	8	∞	∞	5
3	∞	∞	0	7	∞	–5
4	∞	∞	∞	0	10	∞
5	∞	∞	4	∞	0	∞
6	∞	∞	∞	∞	4	0

Algoritmo 22 ALLPAIRSSHORTESTPATH(G, w)

Input: Un grafo G orientato con funzione peso $w : E(G) \to \mathbb{R}$

Output: La matrice dei pesi dei cammini minimi per ciascuna coppia di vertici in $V(G)$

1: $W = (w_{ij})$ con $w_{ij} = w(i, j)$ per ogni $i, j = 1, 2, \ldots, n$
2: $L^{(1)} = W$
3: **per** $k = 2, \ldots, n-1$ **ripeti**
4: **per** $i = 1, 2, \ldots, n$ **ripeti**
5: **per** $j = 1, 2, \ldots, n$ **ripeti**
6: $l_{ij}^{(k)} = \infty$
7: **per** $h = 1, 2, \ldots, n$ **ripeti**
8: $l_{ij}^{(k)} = \min\{l_{ij}^{(k)}, l_{ih}^{(k-1)} + w_{hj}\}$
9: **fine-ciclo**
10: **fine-ciclo**
11: **fine-ciclo**
12: **fine-ciclo**

Cammini di costo minimo fra tutte le coppie

$L^{(1)}$

	1	2	3	4	5	6
1	0	10	∞	∞	∞	20
2	∞	0	8	∞	∞	5
3	∞	∞	0	7	∞	–5
4	∞	∞	∞	0	10	∞
5	∞	∞	4	∞	0	∞
6	∞	∞	∞	∞	4	0

$L^{(2)}$

	1	2	3	4	5	6
1	0	10	18	∞	24	15
2	∞	0	8	15	9	3
3	∞	∞	0	7	–1	–5
4	∞	∞	14	0	10	∞
5	∞	∞	4	11	0	–1
6	∞	∞	8	∞	4	0

Algoritmo 22 ALLPAIRSSHORTESTPATH(G, w)

Input: Un grafo G orientato con funzione peso $w : E(G) \to \mathbb{R}$

Output: La matrice dei pesi dei cammini minimi per ciascuna coppia di vertici in $V(G)$

```
1: W = (w_ij) con w_ij = w(i, j) per ogni i, j = 1, 2, ..., n
2: L^(1) = W
3: per k = 2, ..., n − 1 ripeti
4:     per i = 1, 2, ..., n ripeti
5:         per j = 1, 2, ..., n ripeti
6:             l_ij^(k) = ∞
7:             per h = 1, 2, ..., n ripeti
8:                 l_ij^(k) = min{l_ij^(k), l_ih^(k−1) + w_hj}
9:             fine-ciclo
10:        fine-ciclo
11:    fine-ciclo
12: fine-ciclo
```

Cammini di costo minimo fra tutte le coppie

$L^{(2)}$

	1	2	3	4	5	6
1	0	10	18	∞	24	15
2	∞	0	8	15	9	3
3	∞	∞	0	7	–1	–5
4	∞	∞	14	0	10	∞
5	∞	∞	4	11	0	–1
6	∞	∞	8	∞	4	0

$L^{(3)}$

	1	2	3	4	5	6
1	0	10	18	25	19	13
2	∞	0	8	15	7	3
3	∞	∞	0	7	–1	–5
4	∞	∞	14	0	10	9
5	∞	∞	4	11	0	–1
6	∞	∞	8	15	4	0

Algoritmo 22 ALLPAIRSSHORTESTPATH(G, w)

Input: Un grafo G orientato con funzione peso $w : E(G) \to \mathbb{R}$

Output: La matrice dei pesi dei cammini minimi per ciascuna coppia di vertici in $V(G)$

1: $W = (w_{ij})$ con $w_{ij} = w(i, j)$ per ogni $i, j = 1, 2, \ldots, n$
2: $L^{(1)} = W$
3: **per** $k = 2, \ldots, n-1$ **ripeti**
4: **per** $i = 1, 2, \ldots, n$ **ripeti**
5: **per** $j = 1, 2, \ldots, n$ **ripeti**
6: $l_{ij}^{(k)} = \infty$
7: **per** $h = 1, 2, \ldots, n$ **ripeti**
8: $l_{ij}^{(k)} = \min\{l_{ij}^{(k)}, l_{ih}^{(k-1)} + w_{hj}\}$
9: **fine-ciclo**
10: **fine-ciclo**
11: **fine-ciclo**
12: **fine-ciclo**

Cammini di costo minimo fra tutte le coppie

$L^{(3)}$

	1	2	3	4	5	6
1	0	10	18	25	19	13
2	∞	0	8	15	7	3
3	∞	∞	0	7	–1	–5
4	∞	∞	14	0	10	9
5	∞	∞	4	11	0	–1
6	∞	∞	8	15	4	0

$L^{(4)}$

	1	2	3	4	5	6
1	0	10	18	25	17	13
2	∞	0	8	15	7	3
3	∞	∞	0	7	–1	–5
4	∞	∞	14	0	10	9
5	∞	∞	4	11	0	–1
6	∞	∞	8	15	4	0

Algoritmo 22 ALLPAIRSSHORTESTPATH(G, w)

Input: Un grafo G orientato con funzione peso $w : E(G) \to \mathbb{R}$

Output: La matrice dei pesi dei cammini minimi per ciascuna coppia di vertici in $V(G)$

```
1: W = (w_ij) con w_ij = w(i, j) per ogni i, j = 1, 2, ..., n
2: L^(1) = W
3: per k = 2, ..., n − 1 ripeti
4:     per i = 1, 2, ..., n ripeti
5:         per j = 1, 2, ..., n ripeti
6:             l_ij^(k) = ∞
7:             per h = 1, 2, ..., n ripeti
8:                 l_ij^(k) = min{l_ij^(k), l_ih^(k−1) + w_hj}
9:             fine-ciclo
10:        fine-ciclo
11:    fine-ciclo
12: fine-ciclo
```

Cammini di costo minimo fra tutte le coppie

$L^{(4)}$

	1	2	3	4	5	6
1	0	10	18	25	17	13
2	∞	0	8	15	7	3
3	∞	∞	0	7	–1	–5
4	∞	∞	14	0	10	9
5	∞	∞	4	11	0	–1
6	∞	∞	8	15	4	0

$L^{(5)}$

	1	2	3	4	5	6
1	0	10	18	25	17	13
2	∞	0	8	15	7	3
3	∞	∞	0	7	–1	–5
4	∞	∞	14	0	10	9
5	∞	∞	4	11	0	–1
6	∞	∞	8	15	4	0

Algoritmo 22 ALLPAIRSSHORTESTPATH(G, w)

Input: Un grafo G orientato con funzione peso $w : E(G) \to \mathbb{R}$

Output: La matrice dei pesi dei cammini minimi per ciascuna coppia di vertici in $V(G)$

```
1: W = (w_ij) con w_ij = w(i, j) per ogni i, j = 1, 2, ..., n
2: L^(1) = W
3: per k = 2, ..., n − 1 ripeti
4:     per i = 1, 2, ..., n ripeti
5:         per j = 1, 2, ..., n ripeti
6:             l_ij^(k) = ∞
7:             per h = 1, 2, ..., n ripeti
8:                 l_ij^(k) = min{l_ij^(k), l_ih^(k−1) + w_hj}
9:             fine-ciclo
10:        fine-ciclo
11:    fine-ciclo
12: fine-ciclo
```

Cammini di costo minimo fra tutte le coppie

- Con l'algoritmo appena visto calcoliamo il costo del cammino di costo minimo, ma non lasciamo traccia di *quale* sia tale cammino
- Per **tracciare i cammini di costo minimo** possiamo definire una successione di matrici $\Pi^{(k)}$ il cui generico elemento $\pi_{ij}^{(k)}$ indica il padre del vertice v_j nel cammino da v_i a v_j con al più k spigoli
- Per questo occorre modificare l'algoritmo aggiungendo a riga 2: $\pi_{ij}^{(k)} = \text{null}$ per ogni $v_i, v_j \in V(G)$
- Inoltre l'istruzione a riga 8 deve essere sostituita con le seguenti:

se $l_{ij}^{(k)} > l_{ih}^{(k-1)} + w_{hj}$ **allora** $\pi_{ij}^{(k)} := h$
altrimenti $\pi_{ij}^{(k)} = \pi_{ij}^{(k-1)}$

Algoritmo 22 ALLPAIRSSHORTESTPATH(G, w)

Input: Un grafo G orientato con funzione peso $w : E(G) \to \mathbb{R}$
Output: La matrice dei pesi dei cammini minimi per ciascuna coppia di vertici in $V(G)$

1: $W = (w_{ij})$ con $w_{ij} = w(i, j)$ per ogni $i, j = 1, 2, \ldots, n$
2: $L^{(1)} = W$
3: **per** $k = 2, \ldots, n-1$ **ripeti**
4: **per** $i = 1, 2, \ldots, n$ **ripeti**
5: **per** $j = 1, 2, \ldots, n$ **ripeti**
6: $l_{ij}^{(k)} = \infty$
7: **per** $h = 1, 2, \ldots, n$ **ripeti**
8: $l_{ij}^{(k)} = \min\{l_{ij}^{(k)}, l_{ih}^{(k-1)} + w_{hj}\}$
9: **fine-ciclo**
10: **fine-ciclo**
11: **fine-ciclo**
12: **fine-ciclo**

Una versione più efficiente dello stesso algoritmo

- Nell'algoritmo appena visto la tecnica della programmazione dinamica è utilizzata incrementando di volta in volta di un solo spigolo la massima lunghezza dei cammini di costo minimo
- Per accelerare l'esecuzione dell'algoritmo si può procedere aggregando le soluzioni di due problemi di dimensione pari alla metà della dimensione del problema che stiamo considerando
- Anziché aumentare solo di 1 la dimensione del problema considerato, possiamo raddoppiarne la dimensione ad ogni passo):

$$l_{uv}^{k} = \min_{1 \le z \le n} \{ l_{uz}^{(k/2)} + l_{zv}^{(k/2)} \}$$

- La complessità dell'algoritmo diventa: $O(n^3 \log_2 n)$

Algoritmo 23 FASTALLPAIRSSHORTESTPATH(G, w)

Input: Un grafo G orientato con funzione peso $w : E(G) \to \mathbb{R}$

Output: La matrice dei pesi dei cammini minimi per ciascuna coppia di vertici in $V(G)$

1: $W = (w_{ij})$ con $w_{ij} := w(i, j)$ per ogni $i, j = 1, 2, \ldots, n$
2: $L^{(1)} := W$
3: $k := 1$
4: **fintanto che** $k < n - 1$ **ripeti**
5: **per** $i := 1, 2, \ldots, n$ **ripeti**
6: **per** $j := 1, 2, \ldots, n$ **ripeti**
7: $l_{ij}^{(2k)} := \infty$
8: **per** $h := 1, 2, \ldots, n$ **ripeti**
9: $l_{ij}^{(2k)} := \min\{l_{ij}^{(2k)}, l_{ih}^{(k)} + l_{hj}^{(k)}\}$
10: **fine-ciclo**
11: **fine-ciclo**
12: **fine-ciclo**
13: $k := 2k$
14: **fine-ciclo**

Algoritmo di Floyd-Warshall

Robert W. Floyd
(1936 – 2001)

- L'algoritmo utilizza la tecnica della programmazione dinamica e fu pubblicato indipendentemente da **Floyd** e da **Warshall** nel 1962 su *Communication of the ACM* e sul *Journal of the ACM*
- Sia p: $u \rightsquigarrow v$ un cammino da u a v dato da $p = \langle v_1, v_2, ..., v_k \rangle$, con $u = v_1$ e $v = v_k$; i vertici $v_2, ..., v_{k-1}$ sono ***intermedi*** in p
- Sia $V(G) = \{1, 2, ..., n\}$ e, preso un intero $k \leq n$, consideriamo tutti i cammini da u a v con vertici intermedi appartenenti all'insieme $\{1, 2, ..., k\}$; sia p un cammino di costo minimo tra questi

Stephen Warshall
(1935 – 2006)

- L'**algoritmo di Floyd e Warshall** per il calcolo dei cammini di costo minimo fra tutte le coppie di vertici di un grafo con pesi assegnati agli spigoli (e privo di cicli di costo negativo) si basa su alcune semplici considerazioni:
  - se k **non è un vertice intermedio** di p allora i vertici intermedi di p sono tutti in $\{1, 2, ..., k-1\}$ e quindi un cammino minimo da u a v con vertici intermedi in $\{1, 2, ..., k-1\}$ è anche un cammino minimo da u a v con vertici intermedi in $\{1, 2, ..., k\}$
  - se k **è un vertice intermedio** di p allora possiamo spezzare p in due sotto-cammini: p_1: $u \rightsquigarrow k$ e p_2: $k \rightsquigarrow v$
  - per la sottostruttura ottima dei cammini minimi risulta che p_1 è un cammino minimo da u a k e che p_2 è un cammino minimo da k a v con vertici intermedi in $\{1, ..., k\}$
  - però è vero anche che sia p_1 che p_2 hanno vertici intermedi in $\{1, ..., k-1\}$

Algoritmo di Floyd-Warshall

- Sulla base delle considerazioni precedenti possiamo costruire il procedimento bottom-up su cui si basa l'algoritmo di Floyd e Warshall
- Sia $d_{uv}{}^{(k)}$ il peso del cammino da u a v con vertici intermedi in $\{1, 2, ..., k\}$; vale la seguente definizione ricorsiva per le matrici $D^{(k)}$ quadrate di ordine n:

$$d_{uv}^{(k)} = \begin{cases} w_{uv} & \text{se } k = 0 \\ \min\left\{ d_{uv}^{(k-1)}, d_{uk}^{(k-1)} + d_{kv}^{(k-1)} \right\} & \text{se } k \geq 1 \end{cases}$$

- Visto che un cammino minimo da u a v è semplice e che sicuramente tutti i vertici intermedi si trovano in $\{1, ..., n\}$ la matrice $D^{(n)} = (d^{(n)}{}_{uv})$ fornisce la soluzione finale del problema, ossia risulta $d^{(n)}{}_{uv} = \delta(u,v)$ per ogni $u, v \in V(G)$

Algoritmo di Floyd-Warshall

- Il procedimento iterativo per la costruzione delle matrici $D^{(k)}$ è molto semplice e si basa sulla definizione ricorsiva appena indicata
- La complessità dell'algoritmo è $\Theta(n^3)$
- L'algoritmo si presta bene ad una implementazione priva di strutture dati complicate: sono sufficienti delle matrici numeriche

Algoritmo 24 FLOYDWARSHALL(G, w)

Input: Un grafo G orientato con funzione peso $w : E(G) \to \mathbb{R}$

Output: La matrice dei pesi dei cammini minimi per ciascuna coppia di vertici in $V(G)$

1: $W := (w_{ij})$ con $w_{ij} := \begin{cases} 0 & \text{se } i = j \\ w(i,j) & \text{se } (i,j) \in E(G) \\ \infty & \text{altrimenti} \end{cases}$

2: $D^{(0)} := W$

3: **per** $k := 1, \ldots, n$ **ripeti**

4: **per** $i := 1, \ldots, n$ **ripeti**

5: **per** $j := 1, \ldots, n$ **ripeti**

6: $d_{ij}^{(k)} := \min\left\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right\}$

7: **fine-ciclo**

8: **fine-ciclo**

9: **fine-ciclo**

Algoritmo di Floyd-Warshall

- Integriamo l'algoritmo di Floyd-Warshall per calcolare anche i cammini costo minimo e non solo il costo di tali cammini
- Per far questo definiamo la matrice $\Pi^{(n)}$ in cui $\pi_{uv}{}^{(n)}$ rappresenta il predecessore di v nel cammino di costo minimo da u a v
- Il calcolo di $\pi_{uv}{}^{(n)}$ si basa su $\pi_{uv}{}^{(n-1)}$ sostanzialmente con lo stesso schema di programmazione dinamica con cui sono state calcolate le distanze; la base del procedimento ricorsivo è la seguente:

$$\pi_{uv}^{(0)} = \begin{cases} null & \text{se } u = v \text{ o } w(u,v) = \infty \text{ (cioè se } (u,v) \notin E(G)) \\ u & \text{se } u \neq v \text{ e } w(u,v) < \infty \text{ (cioè se } (u,v) \in E(G)) \end{cases}$$

- Al passo generico dell'algoritmo possiamo calcolare $\pi_{uv}{}^{(k)}$ con la seguente espressione ricorsiva:

$$\pi_{uv}^{(k)} = \begin{cases} \pi_{uv}^{(k-1)} & \text{se } d_{uv}^{(k-1)} \leq d_{uk}^{(k-1)} + d_{kv}^{(k-1)} \\ \pi_{kv}^{(k-1)} & \text{se } d_{uv}^{(k-1)} > d_{uk}^{(k-1)} + d_{kv}^{(k-1)} \end{cases}$$

Algoritmo di Floyd-Warshall

- Possiamo riscrivere l'algoritmo di Floyd-Warshall nel modo seguente, consentendo così il calcolo del costo dei cammini minimi per ogni coppia di vertici $u, v \in V(G)$ mediante la matrice $D^{(n)}$ ed anche la ricostruzione dei cammini stessi, attraverso la matrice $\Pi^{(n)}$

- La complessità dell'algoritmo è la medesima: $\Theta(n^3)$

Algoritmo 25 FLOYDWARSHALL(G, w)

Input: Un grafo G orientato con funzione peso $w : E(G) \to \mathbb{R}$
Output: La matrice dei pesi dei cammini minimi per ciascuna coppia di vertici in $V(G)$

1: $W := (w_{ij})$ con $w_{ij} := w(i,j)$ per ogni $i, j = 1, 2, \ldots, n$
2: $D^{(0)} := W$
3: **per** $u := 1, 2, \ldots, n$ **ripeti**
4: **per** $v := 1, 2, \ldots, n$ **ripeti**
5: **se** $u = v$ o $w_{uv} = \infty$ **allora**
6: $\pi_{uv}^{(0)} := null$
7: **altrimenti**
8: $\pi_{uv}^{(0)} := u$
9: **fine-condizione**
10: **fine-ciclo**
11: **fine-ciclo**
12: **per** $k := 1, 2, \ldots, n$ **ripeti**
13: **per** $u := 1, 2, \ldots, n$ **ripeti**
14: **per** $v := 1, 2, \ldots, n$ **ripeti**
15: **se** $d_{uv}^{(k-1)} \leq d_{uk}^{(k-1)} + d_{kv}^{(k-1)}$ **allora**
16: $\pi_{uv}^{(k)} := \pi_{uv}^{(k-1)}$, $d_{uv}^{(k)} = d_{uv}^{(k-1)}$
17: **altrimenti**
18: $\pi_{uv}^{(k)} := \pi_{kv}^{(k-1)}$, $d_{uv}^{(k)} = d_{uk}^{(k-1)} + d_{kv}^{(k-1)}$
19: **fine-condizione**
20: **fine-ciclo**
21: **fine-ciclo**
22: **fine-ciclo**

inizializzazione

calcolo

Algoritmo di Floyd-Warshall

- I cammini di costo minimo vengono calcolati con l'algoritmo di Floyd-Warshall e sono rappresentati attraverso la matrice $\Pi^{(n)}$ in cui sono indicati i rapporti padre-figlio o predecessore-successore tra i vertici nei diversi cammini di costo minimo ($\pi_{uv}^{(n)}$ rappresenta il predecessore del vertice v nel cammino da u a v)
- Per visualizzare il cammino di costo minimo dal vertice u al vertice v è possibile utilizzare il seguente algoritmo ricorsivo

Algoritmo 26 STAMPACAMMINO$(u, v, \Pi^{(n)})$

Input: La matrice dei predecessori $\Pi^{(n)}$, il vertice di partenza u e quello di arrivo v

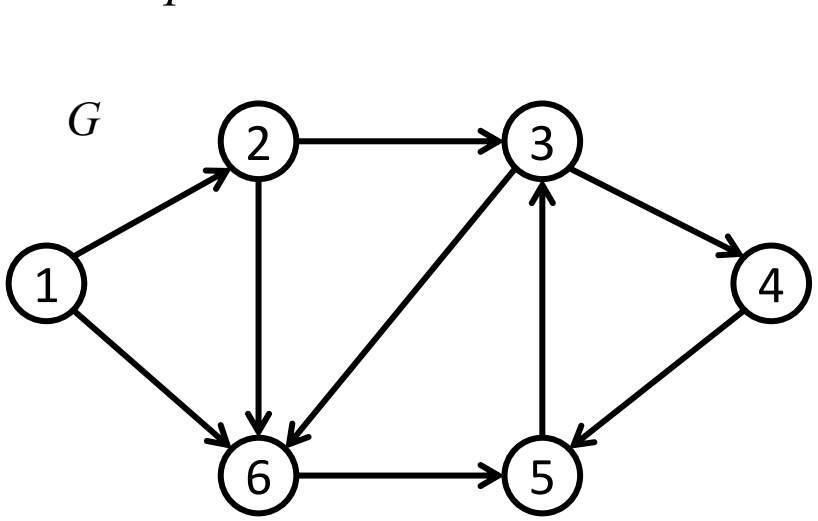
Output: La sequenza $\langle v_1, v_2, \ldots, v_k \rangle$ che costituisce il cammino da u a v calcolato con FLOYD-WARSHALL

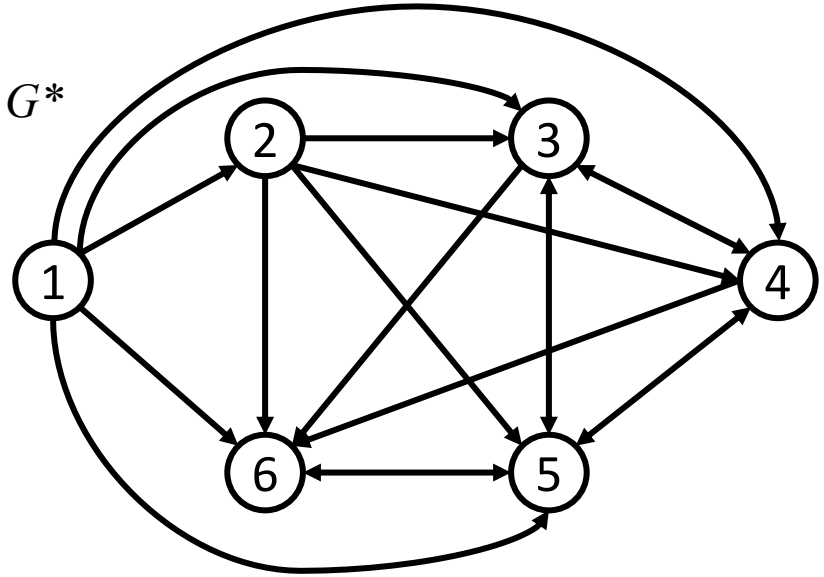
1: **se** $u \neq v$ **allora**
2: STAMPACAMMINO$(u, \pi_{uv}^{(n)}, \Pi^{(n)})$
3: scrivi v
4: **fine-condizione**

- Scambiando l'ordine delle due istruzioni al passo 2 e al passo 3 si inverte l'ordine con cui sono visualizzati i vertici del cammino

Chiusura transitiva di un grafo orientato

- La chiusura transitiva di un grafo $G = (V, E)$ è un grafo $G^* = (V, E^*)$, dove $(u, v) \in E^*$ se e solo se esiste un cammino p: $u \rightsquigarrow v$ in G





- Warshall arrivò all'algoritmo di Floyd-Warshall proprio con l'obiettivo di risolvere il problema della chiusura transitiva di un grafo orientato

Chiusura transitiva di un grafo orientato

- Per calcolare G^* applichiamo un algoritmo di programmazione dinamica simile a quello di Floyd-Warshall, in cui operiamo la sostituzione «$\min \longrightarrow or$» e «$+ \longrightarrow and$»
- Il processo bottom-up consente la costruzione di una sequenza di matrici quadrate $T^{(k)}$ con la seguente struttura ricorsiva

$$t_{uv}^{(0)} = \begin{cases} 0 & u \neq v \text{ e } (u,v) \notin E(G) \\ 1 & u = v \text{ o } (u,v) \in E(G) \end{cases}$$

e per $k \geq 1$:

$$t_{uv}^{(k)} = t_{uv}^{(k-1)} \vee \left(t_{uk}^{(k-1)} \wedge t_{kv}^{(k-1)} \right)$$

- Interpretando i valori 1 e 0 presenti nelle matrici come i valori logici «vero» e «falso», il significato dell'espressione è chiaro:
  - v è raggiungibile da u mediante un cammino con vertici intermedi in $\{1, ..., k\}$
  - se v è raggiungibile da u con un cammino con vertici intermedi $\{1, ..., k-1\}$
  - oppure se esiste un cammino con vertici intermedi in $\{1, ..., k-1\}$ dal vertice u al vertice k ed esiste un cammino dal vertice k al vertice v con vertici intermedi in $\{1, ..., k-1\}$
- Iterando l'algoritmo di Floyd-Warshall così modificato, si giunge alla matrice $T^{(n)}$ che rappresenta la matrice di adiacenza del grafo G^*

Chiusura transitiva di un grafo orientato

- Algoritmo di Floyd-Warshall modificato per il calcolo della **chiusura transitiva** G^* di un grafo orientato G

Algoritmo 27 CHIUSURATRANSITIVA(G)

Input: Un grafo G orientato

Output: La matrice binaria $T^{(n)}$ di adiacenza per G^*

```
1: per u := 1,2,...,n ripeti
2:     per v := 1,2,...,n ripeti
3:         se u = v o (u,v) ∈ E(G) allora
4:             t_uv^(0) := 1
5:         altrimenti
6:             t_uv^(0) := 0
7:         fine-condizione
8:     fine-ciclo
9: fine-ciclo
10: per k := 1,2,...,n ripeti
11:     per u := 1,2,...,n ripeti
12:         per v := 1,2,...,n ripeti
13:             t_uv^(k) := t_uv^(k-1) ∨ (t_uk^(k-1) ∧ t_kv^(k-1))
14:         fine-ciclo
15:     fine-ciclo
16: fine-ciclo
```

Algoritmi per cammini di costo minimo

- Riepiloghiamo nella seguente tabella gli algoritmi analizzati per il problema del calcolo del cammino di costo minimo su un albero con pesi assegnati agli spigoli

Algoritmo	Sorgente singola	Tutte le coppie
Dijkstra	$O(m + n \log_2 n)$	$O(nm + n^2 \log_2 n)$
Bellman – Ford (base)	$O(nm)$	$O(n^2 m)$
Bellman – Ford (DAG)	$O(n + m)$	$O(n^2 + nm)$
All Pairs Shortest Path		$O(n^4)$
Fast All Pairs Shortest Path		$O(n^3 \log_2 n)$
Floyd – Warshall		$\Theta(n^3)$

Riferimenti bibliografici

- Cormen, Leiserson, Rivest, Stein, «*Introduzione agli algoritmi e strutture dati*», terza edizione, McGraw-Hill (Cap. 24 e 25)
- Alfred Aho, John Hopcroft, Jeffrey Ullman, «*Data Structures and Algorithms*», Addison-Wesley, 1987
- Marco Liverani, «*Dispense del Corso di Ottimizzazione Combinatoria: Cammini di costo minimo – Problemi e algoritmi*» (http://www.mat.uniroma3.it/users/liverani/doc/disp_oc_07.pdf)