PROGRAMMAZIONE DINAMICA
DISTANZA DI LEVENSHTEIN
ALGORITMO DI FLOYD-WARSHALL

INFORMATICA

EDIT DISTANCE

DISTANZA DI LEVENSHTEIN

Solo sostituzioni per $n = m$
=> HAMMING DISTANCE

- La distanza di Levenshtein serve a quantificare quanto due stringhe siano diverse
- Abbiamo due sequenze $X = \langle x_1, \ldots, x_m \rangle$ e $Y = \langle y_1, \ldots, y_n \rangle$ e ci chiediamo quale sia il numero minimo di operazioni di **inserimento**, **rimozione** e **sostituzione** necessarie per trasformare X in Y.
- Utile, per esempio, per trovare la correzione automatica (i.e., quale è la parola più simile a quella che abbiamo digitato?)

DISTANZA DI LEVENSHTEIN: OPERAZIONI

- Inserimento di un carattere z in posizione i nella sequenza $X = \langle x_1, \ldots, x_m \rangle$: otteniamo $X' = \langle x_1, \ldots x_{i-1}, z, x_i, \ldots, x_m \rangle$ $|X'| = |X| + 1$
- Inserimento di "S" in posizione 3 di "CASA": "CASSA"
- Cancellazione del carattere in posizione i nella sequenza $X = \langle x_1, \ldots, x_m \rangle$: otteniamo $X' = \langle x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_m \rangle$ $m-1$
- Cancellazione del carattere in posizione 1 di "CASA": "ASA"

DISTANZA DI LEVENSHTEIN: OPERAZIONI

- Sostituzione del carattere in posizione i con il carattere z nella sequenza $X = \langle x_1, \ldots, x_m \rangle$: otteniamo $X' = \langle x_1, \ldots, x_{i-1}, z, x_{i+1}, \ldots, x_m \rangle$
- Sostituzione del carattere in posizione 4 con "E" in "CASA": "CASE"

DISTANZA DI LEVENSHTEIN

- Come possiamo approcciare il problema di trovare il minimo numero di operazioni?
- Ci riconduciamo a sotto-problemi più piccoli
- E vediamo come comporre le soluzioni ottime di problemi più piccoli in modo da formare la soluzione
- Iniziamo cercando dei casi base

DISTANZA DI LEVENSHTEIN: CASI BASE

X_i

Y_j

$d(X_i, Y_j) =$

$d(X_{i-1}, Y_j)$

$d(X_i, Y_{j-1})$

$d(X_{i-1}, Y_{j-1})$

- Se $X = \langle x_1, \ldots, x_m \rangle$ e $Y = \langle \rangle$ allora dobbiamo cancellare tutti i caratteri di X per ottenere Y: servono m operazioni
- Se $X = \langle \rangle$ e $Y = \langle y_1, \ldots, y_n \rangle$ dobbiamo inserire in X tutti i caratteri di Y: servono n operazioni
- Quindi se una delle due sequenze è vuota ci servono tante operazioni di inserimento o cancellazione quanto è la lunghezza dell'altra sequenza

DISTANZA DI LEVENSHTEIN: SOTTO-STRUTTURA OTTIMA (1)

- Se abbiamo le due sequenze $X = \langle x_1, \ldots, x_m \rangle$ e $Y = \langle y_1, \ldots, y_n \rangle$, come possiamo agire guardano l'ultimo elemento?

- Possiamo sostituire x_m con y_n se $x_m \neq y_n$. Dopo questa operazione le due sequenze coincidono sull'ultimo elemento e dobbiamo trovare quante operazioni svolgere per trasformare $X' = \underbrace{\langle x_1, \ldots, x_{m-1} \rangle}$ in $Y' = \underbrace{\langle y_1, \ldots, y_{n-1} \rangle}$

DISTANZA DI LEVENSHTEIN: SOTTO-STRUTTURA OTTIMA (2)

- Se abbiamo le due sequenze $X = \langle x_1, \ldots x_m, \rangle$ e $Y = \langle y_1, \ldots, y_n \rangle$, come possiamo agire guardando l'ultimo elemento?

- Se $x_m = y_n$ non svolgiamo alcuna operazione e dobbiamo trovare quante operazioni svolgere per trasformare $X' = \langle x_1, \ldots, x_{m-1} \rangle$ in $Y' = \langle y_1, \ldots, y_{n-1} \rangle$

DISTANZA DI LEVENSHTEIN: SOTTO-STRUTTURA OTTIMA (3)

- Se abbiamo le due sequenze $X = \langle x_1, \ldots, x_m \rangle$ e $Y = \langle y_1, \ldots, y_n \rangle$, come possiamo agire guardando l'ultimo elemento?

- Possiamo decidere di inserire y_n come ultimo carattere di X e quindi chiederci il numero minimo di operazioni da svolgere per trasformare X in $Y' = \langle y_1, \ldots, y_{n-1} \rangle$

DISTANZA DI LEVENSHTEIN: SOTTO-STRUTTURA OTTIMA (4)

- Se abbiamo le due sequenze $X = \langle x_1, \ldots, x_m \rangle$ e $Y = \langle y_1, \ldots, y_n \rangle$, come possiamo agire guardando l'ultimo elemento?

- Possiamo decidere di cancellare l'ultimo carattere di X e quindi chiederci il numero minimo di operazioni da svolgere per trasformare $X' = \langle x_1, \ldots, x_{m-1} \rangle$ in Y

DISTANZA DI LEVENSHTEIN

- Abbiamo coperto tutti i possibili casi con cui possiamo applicare una qualche operazione per far coincidere le due stringhe sull'ultimo elemento
- Una volta fatto questo ci ritroviamo un problema già semplice da risolvere:
  - O entrambe le sequenze sono più corte di un carattere
  - O almeno una è più corta di un carattere
- Vediamo ora come calcolare il costo minimo

DISTANZA DI LEVENSHTEIN

- Indichiamo con $c_{i,j}$ il costo minimo per trasformare la sotto-sequenza $X' = \langle x_1, \ldots, x_i \rangle$ in $Y' = \langle y_1, \ldots, y_j \rangle$
- Se $i = 0$ o $j = 0$ siamo in uno dei casi base:
  - $c_{i,0} = i$
  - $c_{0,j} = j$

DISTANZA DI LEVENSHTEIN

- Se $i \neq 0$ e $j \neq 0$ allora abbiamo che dobbiamo scegliere il costo minimo tra tutte le operazioni che possiamo effettuare:
  - $1 + c_{i-1,j-1}$ se effettuiamo una sostituzione e $x_i \neq y_j$
  - $c_{i-1,j-1}$ se effettuiamo una sostituzione e $x_i = y_j$
  - $1 + c_{i-1,j}$ se cancelliamo l'ultimo carattere di X'
  - $1 + c_{i,j-1}$ se inseriamo l'ultimo carattere di Y'

DISTANZA DI LEVENSHTEIN

- Abbiamo quindi che

$$c_{i,j} = \begin{cases} \max(i,j) & \text{se } i = 0 \text{ o } j = 0 \\ \min(1 + c_{i-1,j-1}, 1 + c_{i,j-1}, 1 + c_{i-1,j}) & \text{se } i,j \neq 0 \text{ e } x_i \neq y_j \\ \min(c_{i-1,j-1}, 1 + c_{i,j-1}, 1 + c_{i-1,j}) & \text{se } i,j \neq 0 \text{ e } x_i = y_j \end{cases}$$

- Questo ci permette di creare una tabella di $m+1$ righe e $n+1$ colonne che riempiremo coi valori di $c_{i,j}$

PSEUDOCODICE: DISTANZA DI LEVENSHTEIN

```
Parametri: sequenze X e Y di lunghezza m e n
c = matrice (m+1)×(n+1) # la matrice che conterrà tutti i c_{i,j}
for i in range(0, m+1): # primo caso base, la sequenza Y è vuota
    c[i][0] = i
for j in range(0, n+1): # secondo caso base, la sequenza X è vuota
    c[0][j] = j
for i in range(1, m+1):
    for j in range(1, n+1):
        cancellazione = 1 + c[i−1][j]
        inserimento = 1 + c[i][j−1]
        sostituzione = c[i−1][j−1]
        if X[i] ≠ Y[j]: # la sostituzione è effettuata solo se x_i non coincide con y_j
            sostituzione = sostituzione + 1
        c[i][j] = min(sostituzione, inserimento, cancellazione)
return c[m][n]
```

DISTANZA DI LEVENSHTEIN

			A	B	A	C	O
		0	1	2	3	4	5
	0	0	1	2	3	4	5
A	1	1					
B	2	2					
B	3	3					
A	4	4					

Casi base

Casi base

DISTANZA DI LEVENSHTEIN

			A	B	A	C	O
		0	1	2	3	4	5
	0	0	1	2	3	4	5
A	1	1	0				
B	2	2					
B	3	3					
A	4	4					

Dobbiamo fare il minimo tra
1+C[0][1]
1+C[1][0]
C[0][0]

Questo perché $x_1 = y_1$

DISTANZA DI LEVENSHTEIN

			A	B	A	C	O
		0	1	2	3	4	5
	0	0	1	2	3	4	5
A	1	1	0	1			
B	2	2					
B	3	3					
A	4	4					

Dobbiamo fare il minimo tra
1+C[0][2]
1+C[1][1]
1+C[0][1]

Questo perché $x_1 \neq y_2$

DISTANZA DI LEVENSHTEIN

A B A C O
A B _ B A

			A	B	A	C	O
		0	1	2	3	4	5
	0	0	1	2	3	4	5
A	1	1	0	1	2	3	4
B	2	2	1	0	1	2	3
B	3	3	2	1	1	2	3
A	4	4	3	2	1	2	3

Distanza di Levenshtein tra "ABBA" e "ABACO"

DISTANZA DI LEVENSHTEIN: COMPLESSITÀ

- Dobbiamo calcolare tutti i valori contenuti in una tabella di dimensione $(m+1) \times (n+1)$, quindi $O(mn)$ valori
- Riempire ogni valore richiede solamente un numero costante di operazioni, quindi il costo totale è $O(mn)$

DISTANZA DI LEVENSHTEIN: VARIANTI

- Sfruttando la stessa idea possiamo definire molteplici varianti dello stesso algoritmo
- Possiamo dare costi diversi a seconda dell'operazione, per esempio w_{del}, w_{ins} e w_{sub} per cancellazione, inserimento e sostituzione. In quel caso è sufficiente cambiare lievemente la definizione di $c_{i,j}$:

$$c_{i,j} = \begin{cases} \max(w_{\text{del}} i, w_{\text{ins}} j) & \text{se } i = 0 \text{ o } j = 0 \\ \min(w_{\text{sub}} + c_{i-1,j-1}, w_{\text{ins}} + c_{i,j-1}, w_{\text{del}} + c_{i-1,j}) & \text{se } i, j \neq 0 \text{ e } x_i \neq y_j \\ \min(c_{i-1,j-1}, w_{\text{ins}} + c_{i,j-1}, w_{\text{del}} + c_{i-1,j}) & \text{se } i, j \neq 0 \text{ e } x_i = y_j \end{cases}$$

ALGORITMO DI FLOYD-WARSHALL

- L'algoritmo di Floyd-Warshall è un algoritmo per trovare il percorso più corto tra ogni coppia di nodi in un grafo
- Ovvero, dato un grafo $G = (V, E)$ avremo come risultato una tabella D in cui in posizione (i, j) è la lunghezza del percorso di peso/costo minimo tra i e j.
- Il grafo può essere orientato, pesato e anche con pesi negativi sugli archi

ALGORITMO DI FLOYD-WARSHALL

- Supponiamo che i nodi corrispondano agli interi $\boxed{\{0,\ldots,k\}}$
- Proviamo a definire in modo ricorsivo il percorso tra due nodi i e j
- Per fare questo consideriamo una restrizione sui nodi intermedi che possono essere contenuti nel percorso tra i e j
- Indichiamo con $d^k_{i,j}$ la lunghezza del percorso più corto tra i e j che contenga nodi intermedi solo nodi in $\{0,\ldots,k-1\}$

$d^n_{i,j}$

$|V| = n$

$1, \longrightarrow n$

$\boxed{d^{K}_{i,j}}$

$K = 0$

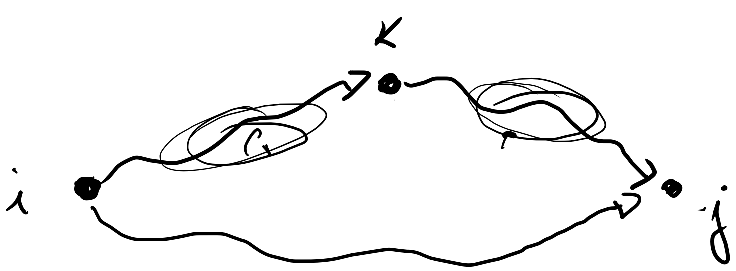
$\Downarrow$

no vertici intermedi nel cammino minimo

$i \longrightarrow j \qquad d^{0}_{i,j} = \begin{cases} w_{i,j} & , \quad (i,j) \in E \\ \infty & , \quad (i,j) \notin E \end{cases}$

$K > 0$

$d^{K}_{i,j}$



Se $i \to K \to j$ allora $d^{K}_{i,j} = \underbrace{d^{K-1}_{i,K} + d^{K-1}_{K,j}}$

altrimenti $d^{K}_{i,j} = d^{K-1}_{i,j}$

$$d^{K}_{i,j} = \min\left\{ d^{K-1}_{i,j} \, , \; d^{K-1}_{i,K} + d^{K-1}_{K,j} \right\}$$

ALGORITMO DI FLOYD-WARSHALL

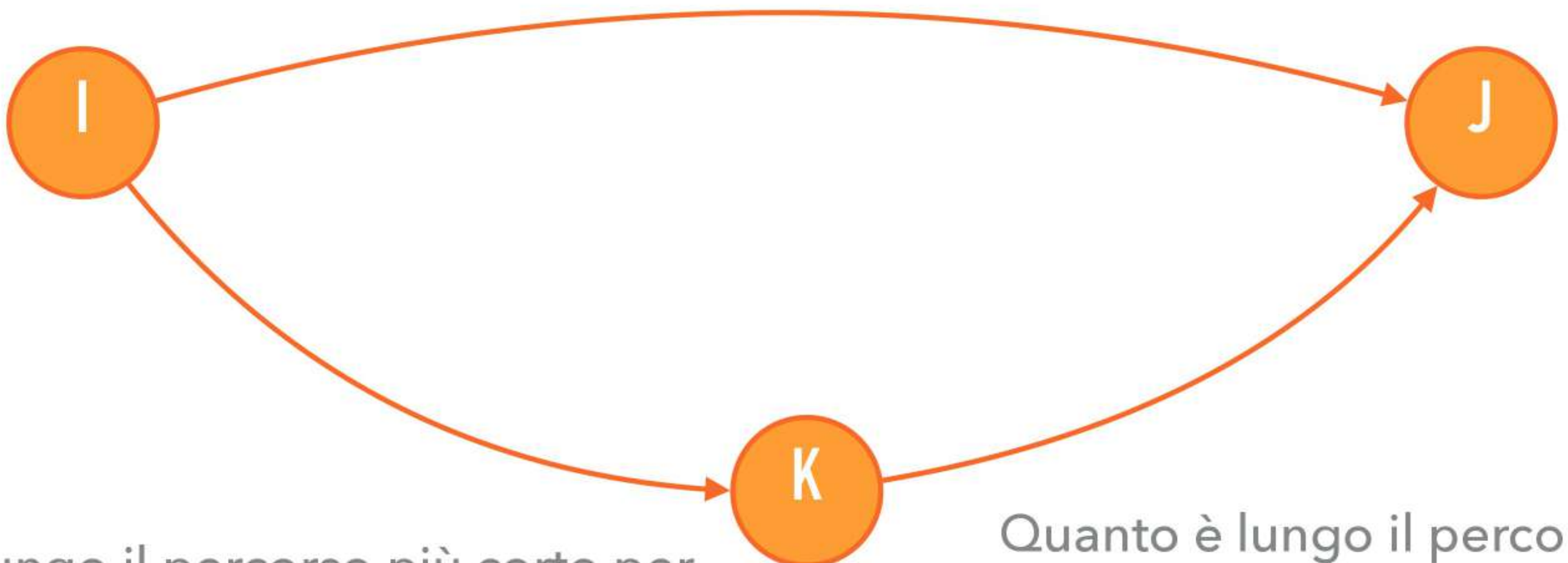
- Il caso base è quando non possiamo avere nodi intermedi, quindi $d^0_{i,j}$.
- Dato che non possiamo avere nodi intermedi ci sono solo tre possibilità:
  - $d^0_{i,i} = 0$, dato che nodo di destinazione e partenza sono lo stesso
  - $d^0_{i,j} = w_{i,j}$ se esiste un arco tra i e j
  - $d^0_{i,j} = +\infty$ se nessun arco connette i con j

ALGORITMO DI FLOYD-WARSHALL

- Per un generico $d_{i,j}^{k+1}$ abbiamo due casi:
  - Il nodo k non viene usato nel percorso di lunghezza minima per andare da i a j: in quel caso $d_{i,j}^{k+1} = d_{i,j}^{k}$
  - Il nodo k viene usato nel percorso di lunghezza minima per andare da i a j. Dato che un nodo può apparire una sola volta nel percorso non useremo k per andare da i a k e per andare da k a j, quindi $d_{i,j}^{k+1} = d_{i,k}^{k} + d_{k,j}^{k}$

ALGORITMO DI FLOYD-WARSHALL

Quanto è lungo il percorso più corto per andare da i a j usando solo $\{0,\ldots k\}$ come nodi intermedi e senza passare per k? $d^k_{i,j}$



Quanto è lungo il percorso più corto per andare da i a k usando solo $\{0,\ldots k\}$ come nodi intermedi? $d^k_{i,k}$

Quanto è lungo il percorso più corto per andare da k a j usando solo $\{0,\ldots k\}$ come nodi intermedi? $d^k_{k,j}$

Totale: $d^k_{i,k} + d^k_{k,j}$

ALGORITMO DI FLOYD-WARSHALL

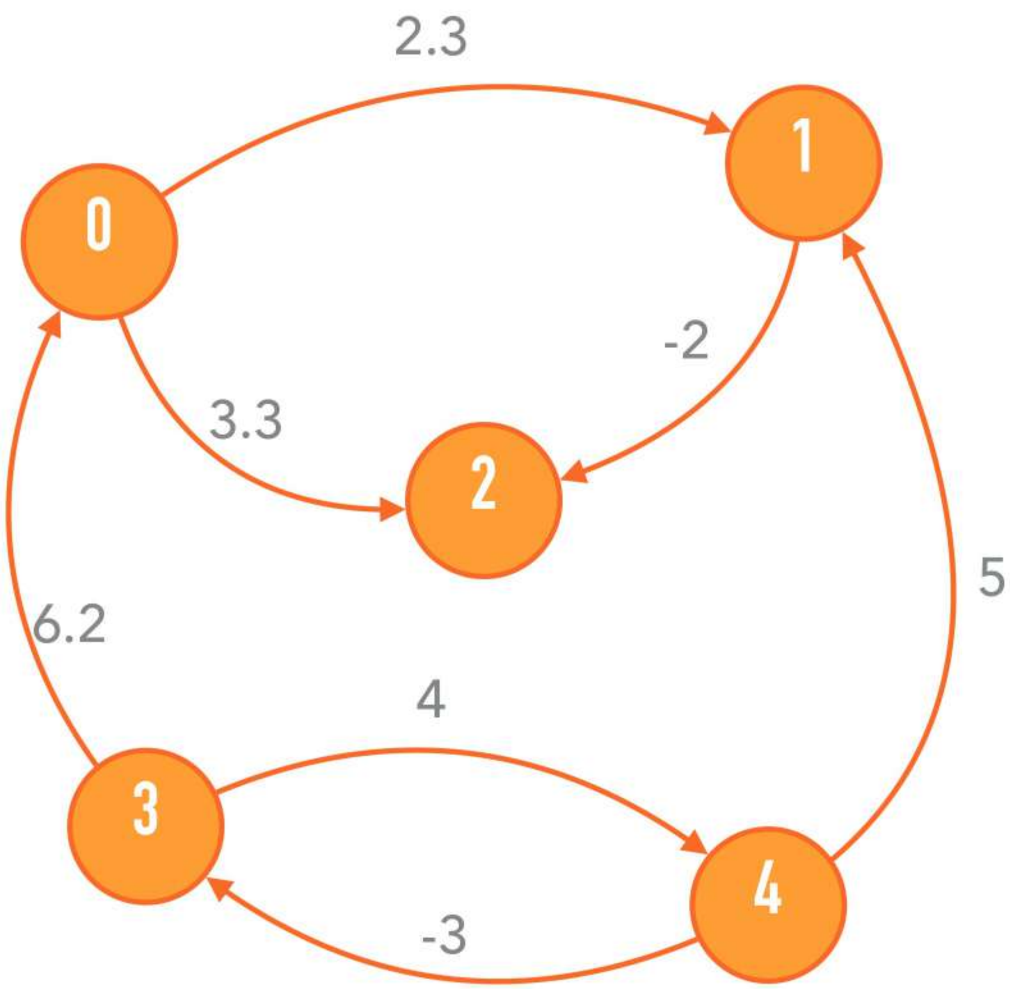
- Abbiamo quindi che $d_{i,j}^{k+1} = \min\{d_{i,j}^{k}, d_{i,k}^{k} + d_{k,j}^{k}\}$ dato che abbiamo solo due scelte possibili e vogliamo il percorso di lunghezza/peso minimo
- Possiamo iniziare da $k = 0$, in cui conosciamo tutti i valori, fino ad arrivare a $k = n$, in cui possiamo utilizzare tutti gli n vertici del grafo come nodi intermedi

PSEUDOCODICE: FLOYD-WARSHALL

Parametri: Matrice di adiacenza del grafo W di n nodi con $+\infty$ dove gli archi sono assenti

```
D = array di dimensione (n + 1) × n × n # contiene tutti i d^k_{i,j}
D[0] = W # la matrice W contiene già i casi base
for k in range(0,n):
    # calcoliamo d^{k+1}_{i,j} al variare di i e j
    for i in range(0,n):
        for j in range(0,n):
            D[k+1][i][j] = min(D[k][i][j], D[k][i][k] + D[k][k][j])
return D[n]
```

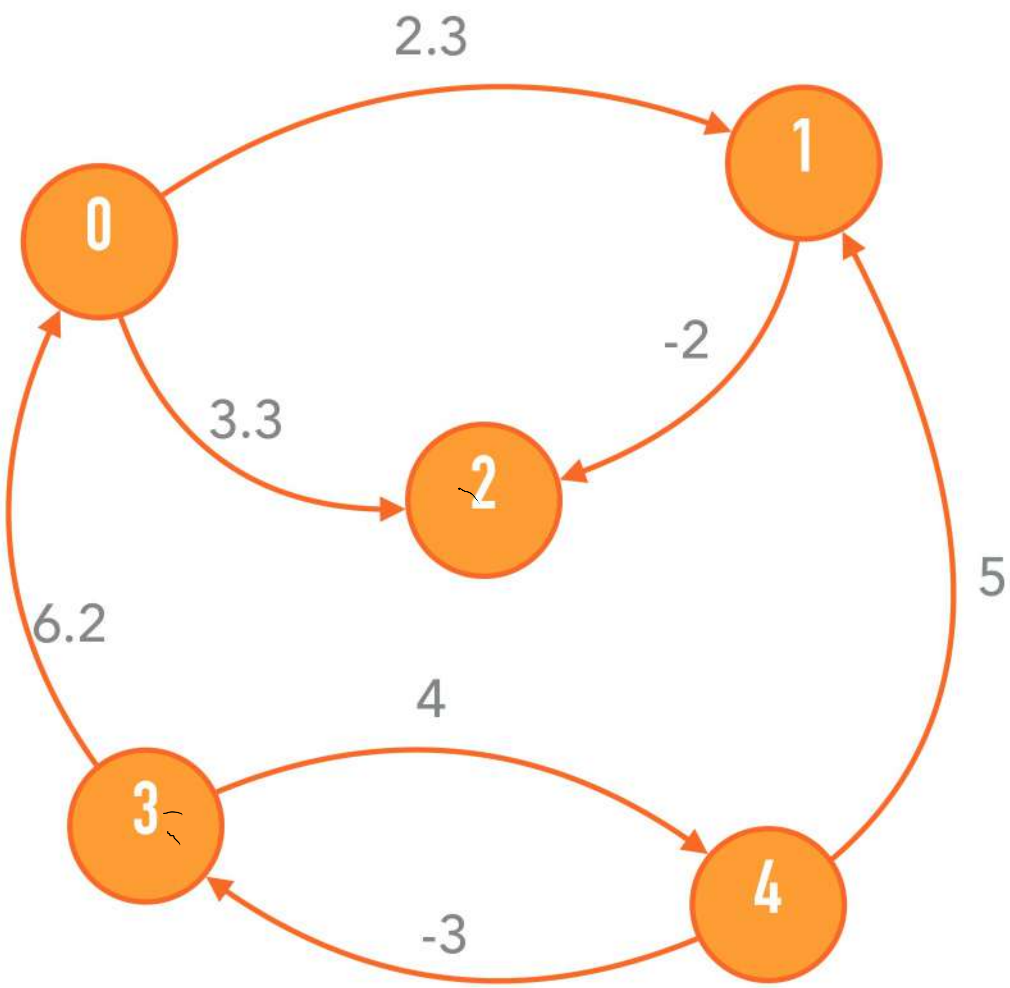
ALGORITMO DI FLOYD-WARSHALL



W=D[0]=

0	2.3	3.3	∞	∞
∞	0	-2	∞	∞
∞	∞	0	∞	∞
6.2	∞	∞	0	4
∞	5	∞	-3	0

ALGORITMO DI FLOYD-WARSHALL



W=D[0]=

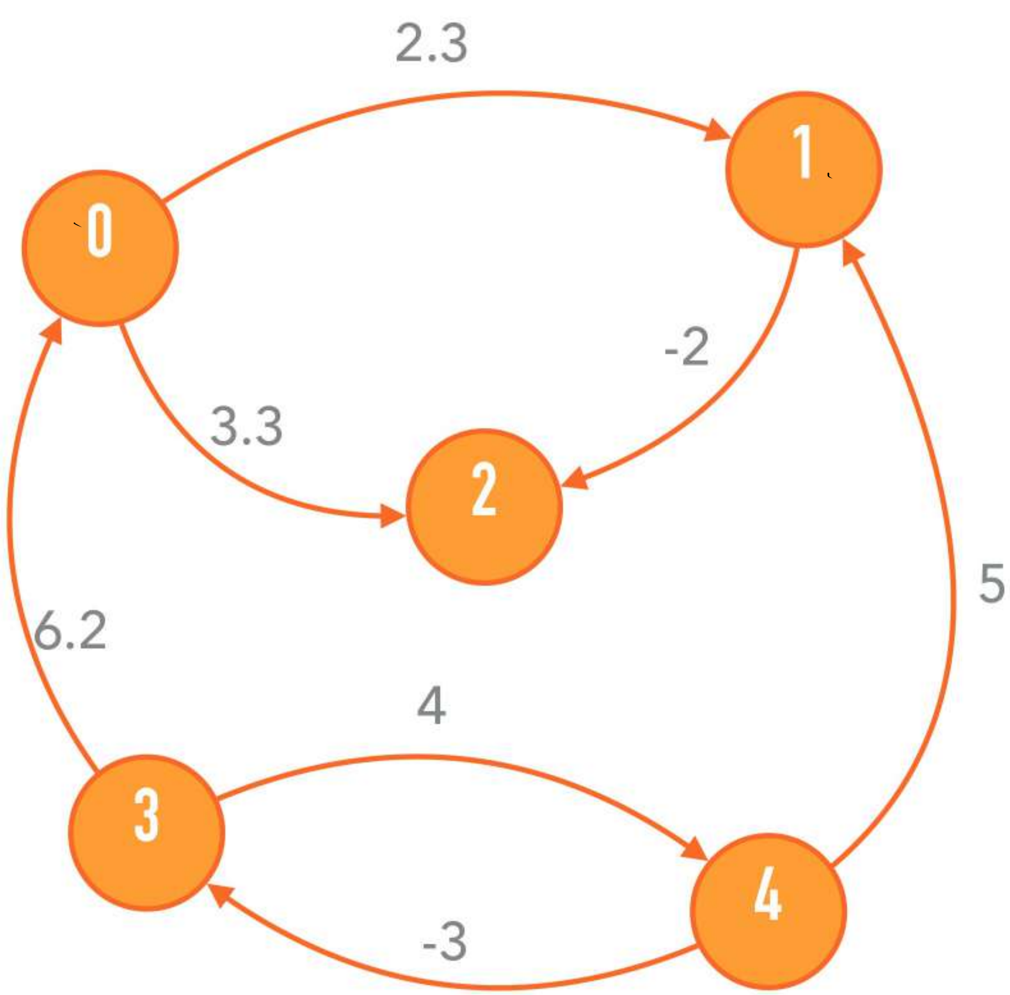
0	2.3	3.3	∞	∞
∞	0	-2	∞	∞
∞	∞	0	∞	∞
6.2	∞	∞	0	4
∞	5	∞	-3	0

D[1]=

0	2.3	3.3	∞	∞
∞	0	-2	∞	∞
∞	∞	0	∞	∞
6.2	8.5	9.5	0	4
∞	5	∞	-3	0

Possiamo passare per il nodo 0

ALGORITMO DI FLOYD-WARSHALL



D[1]=

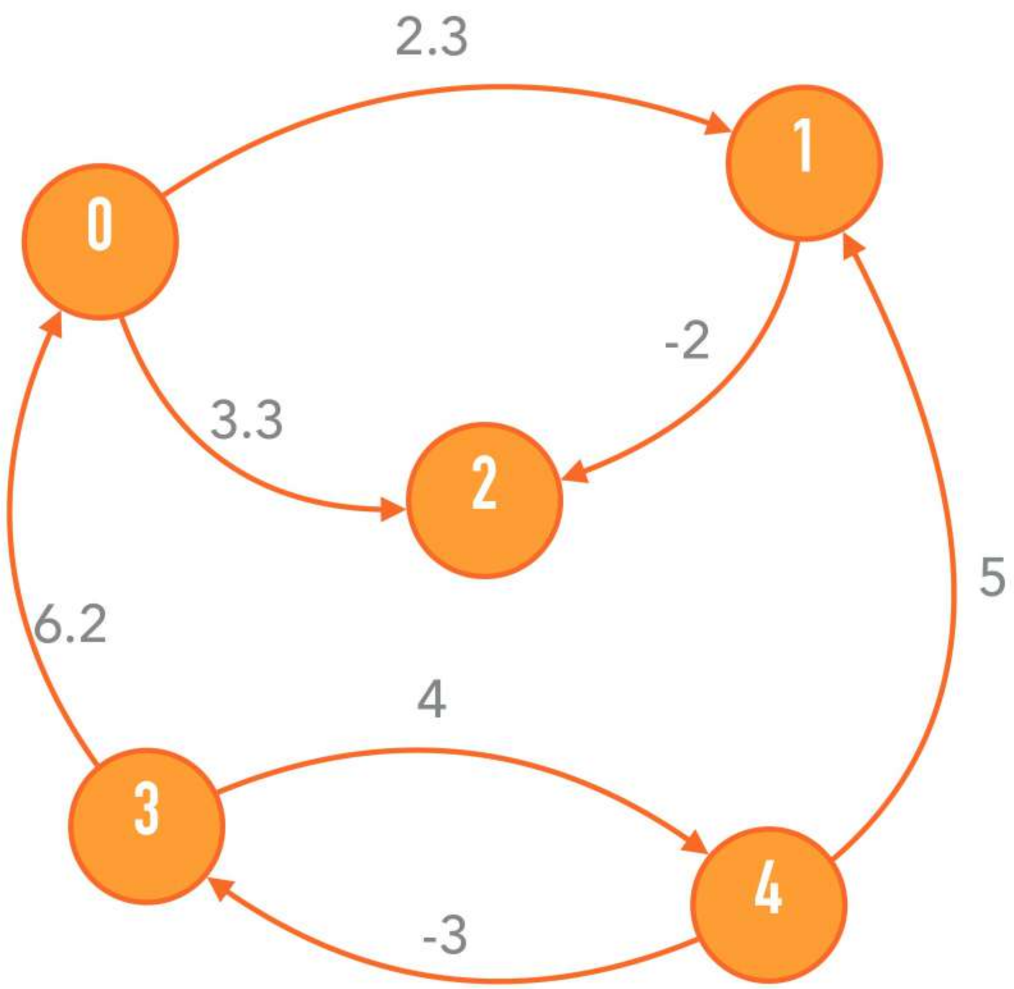
0	2.3	3.3	∞	∞
∞	0	-2	∞	∞
∞	∞	0	∞	∞
6.2	8.5	9.5	0	4
∞	5	∞	-3	0

D[2]=

0	2.3	0.3	∞	∞
∞	0	-2	∞	∞
∞	∞	0	∞	∞
6.2	8.5	6.5	0	4
∞	5	3	-3	0

Possiamo passare per i nodi 0 e 1

ALGORITMO DI FLOYD-WARSHALL



D[2]=

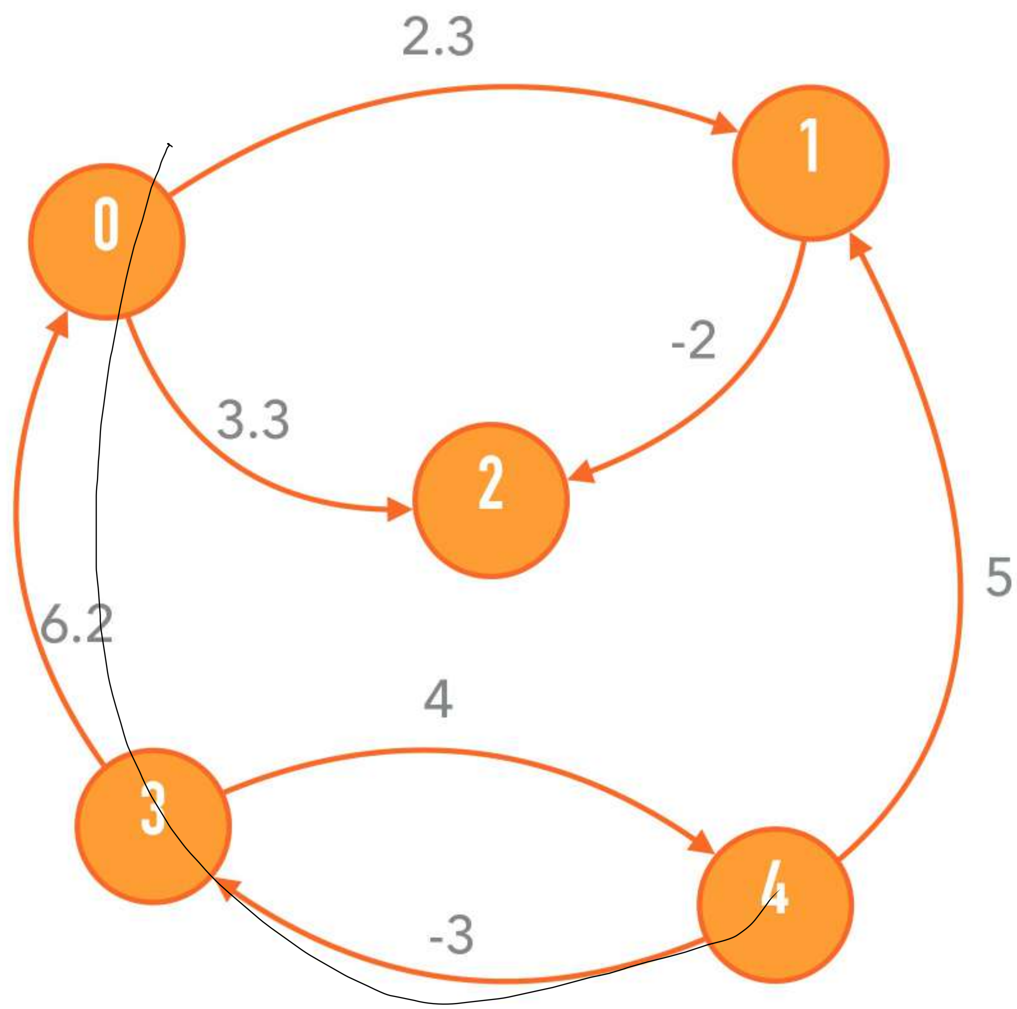
0	2.3	0.3	∞	∞
∞	0	-2	∞	∞
∞	∞	0	∞	∞
6.2	8.5	6.5	0	4
∞	5	3	-3	0

D[3]=

0	2.3	0.3	∞	∞
∞	0	-2	∞	∞
∞	∞	0	∞	∞
6.2	8.5	6.5	0	4
∞	5	3	-3	0

Possiamo passare per i nodi 0, 1 e 2

ALGORITMO DI FLOYD-WARSHALL



D[3]=

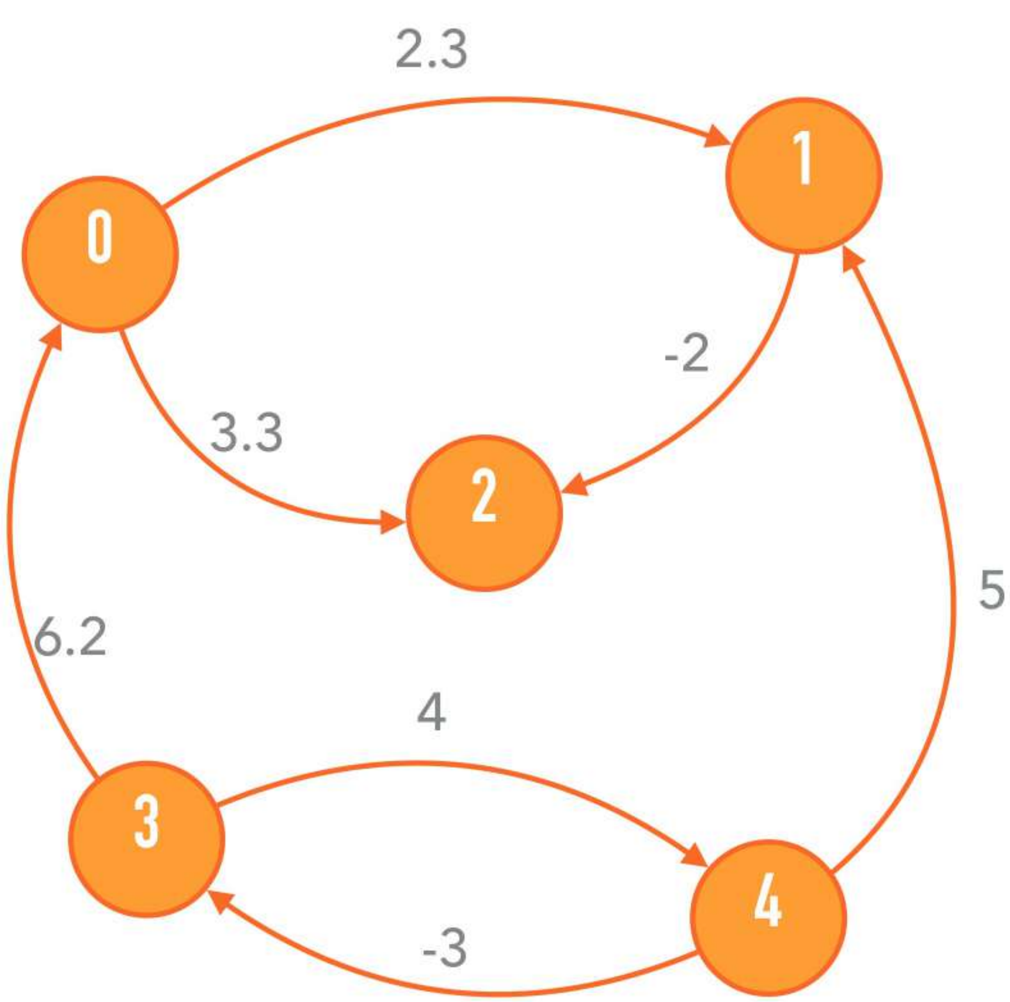
0	2.3	0.3	∞	∞
∞	0	-2	∞	∞
∞	∞	0	∞	∞
6.2	8.5	6.5	0	4
∞	5	3	-3	0

D[4]=

0	2.3	0.3	∞	∞
∞	0	-2	∞	∞
∞	∞	0	∞	∞
6.2	8.5	6.5	0	4
3.2	5	3	-3	0

Possiamo passare per i nodi 0, 1, 2 e 3

ALGORITMO DI FLOYD-WARSHALL



Possiamo passare per i nodi 0, 1, 2, 3 e 4

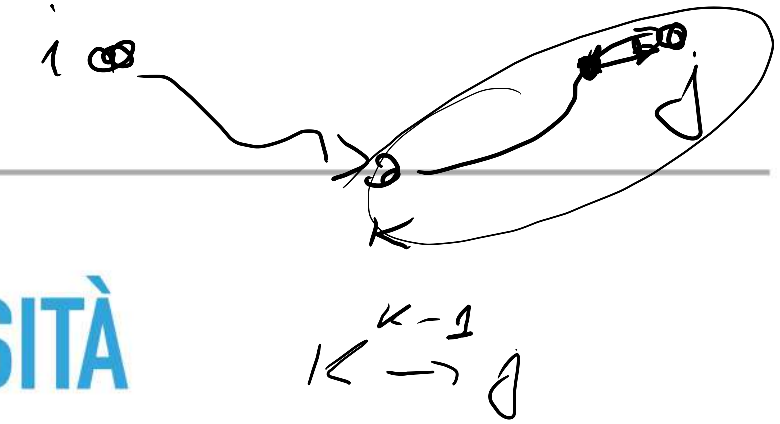
D[4]=

0	2.3	0.3	∞	∞
∞	0	-2	∞	∞
∞	∞	0	∞	∞
6.2	8.5	6.5	0	4
3.2	5	3	-3	0

D[5]=

0	2.3	0.3	∞	∞
∞	0	-2	∞	∞
∞	∞	0	∞	∞
6.2	8.5	6.5	0	4
3.2	5	3	-3	0

Matrice con la lunghezza dei percorsi di costo minimo tra tutte le coppie di vertici



ALGORITMO DI FLOYD-WARSHALL: COMPLESSITÀ

- Abbiamo tre cicli for innestati e ognuno di questi cicli for svolge $|V| = n$ iterazioni

- Ne segue che il tempo di esecuzione è $O(V^3)$, quindi cubico

- Notiamo che rispetto all'algoritmo di Bellman-Ford, che richiede tempo $O(VE)$ con l'algoritmo di Floyd-Warshall otteniamo la distanza tra ogni coppia di vertici, non solo a partire da un nodo sorgente dato

ALGORITMO DI FLOYD-WARSHALL: VARIANTI

- Possiamo aggiungere molteplici restrizioni e varianti all'algoritmo di Floyd-Warshall a seconda delle necessità
- Come esempio, pensiamo di voler stabilire per ogni coppia di nodi i e j il cammino semplice meno costoso per andare da i a j che però contenga un numero pari di nodi
- Come possiamo modificare l'algoritmo?

ALGORITMO DI FLOYD-WARSHALL: VARIANTI

- Teniamo traccia di una informazione aggiuntiva: se il cammino che abbiamo è con un numero pari o dispari di nodi:
  - $d_{i,j}^{k,\text{pari}}$ è il costo minimo per andare da i a j passando per un numero pari di nodi usando solo $\{0,\ldots,k-1\}$ come nodi intermedi
  - $d_{i,j}^{k,\text{dispari}}$ è definito in modo simile ma con un numero dispari di nodi

ALGORITMO DI FLOYD-WARSHALL: VARIANTI

- I casi base sono:
  - $d_{i,i}^{0,\text{pari}} = 0$
  - $d_{i,j}^{0,\text{pari}} = w_{i,j}$ se esiste l'arco (i, j)
  - $d_{i,j}^{0,\text{pari}} = +\infty$ altrimenti
  - $d_{i,j}^{0,\text{dispari}} = +\infty$ dato che senza nodi intermedi ogni percorso ha solo un numero pari di nodi intermedi

ALGORITMO DI FLOYD-WARSHALL: VARIANTI

- Il calcolo delle soluzioni di sotto-problemi cambia solo per mantenere corretta la parità:
- $d_{i,j}^{k+1,\text{pari}} = \min\{d_{i,j}^{k,\text{pari}}, d_{i,k}^{k,\text{dispari}} + d_{k,j}^{k,\text{pari}}, d_{i,k}^{k,\text{pari}} + d_{k,j}^{k,\text{dispari}}\}$
- $d_{i,j}^{k+1,\text{dispari}} = \min\{d_{i,j}^{k,\text{dispari}}, d_{i,k}^{k,\text{pari}} + d_{k,j}^{k,\text{pari}}, d_{i,k}^{k,\text{dispari}} + d_{k,j}^{k,\text{dispari}}\}$
- Ovvero trattiamo in modo esplicito come combinare percorsi di parità diversa cambia la parità del percorso finale

ALGORITMO DI FLOYD-WARSHALL: ESERCIZI

- Alcune varianti che si possono provare a definire, come esercizio:
- Dato un grafo stabilire il costo del percorso di lunghezza minima che contiene un numero **dispari** di **archi**.
- In questo caso cambia un poco la ricorrenza (e i casi base) perché contiamo gli archi e non i nodi