Università di Pisa

Pietro Ducange

Algoritmi e strutture dati

a.a. 2019/2020

Grafi II Parte

Si ringrazia la prof. Nicoletta De Francesco per aver messo a disposizione la maggior parte delle slide utilizzate nella presente lezione

algoritmo di Dijkstra

Si applica ai grafi orientati che hanno peso positivo sugli archi

Trova i cammini minimi da un nodo di partenza a tutti gli altri nodi

Basato sulla metodologia greedy

algoritmo di Dijkstra

I cammini minimi, inizialmente posti a “infinito”, vengono aggiornati con stime via via più precise tramite un ciclo nel quale, ad ogni iterazione, un nuovo nodo viene “sistemato”, nel senso che il cammino minimo per lui viene stabilizzato. Alla fine tutti I nodi sono sistemati.

algoritmo di Dijkstra

Utilizza due tabelle dist (distanza) e pred (predecessore) con n elementi (n=numero dei nodi)

Per ogni nodo A, dist (A) contiene in ogni momento la lunghezza di un cammino dal nodo iniziale ad A e pred(A) il predecessore di A in questo cammino.

I nodi sono divisi in due gruppi: quelli già sistemati, per i quali i valori delle tabelle dist e pred sono definitivi, e quelli da sistemare (insieme Q). Per i nodi sistemati, dist contiene la lunghezza del minimo cammino e pred permette di ricostruirlo. Per gli altri, dist e pred contengono dati relativi al cammino trovato fino a quel momento, che eventualmente possono essere cambiati se si trova un cammino più corto.

algoritmo di Dijkstra

Inizialmente nessun nodo è sistemato: Q contiene tutti i nodi.

Si esegue poi un ciclo. Ad ogni passo

1. **si considera "sistemato" il nodo, fra quelli di Q, con dist minore e lo si toglie da Q;**
2. **si aggiornano pred e dist per gli immediati successori di questo nodo;**

Il ciclo termina quando Q contiene un solo nodo.

algoritmo di Dijkstra

```
1    Q = N;
2    per ogni nodo p diverso da p0 {              // O(n)
         dist(p)=infinito, pred(p)=vuoto;
     }
     dist(p0)=0;
```

algoritmo di Dijkstra

```
4   while (Q contiene più di un nodo) {
5       estrai da Q il nodo p con minima dist(p);   // O(logn)
6       per ogni nodo q successore di p {
            lpq=lunghezza dell'arco (p,q);
            if (dist(p)+lpq < dist(q)) {            //O(1)
                    dist(q)=dist(p)+lpq;    //O(1)
                    pred(q)=p;              //O(1)
7                   re-inserisci in Q il nodo q modificato; // O(logn)

  }
   }
```

L'insieme Q è memorizzato in un min-heap. Di conseguenza i comandi

5 estrai da Q il nodo p con minima dist(p);

7 re-inserisci in Q il nodo q modificato;

hanno complessità O(logn)

Numero iterazioni del ciclo while : n

Complessitá iterazione: C[5]+ m/n C[7]

= O(logn + (m/n) logn)

Complessità del ciclo: O(n(logn + (m/n) logn))=

O(nlogn + mlogn)

Considerazioni

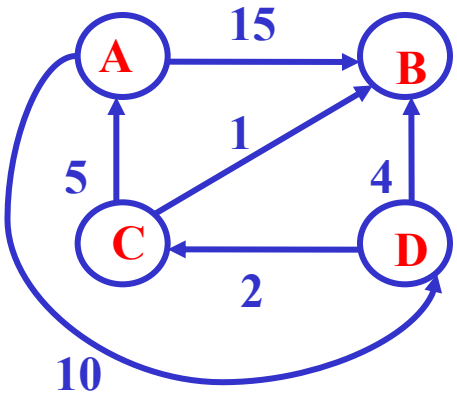
Ad ogni iterazione del ciclo i nodi già scelti (eliminati da Q) sono "sistemati":

per i nodi già scelti dist contiene la lunghezza del cammino minimo e pred permette di ricostruirlo.

Il cammino minimo per i nodi già scelti passa soltanto da nodi già scelti

esempio

$Q = \{A, B, C, D\}$



dist/pred

A	B	C	D
0 | -	inf | -	inf | -	inf | -

estraggo A: dist(A)=0

dist(A)+ |(A,B)| < dist(B))
0 + 15 < inf.

→ dist(B)=15,pred(B)=A

dist(A)+ |(A,D)| < dist(D))
0 + 10 < inf.

→ dist(D)=10, pred(B)=A

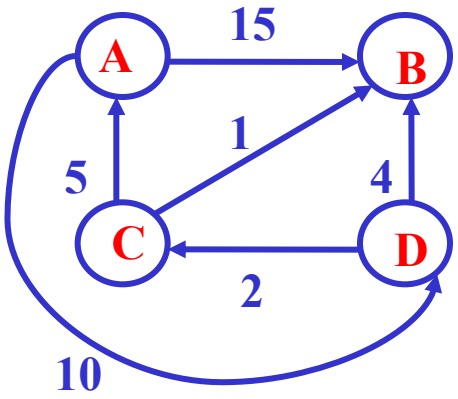
A	B	C	D
0 | -	15 | A	inf | -	10 | A

$Q = \{B, C, D\}$

dist/pred

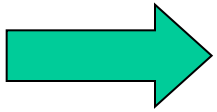
A	B	C	D
0 | -	15 | A	inf | -	10 | A

Q = { B, C, D}



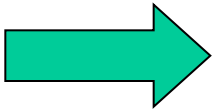
estraggo D: dist(D)=10

dist(D)+ |(D,B)| < dist(B))
10 + 4 < 15



dist(B)=14, pred(B)=D

dist(D)+ |(D,C)| < dist(C))
10 + 2 < inf.



dist(C)=12, pred(C)=D

A	B	C	D
0 | -	14 | D	12 | D	10 | A

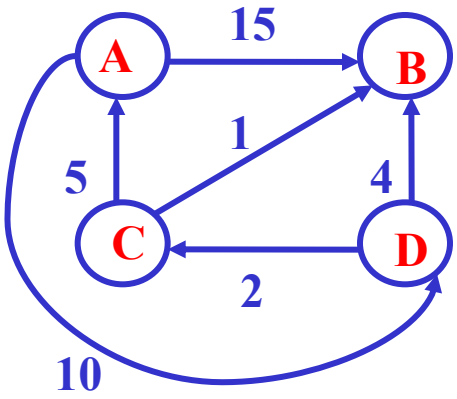
Q = {B, C}

esempio

dist/pred

A	B	C	D
0 | -	14 | D	12 | D	10 | A

Q = { B, C}



estraggo C: dist(C)=12

dist(C)+ |(C,B)| < dist(B))
12 + 1 < 14

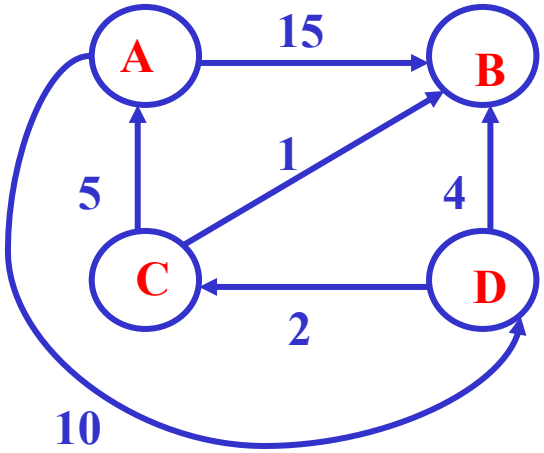
dist(B)=13, pred(B)=C

A	B	C	D
0 | -	13 | C	12 | D	10 | A

Q = {B}

soluzione

A	B	C	D
0 | -	13 | C	12 | D	10 | A



da A a B: A->D->C ->B lung=13

da A a C: A->D->C lung=12

da A a D: A->D lung=10

Nodo scelto	Q
	A, B, C, D
A	B, C, D
D	B, C
C	B

A	B	C	D
0 /-	i /-	i /-	i/-
0 /-	15/A	i /-	10/A
0 /-	14/D	12/D	10/A
0/-	13/C	12/D	10/A

dist/pred

Bibliografia

Demetrescu:
Capitolo 14.5

Cormen:
Paragrafo 24.4

Algoritmo PageRank di Google (cenni)

- **Serve al motore di ricerca per trovare le pagine web di interesse per una interrogazione**
- **Si basa sulle connessioni (link) fra le pagine**
- **Considera la rete come un grafo in cui le pagine sono i nodi e i link sono gli archi**

Algoritmo PageRank di Google

- **Calcola il «rango» (rilevanza) di una pagina web P : R(P)**
- **La rilevanza di P dipende da quanto sono rilevanti le pagine che puntano a P (hanno un link verso P) ma anche da quanti link escono da queste pagine**

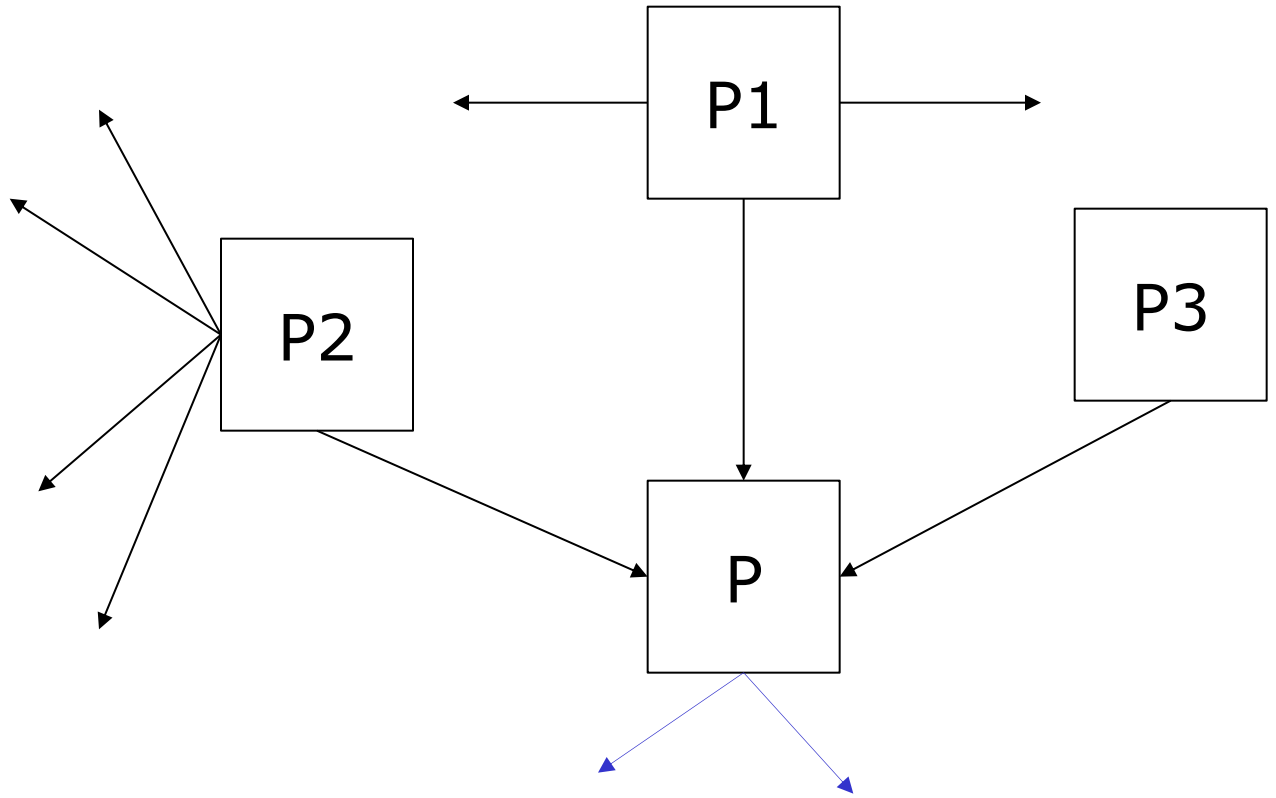
Algoritmo PageRank: formula base

$$R(P) = \sum_{Q->P} \frac{R(Q)}{|Q|}$$

|Q| = numero di link uscenti da Q

Q->P: link da Q a P

Algoritmo PageRank: esempio



R(P1)= 10
R(P2)= 10
R(P3)= 6

R(P)= 10/3 + 10/5 + 6/1= 11,3

Algoritmo PageRank

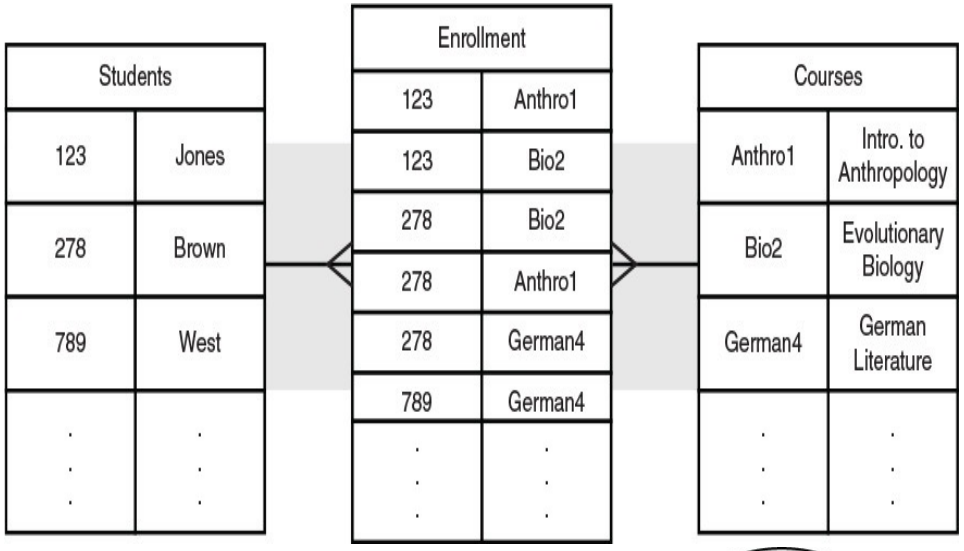
- **Il calcolo viene fatto calcolando il rango dei nodi in modo iterativo utilizzando la matrice di adiacenza e partendo da un valore del rango uguale per tutti i nodi**

- **Sono necessari aggiustamenti della formula per assicurarne la convergenza (cicli, nodi pozzo)**

Graph Databases

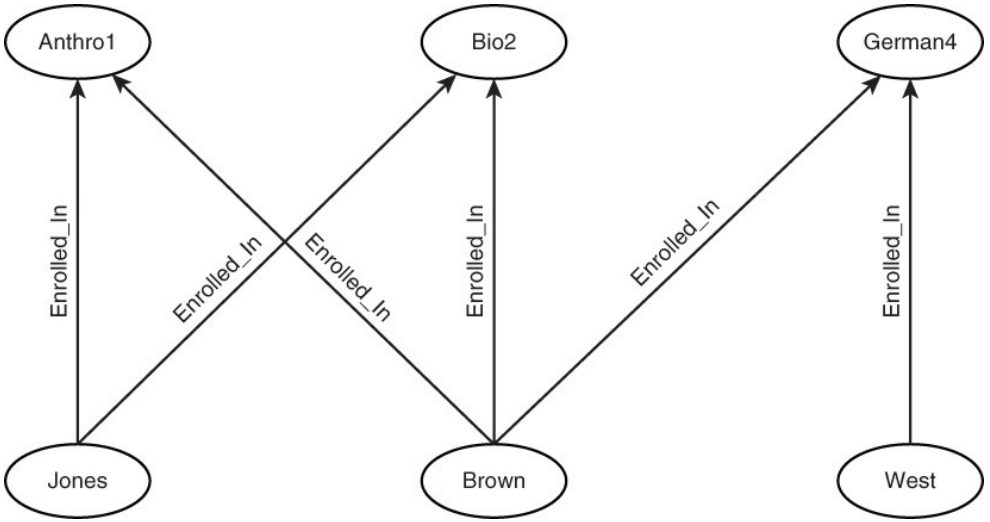
- Adottano vertici e archi per memorizzare relazioni esplicite tra entità.
- Nei database relazionali, le connessioni non sono rappresentate come collegamenti. Invece, due entità condividono un valore di attributo comune, che è noto come una chiave.
- Le operazioni di join sono usate nei database relazionali per trovare connessioni o collegamenti.
- Quando si ha a che fare con un'enorme quantità di dati (diverse tabelle con troppe righe) le operazioni di join diventano computazionalmente costose.

Graph Databases vs Relational Databases

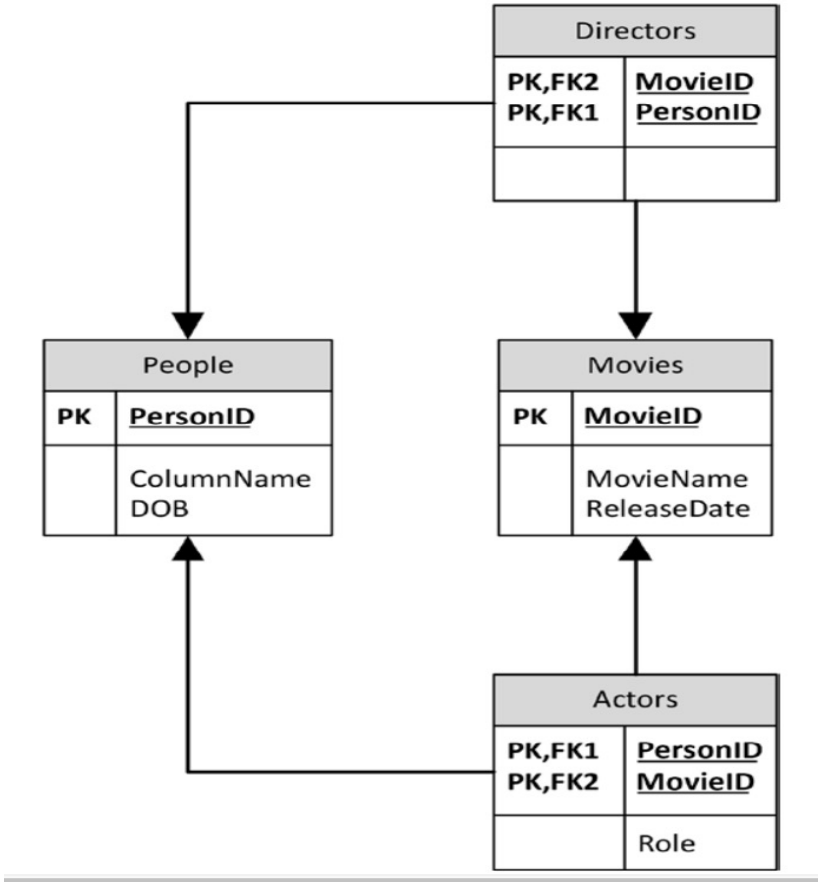


Esempio di query: elencare tutti i corsi a cui è iscritto un particolare studente.

Gli archi tra studenti e corsi ci permettono di interrogare rapidamente i database.



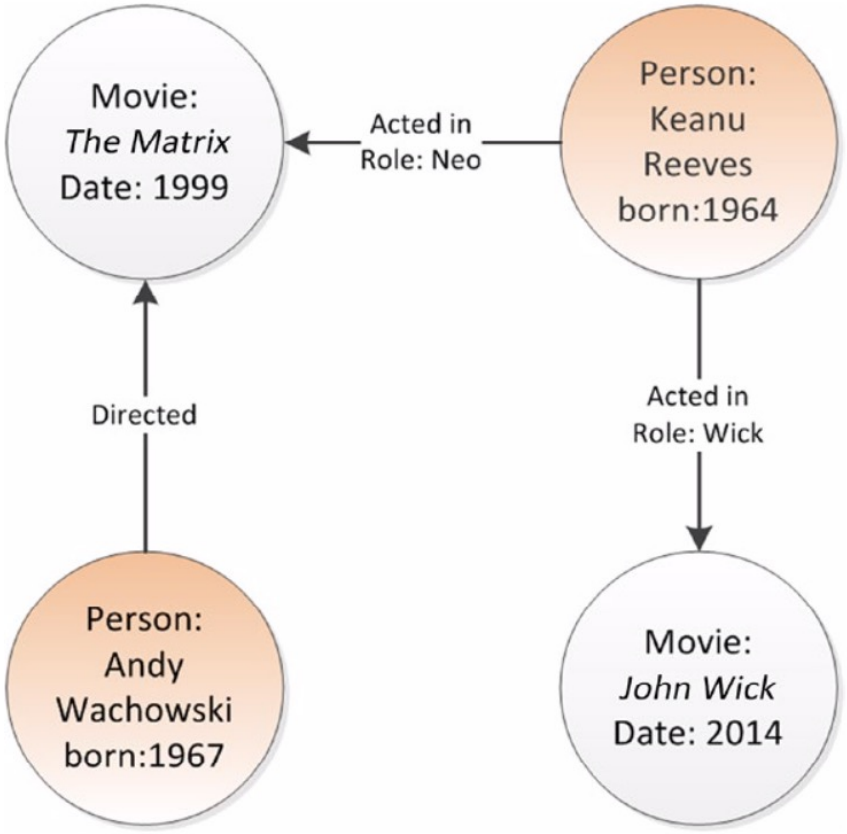
HollywoodDB: Modello Relazionale



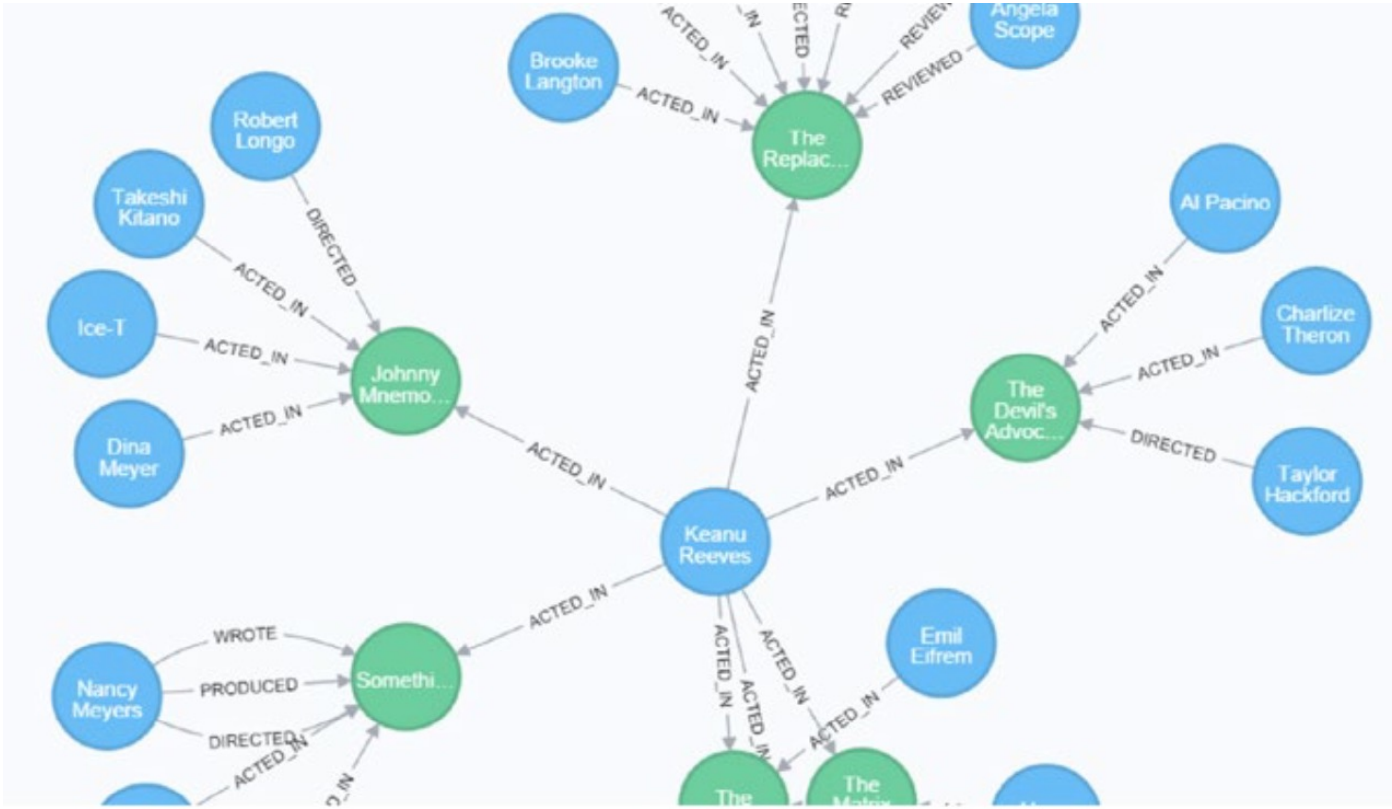

```
1  SELECT p2.personname, m1.movieName
2    FROM people p1
3         JOIN actors a1 ON (p1.personid = a1.personid)
4         JOIN movies m1 ON (a1.movieid = m1.movieid)
5         JOIN actors a2 ON (a2.movieid = m1.movieid)
6         JOIN people p2 ON (p2.personid = a2.personid)
7   WHERE p1.personname = 'Keanu Reeves';
```

La query restituisce tutti gli attori che hanno lavorato con Keanu Reeve (e anche il film)

HollywoodDB: Graph Model

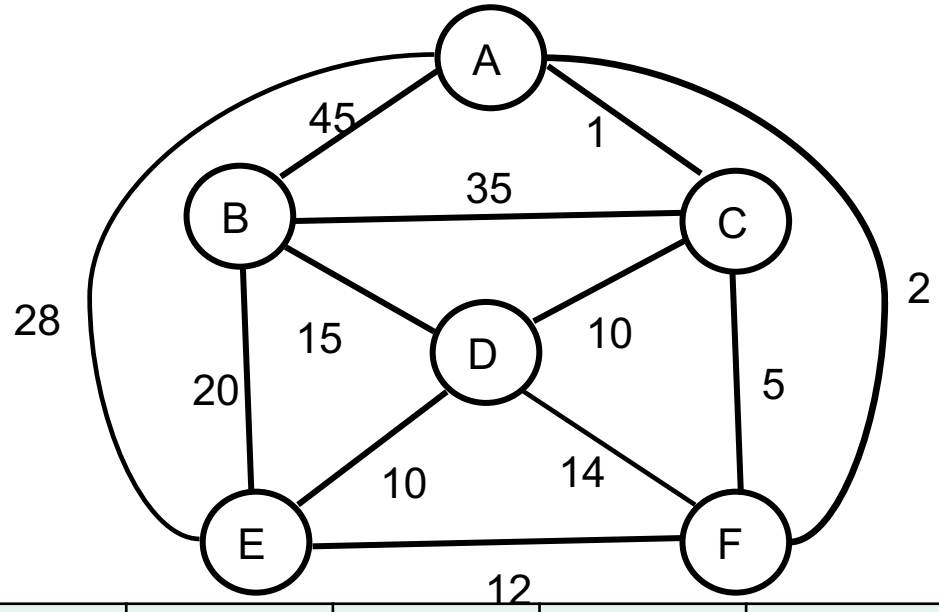


HollywoodDB: Graph Model




```
neo4j-sh (?)$ MATCH (kenau:Person {name:"Keanu Reeves"})
                    -[:ACTED_IN]->(movie)<-[:ACTED_IN]-(coStar)
              RETURN coStar.name;
```

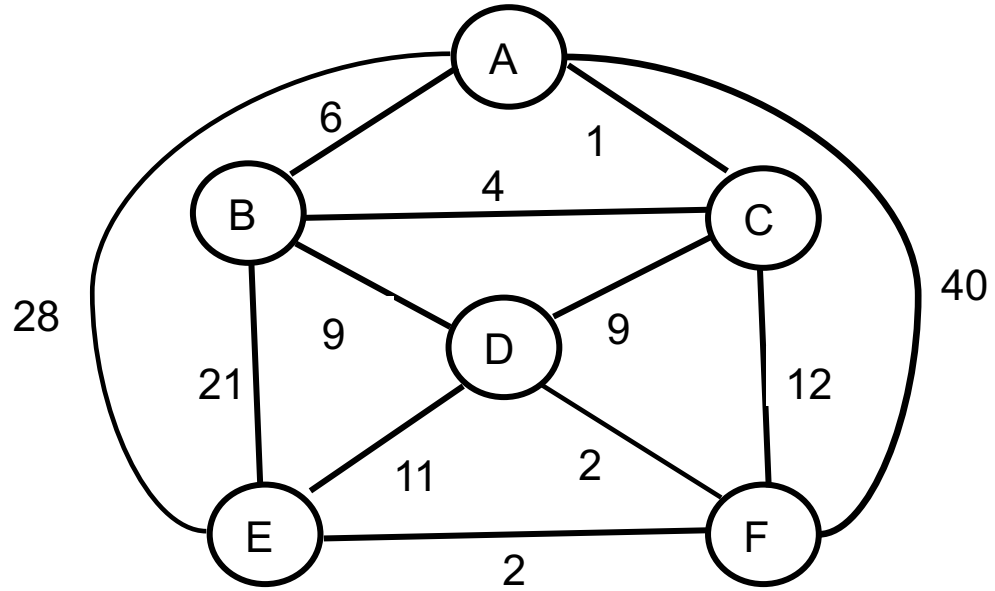
1. Applicare l'algoritmo di Dijkstra al grafo seguente con nodo di partenza F



Q	Nodo scelto	A	B	C	D	E	F
A, B, C, D, E, F		inf | -	inf | -	inf | -	inf | -	inf | -	0 | -
							0 | -
							0 | -
							0 | -
							0 | -
							0 | -

Cammini minimi: **FA (2)**, **FACDB (28)**, **FAC (3)**, **FACD (13), FE (12)**

2. Applicare l'algoritmo di Dijkstra al grafo seguente con nodo di partenza A



Q	Nodo scelto	A	B	C	D	E	F
A, B, C, D, E, F		0 | -	inf | -	inf | -	inf | -	inf | -	Inf | -
		0 | -					
		0 | -					
		0 | -					
		0 | -					
		0 | -					

Cammini minimi: **ACB (5)**, **AC (1)**, AC**D (10)**, ACDF**E (14), ACDF (12)**