

Introduzione alla Programmazione Dinamica

Programmazione dinamica: i problemi devono essere formulati in termini ricorsivi, per poi essere trascritti, iterativamente, ottimizzando l'algoritmo. È bottom-up.

Strategia bottom-up: I sottoproblemi vengono risolti dal più piccolo al più grande.

Fibonacci: formulazione ricorsiva.

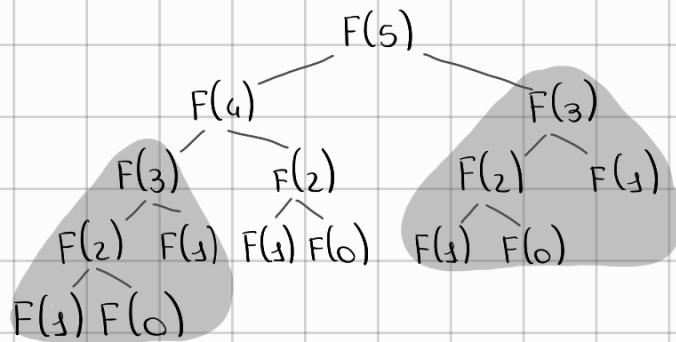
```
if n ≤ 1  
    return n  
else  
    return FIB(n-1) + FIB(n-2)
```

Tempi di calcolo:

$$T(n) = O(1), \text{ se } n \leq 1$$
$$T(n) = K_1 \left(\frac{1+\sqrt{5}}{2}\right)^{n-1} \cdot K_2 \left(\frac{1-\sqrt{5}}{2}\right)^{n-2} + c$$

(esponenziale)

Albero di ricorsione: La ricorsione non si accorge che ha già eseguito dei calcoli.



Soluzione con PD: Usa un array F di dimensione n per memorizzare i risultati dei sottoproblemi.

Codice Fibonacci con PD:

```
int FIB-DP(n)  
if n ≤ 1  
    return n  
else  
    F[0] = 0, F[1] = 1  
    for i=2 to n  
        F[i] = FIB-DP(n-1) + FIB-DP(n-2)  
    return F[n]
```

Tempi e spazio di calcolo: $\Theta(n)$

Grafico:

0	1	1	2	3	5	8
---	---	---	---	---	---	---

Ottimizzazione: Possiamo ottimizzare lo spazio usando un array con 3 elementi; ed effettuando aggiornamenti sulle celle

DP è utile quando l'algoritmo ricorsivo top-down è inefficiente, ovvero quando effettua chiamate ripetute.

DP è vantaggiosa quando il numero di chiamate distinte è polinomiale

Problema di ottimizzazione: per ogni istanza del problema esiste un insieme di possibili soluzioni:

- funzione obiettivo: associa un valore ad ogni soluzione possibile.
- output: soluzione possibile per cui il valore restituito dalla funzione obiettivo è il max/min
- soluzione ottimale: soluzione dell'output
- valore ottimo: valore restituito dall'output

$X = \langle X_1, \dots, X_n \rangle$ sequenza di n simboli

$Z = \langle Z_1, \dots, Z_k \rangle$ sottosequenza di X sse \exists sequenza $\langle i_1, \dots, i_k \rangle$ strettamente crescente di indici tali che $\forall j \in \{1, \dots, k\} \quad Z_j = X_{i_j}$

Esempio : $X = \langle A, B, C, B, D, A, B \rangle \quad n=7$

$Z = \langle B, C, D, B \rangle$ è una sottosequenza di X perché $\exists \langle 2, 3, 5, 7 \rangle$ tc

$$Z_1 = X_{i_1}, \quad Z_2 = X_{i_2}, \quad Z_3 = X_{i_3}, \quad Z_4 = X_{i_4}$$

Definizione: $X = \langle x_1, \dots, x_m \rangle$, $Y = \langle y_1, \dots, y_n \rangle$ su medesimo alfabeto, trovare la sottosequenza Z di X e di Y t.c. $|Z| = \max\{|W| \mid W \text{ sottosequenza sia di } X \text{ che di } Y\}$

LCS è un problema di ottimizzazione di massimo dove $|Z|$ è il valore ottimo del problema e Z è una soluzione ottimale.

Esempio: $X = \langle A, B, C, B, D, A, B \rangle$ $Y = \langle B, D, C, A, B, A \rangle$
 $Z = \langle B, D, A, B \rangle \quad l=4$ o qualsiasi altro W con $|W|=4$

Sottostruttura Ottima

Date $X = \langle x_1, x_2, \dots, x_{m-1}, x_m \rangle$ e $Y = \langle y_1, y_2, \dots, y_{n-1}, y_n \rangle$

$$x_m = y_n \rightarrow \text{LCS}(X, Y) = \text{LCS}(X_{m-1}, Y_{n-1}) + \langle x_m \rangle$$

$$x_m \neq y_n \rightarrow \text{LCS}(X, Y) = \max\{\text{LCS}(X_{m-1}, Y_n), \text{LCS}(X_m, Y_{n-1})\}$$

Equazioni di ricorrenza: si scrivono senza sapere il valore delle soluzioni dei sottoproblemi: non sapendo solamente che tali soluzioni esistano e si possono utilizzare (black-box)

$$i=0 \vee j=0 \quad (\text{caso base})$$

$$c_{i,j}=0$$

$$i>0 \wedge j>0 \quad (\text{passo ricorsivo})$$

$$x_i = y_j \rightarrow c_{i,j} = c_{i-1,j-1} + 1$$

$$x_i \neq y_j \rightarrow c_{i,j} = \max\{c_{i-1,j}, c_{i,j-1}\}$$

Ottengo $c_{m,n}$ valore ottimo del problema

totale $(m+1)(n+1)$ coefficienti $c_{i,j}$

Algoritmi ricorsivi top-down

$LCS(X, Y, i, j)$ calcola $Z^{(i,j)}$

```

if  $i=0 \vee j=0$  return  $\leftrightarrow$ 
else if  $x_i = y_j$ 
    return  $LCS(X, Y, i-1, j-1) \cup x_i$ 
else
    A =  $LCS(X, Y, i-1, j)$ 
    B =  $LCS(X, Y, i, j-1)$ 
    if  $|A| \geq |B|$  return A
    else return B
  
```

$L-LCS-R(X, Y, i, j)$ calcola $c_{i,j}$

```

if  $i=0 \vee j=0$  return 0
else
    if  $x_i = y_j$  return
         $L-LCS-R(X, Y, i-1, j-1) + 1$ 
    else
        A =  $LCS-R(X, Y, i-1, j)$ 
        B =  $LCS-R(X, Y, i, j-1)$ 
        return  $\max\{A, B\}$ 
  
```

Dimostrazione Sottostruttura ottima

Date $X = \langle x_1, x_2, \dots, x_m \rangle$ e $Y = \langle y_1, y_2, \dots, y_n \rangle$ e $LCS(X, Y) = Z = \langle z_1, z_2, \dots, z_k \rangle$:

$x_m = y_n$:

1. $z_k = x_m = y_n$
2. $z_{k-1} = LCS(X_{m-1}, Y_{n-1})$

$x_m \neq y_n$:

3. $z_k \neq x_m \rightarrow LCS(X, Y) = LCS(X_{m-1}, Y_n)$
4. $z_k \neq y_n \rightarrow LCS(X, Y) = LCS(X, Y_{n-1})$

1. Dimostrazione per assurdo: se $z_k \neq x_m$ per assurdo, allora $Z' = Z \cup \langle x_m \rangle$ è tc Z' è sottosequenza di X e Y e $|Z'| > |Z|$, ma quindi Z non è più la più lunga sottosequenza comune di X e Y .

2. Dimostrazione per assurdo: $z_{k-1} \neq LCS(X_{m-1}, Y_{n-1})$ per assurdo allora esiste Z'' sottosequenza comune di X_{m-1} e Y_{n-1} tc $|Z''| > |Z_{k-1}| = k-1$? Se prendo $Z'' \cup \langle x_m \rangle$ ottengo $|Z''| + 1 > k$. Giungiamo ad un assurdo.

3. Dimostrazione per assurdo: $z_k \neq x_m \rightarrow Z$ è sottosequenza di X_{m-1} e Y . $Z \neq LCS(X_{m-1}, Y)$ per assurdo allora esiste Z' tc $|Z'| > k$. Ma $x_m \neq z_k$ quindi Z' è sottosequenza di X e Y più lunga di k . Assurdo.

Algoritmo DP (bottom-up)

Idea algoritmo iterativo bottom-up per ottenere l'ottimo:

1. Si costruisce una matrice C di $m+1$ righe e $n+1$ colonne
2. Si indicizzano le righe e colonne a partire da 0.
3. Si riempie C in modo tale che $C[i,j] = c_{i,j}$
4. Valore ottimo $\rightarrow C[m,n] = c_{m,n}$

Algoritmo DP (bottom-up)

```
int ottimo.DP(x, y)
```

```
for i from 0 to m do
```

$$C[i,0] = 0$$

```
for j from 0 to n do
```

$$C[0,j] = 0$$

```
for i from 1 to m do
```

```
for j from 1 to n do
```

```
if  $x_i = y_j$  then
```

$$C[i,j] = C[i-1,j-1] + 1$$

```
else
```

$$C[i,j] = \max(C[i-1,j], C[i,j-1])$$

```
return C[m,n]
```

Matrice dei coefficienti

	E	B	D	C	A	B	A
E	0	0	0	0	0	0	0
A	0	0 [↑]	0 [↑]	0 [↑]	1	1	1
B	0	1	1	1 [↑]	1	2	2
C	0	1	1	2	2	2	2
B	0	1	1	1	1	1	1
D	0	1	2	2	2	3	3
A	0	1	2	2	3	3	4
B	0	1	2	1	3	4	4

Procedura ricostrisci-LCS(C, i, j)

if $i > 0$ and $j > 0$ then

if $x_i = y_j$ then

 ricostrisci-LCS($C, i-1, j-1$)

 print x_i

else

 if $C[i, j] = C[i-1, j]$ then
 ricostrisci-LCS($C, i-1, j$)

 else

 ricostrisci-LCS($C, i, j-1$)

	E	B	D	C	A	B	A
E	0	0	0	0	0	0	0
A	0	0 [↑]	0 [↑]	0 [↑]	1 [↑]	1 [↑]	1 [↑]
B	0	1 [↑]	2 [↑]				
C	0	1 [↑]	1 [↑]	2 [↑]	2 [↑]	2 [↑]	2 [↑]
B	0	1 [↑]	1 [↑]	2 [↑]	2 [↑]	3 [↑]	3 [↑]
D	0	1 [↑]	2 [↑]	2 [↑]	2 [↑]	3 [↑]	3 [↑]
A	0	1 [↑]	2 [↑]	2 [↑]	3 [↑]	3 [↑]	4 [↑]
B	0	1 [↑]	2 [↑]	1 [↑]	3 [↑]	4 [↑]	4 [↑]

N° iterazioni per $X = Y$? $m = n$ iterazioni

N° iterazioni per X sottosequenza di Y? $|Y| = n$ iterazioni

N° iterazioni per X e Y quando non hanno elementi in comune? $\max\{m, n\}$

Ricostruzione LCS iterativa

List Ricostrisci-LCS(C, m, n)

$i \leftarrow m$, $j \leftarrow n$; $LCS \leftarrow$ empty list;

while $i > 0$ and $j > 0$ do

 if $x_i = y_j$ then

 aggiungi in testa x_i a LCS

$i \leftarrow i-1$

$j \leftarrow j-1$

 else

 if $C[i, j] = C[i-1, j]$ then

$i \leftarrow i-1$

 else

$j \leftarrow j-1$

return LCS

Definizione del problema PB: Data una sequenza X di n numeri interi, si determini una tra le più lunghe sottosequenze crescenti di X .

Definizione del problema PBR: Data una sequenza X di n numeri interi, si determini la lunghezza di una tra le più lunghe sottosequenze crescenti di X .

Sottoproblema di dimensione i : Data una sequenza X di m numeri interi, si determini la lunghezza di una tra le più lunghe sottosequenze crescenti di X .

Variabile associata c_i : lunghezza di una tra le più lunghe sottosequenze crescenti di X .

Considerato un qualsiasi sottoproblema di dimensione i , assumendo di aver già risolto tutti i sottoproblemi di dimensione minore come è possibile ottenere la soluzione c_i andando a combinare le soluzioni dei sottoproblemi di dimensione minore $< i$ ed eventualmente accodando l'elemento x_i estratto dall'input X ?

Mancanza informazione: Date solamente le variabili $\{c_0, \dots, c_{i-1}\}$ e il prefisso X_i non c'è alcun modo per poter comprendere se l'elemento x_i possa essere accodato: non sappiamo con quale elemento termini ognuna di queste sottosequenze ma ne conosciamo solo la lunghezza.

Problema ausiliario P' di PBR: Data una sequenza X di n numeri interi, si determini la lunghezza di una tra le più lunghe sottosequenze crescenti di X e che termina con x_m .

Teorema proprietà sottostruzione ottima: Sia X una sequenza di m numeri interi e sia X_i un suo prefisso di lunghezza $1 \leq i \leq m$. Sia Z^i una tra le più lunghe sottosequenze crescenti di X_i e che termina con x_i . Allora vale che $Z^i = Z^* | x_i$ con $Z^* \in W_i$ e $|Z^*| = \max_{W \in W_i} \{|W|\}$ dove W_i è l'insieme di tutte le sottosequenze crescenti di X_j che finiscono con x_j e su cui è possibile concatenare x_i , ovvero $W_i = \bigcup_{1 \leq j \leq i : x_j < x_i} \{W \text{ sottosequenza crescente di } X_j \text{ che termina con } x_j\}$

Dimostrazione proprietà sottostruttura ottimale. Per assurdo si suppone che $Z^*|_{X_i}$ non sia la soluzione del problema i -esimo. Allora vengono le seguenti informazioni:

$$Z' = Z^*|_{X_i}$$

$|Z'| > |Z^*|$ (dato che Z' si ottiene partendo da Z^* e non Z^* , allora necessariamente Z' sarà più lunga di Z^*)

dove Z' è una qualunque sottosequenza crescente di un prefisso più piccolo di X_i . Sia ora z' l'ultimo elemento di Z' . Vale quindi $z' < x_i$. Sia $h < i$ il più grande indice tale che $x_h = z'$. Per come è stato definito W_h , si ottiene che Z' è W_h . Infatti Z' è una sottosequenza crescente di X_h , che termina con $x_h < x_i$. Ciò porta ad una contraddizione: dal punto 2 vale che $|Z'| > |Z^*|$ ma ciò è in contraddizione con l'ipotesi che $|Z^*| = \max_{W \in W_i} \{|W|\}$

Equazioni di ricorrenza: si scrivono senza sapere il valore delle soluzioni dei sottoproblemi ma sapendo solamente che tali soluzioni esistono e si possono utilizzare.

Caso base: il caso base si ha quando $i=0$ $\forall i=1$, ovvero il prefisso è la sequenza vuota oppure è una sequenza composta da un solo elemento. Avremo che nel primo caso $c_0=0$ e nel secondo $c_1=1$.

Passo ricorsivo: il passo ricorsivo si ha per qualunque sottoproblema di dimensione (i) con $i > 1$. I dati disponibili per calcolare c_i sono: X e in particolare x_i e tutte le variabili che $\{c_0, \dots, c_{i-1}\}$. Queste rappresentano le lunghezze delle più lunghe sottosequenze crescenti X_3, \dots, X_{i-1} che terminano con x_3, \dots, x_{i-1} . Tra queste ci saranno sottosequenze cui è possibile accodare x_i in quanto maggiore dell'ultimo elemento e altre no. Se perdiamo la più lunga sottosequenza cui è possibile accodare x_i e accodiamo x_i , otteniamo la più lunga sottosequenza crescente X_i che termina con x_i , ovvero:

$$c_i = 1 + \max \{c_h \mid 1 \leq h \leq i \wedge x_h < x_i\}$$

Poiché l'insieme può essere vuoto assumiamo che $\max \emptyset = 0$

LIS

Soluzione problema: Una volta calcolati i c_0, \dots, c_m si hanno a disposizione tutte le lunghezze di una tra le più lunghe sottosequenze crescenti dei vari prefissi di X e che terminano con l'ultimo elemento del prefisso. La soluzione del problema ausiliario P' è c_m mentre quella del PBR è $\max\{c_i \mid 1 \leq i \leq m\}$

Algoritmo DP:

procedure LIS(x)

$c[1] = 1$

$\max = c[1]$

for $i = 2$ to m do

$\text{temp} = 0$

 for $h = 1$ to $i - 1$

 if $(x_h < x_i) \wedge (c[h] > \text{temp})$

$\text{temp} = c[h]$

$c[i] = 1 + \text{temp}$

 if $c[i] > \max$

$\max = c[i]$

return \max

Definizione: Dato $X = \{1, 2, \dots, i, \dots, n\}$ n oggetti, un intero C e due funzioni:

$V: X \rightarrow \mathbb{N}$ t.c. $V(i) = v_i$ è il valore dell'oggetto i ;

$W: X \rightarrow \mathbb{N}$ t.c. $W(i) = w_i$ è il peso dell'oggetto i ;

si vuole trovare un insieme $S = \{i_1, i_2, \dots, i_k\}$ di X t.c.

$W_S = \sum_{j=1}^k w(i_j) \leq C$ (la somma dei pesi dell'insieme S è \leq del limite)

$V_S = \sum_{j=1}^k v(i_j) = \max$ tra tutti i valori dei possibili sottinsiemi di X compatibili con lo zaino.

Problema: ottimizzazione di massimo dove:

$(n, C) \rightarrow$ dimensione del problema

Sol. possibili \rightarrow \forall sottinsieme S' t.c. $W_{S'} \leq C$

Funz. obiettivo \rightarrow valore totale $V_{S'}$ della soluzione possibile S' .

Valore ottimo \rightarrow valore totale di S

Soluzione ottimale $\rightarrow S$

Sottostruttura ottima: data $X = \{1, \dots, n\}$ e una sol. ottimale S e capacità C si può verificare

$x_n \in S$: avremo che $C \geq w_n$, allora $S' = S \setminus \{n\}$ sol. ottimale di $X \setminus \{n\} = \{1, \dots, n-1\}$ e $C' = C - w_n$
dimostrazione:

1. S' compatibile con $C - w_n$. $W_{S'} \leq C$ allora $W_{S'} + w_n \leq C - w_n$ e ne segue $W_{S'} \leq C - w_n$

2. S' ha valore totale $V_{S'}$ massimo: se esiste S'' t.c. $V_{S''} > V_{S'}$ e $W_{S''} \leq C - w_n$

allora $S'' \cup \{n\}$ ottimale per X compatibile con C e valore $> V$; contro l'ipotesi.

Problema $(n-1, C - w_n)$

$x_n \notin S$: S sol. ottimale per $X \setminus \{n\}$ e capacità = C

dimostrazione:

Se $\exists S'$ sol. ottimale per $X \setminus \{n\}$ compatibile con C e valore ottimo $V_{S'} > V_S$ allora

S' soluzione ottimale per X . Contro ipotesi.

Problema $(n-1, C)$

Passo ricorsivo (n, C): Dato $X = \{1, \dots, n\}$, C, S :

$$S_{n,C} = \begin{cases} S_{n-1,C} \cup \{S' \cup \{n\}, S''\} & \text{se } C \geq w_n \\ S_{n-1,C} & \text{se } C < w_n \end{cases}$$

dove $S' = S_{n-1,C-w_n}$ e $S'' = S_{n-1,C}$.

Def. sottoproblemi: dato $i \in \{0, 1, \dots, n\}$ e $c \in \{0, 1, \dots, C\}$ avremo come numero di sottoprobl.: $(n+1)(C+1)$
dove $i=n$ e $c=C$ è il problema principale.

Equazioni di ricorrenza:

caso base: tutti i sottoproblemi (i, c) tc $i=0 \vee c=0$

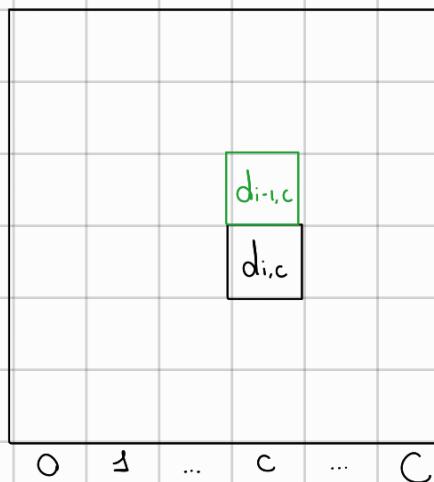
passo ricorsivo: tutti i sottoproblemi (i, c) tc $i>0 \wedge c>0$ dove

$$S_{i,c} = \begin{cases} S_{i-1,c-w_i} \cup \{i\}, S_{i-1,c} & \text{se } c \geq w_i \\ S_{i-1,c} & \text{se } c < w_i \end{cases}$$

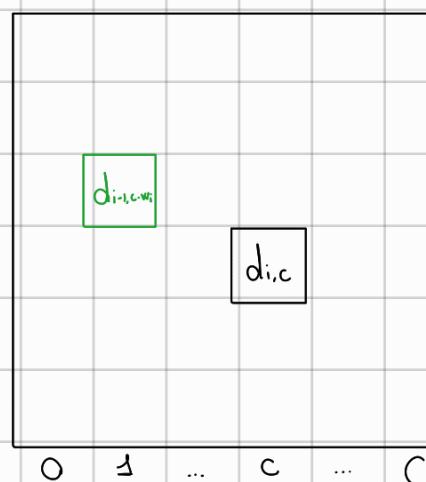
Def. coefficienti: $d_{i,c}$ valore totale di $S_{i,c}$ numero coefficienti: $(n+1)(C+1)$ dove $d_{n,C}$ val. ottimo

Algoritmo DP (bottom-up)

1. costruzione matrice $D[0 \dots n][0 \dots C]$
2. riempimento di D in modo tale che $D[i,c] = d_{i,c}$
3. valore ottimo $D[n,C]$



0
1
...
 $i (v_i, w_i)$
...
n



0
1
...
 $i (v_i, w_i)$
...
n

Knapsack - valore ottimo e sol. ottimale

4

Algoritmo DP valore ottimo:

```
int calcolo_ottimo_zaino(n, C, V, W)
    for i = 0 to n
        D[i, 0] = 0
    for c = 1 to C
        D[0, c] = 0
    for i = 1 to n
        for c = 1 to n
            if c ≥ wi ∧ D[i-1, c-wi] + vi > D[i-1, c]
                D[i, c] = D[i-1, c-wi] + vi
            else:
                D[i, c] = D[i-1, c]
    return D[n, C]
```

tempo e spazio: $\Theta(nC)$. Knapsack è np-completo pseudopolinomiale in quanto l'output varia al cambiare di C . Con C molto grande si necessitano $\log C$ bit e quindi esponenziale.

Algoritmo ricorsivo sol. ottimale:

```
Procedura ricostruzione_zaino(D, i, c).
    if i > 0 ∧ j > 0
        if c ≥ wi
            if D[i, c] == D[i-1, c]
                ricostruisci_zaino(D, i-1, c)
            else
                ricostruisci_zaino(D, i-1, c-wi)
                print i
        else
            ricostruisci_zaino(D, i-1, c) //sforo matrice, quindi prendo su
```

Floyd-Warshall

Problema dei cammini minimi:

Input: Grafo $G = (V, E, W)$ senza capi orientato e pesato

$V = \{1, 2, \dots, n\}$ insieme dei vertici. $E = \{e_1, e_2, \dots, e_n\}$ insieme degli archi.

$W: E \rightarrow \mathbb{R}^+$ tc $W(i, j) = w_{ij}$ = peso dell'arco (i, j)

Output: $\forall (i, j)$ trovare cammino di peso minimo che parte da i e finisce in j .

Problema ottimizzazione minima:

dim. problema: n

sol. possibili (i, j) : tutti i cammini (i, j)

fanz obiettivo: peso del cammino

valore ottimo: peso del cammino minimo da i a j

soluzione ottimale: un cammino minimo da i a j

Funzione peso w : $W: V \times V \rightarrow \mathbb{R}^+$ tc $W(i, j) = w_{ij}$ con:

$w_{ij} = 0$ se $i = j$

$w_{ij} = \text{peso } (i, j)$ se $(i, j) \in E$

$w_{ij} = \infty$ se $(i, j) \notin E$

Output: Preso in input la matrice W , avremo in output:

Matrice $D_{n \times n}$ dove $d_{i,j}$ peso del cammino minimo da i a j

Matrice $\Pi_{n \times n}$ dove $\pi_{i,j}$ predecessore di j nel cammino minimo da i a j

Matrice D :

$d_{i,j} = 0$ se $i = j$

$d_{i,j} = \text{peso cammino minimo se } \exists \text{ cammino}$

$d_{i,j} = \infty$ se $\nexists \text{ cammino}$

Matrice Π :

$\pi_{i,j} = \text{NIL}$ se $i = j$

$\pi_{i,j} = v$ tc $(v, j) \in E$

$\pi_{i,j} = \text{NIL}$ se $\nexists \text{ cammino } i, j$

Albero dei predecessori: la riga i -esima della matrice Π crea l'albero dei predecessori di i

Algoritmo Floyd-Warshall: Diamo un ordine sui vertici del grafo e parametrizziamo rispetto ai vertici intermedi del cammino:

P_{ij}^0 : cammino i,j senza vertici intermedi

P_{ij}^1 : cammino i,j con vertici intermedi $\in \{j\}$

P_{ij}^2 : cammino i,j con vertici intermedi $\in \{j, 2\}$

P_{ij}^n : cammino i,j con vertici intermedi $\in \{1, 2, \dots, n\}$

Definizione sottoproblemi: Per ogni coppia (i,j) trovare il cammino P_{ij}^K dal vertice i al vertice j che ha vertici intermedi $\in \{1, 2, \dots, K\}$ se $K > 0$ o non ne ha se $K = 0$:

$$K \in \{0, 1, \dots, n\}$$

$$i \in \{1, \dots, n\}$$

$$j \in \{1, \dots, n\}$$

Numero sottoproblemi: $n^2 \cdot K + 1$

Sottostruttura ottimale: Data una soluzione ottimale ci troviamo in 2 casi:

$$P_{ij}^K = \begin{cases} P_{ij}^{K-1} & \text{se il vertice } K \text{ non è uno dei vertici intermedi;} \\ P_{ik}^{K-1} + P_{kj}^{K-1} & \text{se il vertice } K \text{ è uno dei vertici intermedi.} \end{cases}$$

Equazioni di ricorrenza:

caso base:

$$P_{ij}^0 = \langle i \rangle \quad \text{se } i=j$$

$$P_{ij}^0 = \langle i, j \rangle \quad \text{se } (i, j) \in E$$

$$P_{ij}^0 = \text{NIL} \quad \text{se } (i, j) \notin E$$

passo ricorsivo

$$P_{ij}^K = \min_p \{ P_{ij}^{K-1}, P_{ik}^{K-1} + P_{kj}^{K-1} \}$$

Coefficienti d_{ij}^K : peso del cammino P_{ij}^K ($n^2(K+1)$ coefficienti)

Algoritmo bottom-up:

Procedura calcola-valor-ottimi-FW(V, E, W):

$$D^0 = W$$

Π^0 : ($n \times n$) matrix of NIL values

for $i=1$ to n

 for $j=1$ to n

 if $i \neq j$ and $W_{ij} \neq \infty$

$$\Pi^0[i,j] = i$$

 for $k=1$ to n

 for $i=1$ to n

 for $j=1$ to n

$$D^K[i,j] = D^{K-1}[i,j]$$

$$\Pi^K[i,j] = \Pi^{K-1}[i,j]$$

 if $i \neq K$ AND $j \neq K$

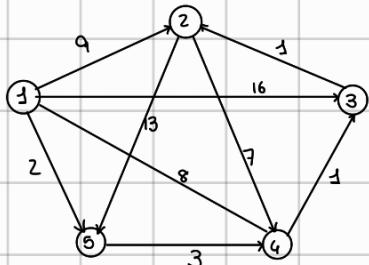
$$\text{if } D^K[i,j] > D^{K-1}[i,K] + D^{K-1}[K,j]$$

$$D^K[i,j] = D^{K-1}[i,K] + D^{K-1}[K,j]$$

$$\Pi^K[i,j] = \Pi^{K-1}[K,j]$$

Tempo e spazio: $\Theta(n^3)$ per il tempo, $\Theta(n^3)$ ottimizzabile a $\Theta(n^2)$ per lo spazio.

Esempio:



D	1	2	3	4	5		1	2	3	4	5
1	0	7	6	5	2		NIL	3	4	5	1
2	15	0	8	7	33		4	NIL	4	2	2
3	36	1	0	8	34		4	3	NIL	2	2
4	8	2	1	0	30		4	3	4	NIL	1
5	51	5	4	3	0		4	3	4	5	NIL