Analisi e Progetto di Algoritmi

Elia Ronchetti
@ulerich

2023/2024

Indice

Capitolo 1

Programmazione Dinamica - DP

La programmazione dinamica (DP - Dynamic Programming) è una tecnica che (come il Divide et Impera), risolve i problemi combinando le soluzioni dei sottoproblemi.
Divide et Impera è ottimo quando i sottoproblemi da risolvere sono indipendenti, mentre DP è efficace quando i sottoproblemi non sono indipendenti e quindi hanno in comune dei sottosottoproblemi e le tecniche di risoluzione top-down risultano quindi inefficienti (chiamate ripetute). La programmazione dinamica si applica tipicamente ai **problemi di ottimizzazione**.

1.1 Problemi di ottimizzazione

Sono problemi dove ci sono molte soluzioni possibile. Ogni soluzione ha un valore e e si vuole trovare una solzuione con il valore ottimo. Ci possono essere più soluzioni che raggiungono il valore ottimo.

1.2 Il processo di sviluppo

Il processo di sviluppo è diviso in 4 fasi:

- Caratterizzare la struttura di una soluzione ottima
- Definire in modo ricorsivo il valore di una soluzione ottima
- Calcolare il valore di una soluzione ottima, di solito con uno schema bottom-up (dal basso verso l'alto, risulta spesso più efficiente rispetto a top-down)

- Costruire una soluzione ottima dalle informazioni calcolate

1.3 Esempio - Fibonacci

Classico esempio è l'esecuzione di Fibonacci. Utilizzando la ricorsione pura si effettuano più volte le stesse chiamate (perchè i sotto-numeri sono gli stessi). Se invece utilizziamo la DP, con un approccio Bottom-Up ci dobbiamo chiedere, ma chi è Fibonacci di n? è Fibonacci di (1)+Fibonacci(2)+ ... + Fibonacci(n). In pratica inizio a calcolare le soluzioni dal sottoproblema più piccolo a salire, così facendo possiamo risparmiare molto tempo, al costo però di un maggiore utilizzo di spazio, dato che ho un Array che deve memorizzare i valori. Si tratta di un compromesso accettabile dato che senza usare Array il tempo di esecuzione sarebbe esponenziale.

1.3.1 Passaggi

A livello pratico dobbiamo:

1. Scomporre il problem in sottoproblemi di dimensione inferiore
2. Formulare la soluzione in maniera ricorsiva - Equazioni di Ricorrenza
3. Usare una strategia bottom-up (non top-down)
4. Memorizzare i risultati in una opportuna struttura dati
5. Individuare il "luogo" che contiene la soluzione del problema (nel caso di Fibonacci l'ultima cella a destra)

DP risulta vantaggiosa quando il numero di chiamate distine è polinomiale (il numero totale di chiamate è esponenziale).

1.4 Osservazioni sui problemi di ottimizzazione

Per ogni istanza del problema esiste un insieme di soluzioni posibili (feasible solutions), più soluzioni perchè le soluzioni ottime possono essere diverse.
Esiste una funzione obiettivo che associa un valore ad ogni soluzione possibile e restituisce come OUTPUT una soluzione possibile (soluzione ottimale) per cui il valore restituito dalla funzione obiettivo è massimo/minimo (valore ottimo).

1.5 LCS - Longest Common Subsequence

Si tratta di un problema che ha come istanza due sequenze di valori e richiede di trovare la più grande sottosequenza comune fra di esse. Si tratta di un problema di ottimizzazione, per questo usare DP è un'ottima idea.

1.5.1 Definizioni di base

Sequenza Successione di elementi topologicamente ordinati, presi da un insieme $\sum$.
Per esempio $X =< 2, 4, 10, 5, 9, 11 >$, più in generale:

- $X =< x_1, x_2, ..., x_m > \rightarrow$ sequenza di $m = |X|$ elementi

Prefisso di lunghezza i Primi i elementi della sequenza:

- $X =< x_1, x_2, ..., x_i > \rightarrow$ prefisso di lunghezza i di X

Dato $X =< 2, 4, 10, 5, 9, 11 >$ per esempio $X_3 =< 2, 4, 10 >$.

i-esimo elemento Indichiamo con $X[i]$ l'i-esimo elemento x_i della sequenza X.

Sottosequenza Una qualsiasi successione di elementi (anche non consecutivi) di una sequenza che però rispettino l'ordine sulla sequenza.
Per esempio data una sequenza $X =< 2, 4, 10, 5, 9, 11 >$

- $Z =< 4, 5, 9 >$ è una sottosequenza di X
- $Z =<>= \epsilon$ è una sottosequenza di X
- $< 9, 5, 4 >$ NON è una sottosequenza di X

Definizione formale di sottosequenza Data $X =< x_1, x_2, ..., x_m >$, una sequenza $Z =< z_1, z_2, ..., z_k >$ $(k \leq m)$ è sottosequenza di X se esiste una successione di k indici interi $i_1 < i_2 < .. < i_k$ tali che $X[i_j] = z_j$ per j compreso tra 1 e k.

Esempio sottosequenza Dato $X =< 2, 4, 10, 5, 9, 11 >$, $Z =< 4, 5, 9 >$ è una sottosequenza di X.

Sottosequenza comune di X e Y è una sottosequenza si di X che di Y.

$$X =< 1, 13, 5, 3, 1, 12, 8, 11, 6, 10, 10 >$$

$$Y =< 1, 5, 5, 2, 3, 1, 12, 8, 8, 10 >$$

$$S =< 5, 3, 1, 8, 10 >$$

S è sottosequenza comune di X e Y.

LCS è la più lunga sottosequenza comune Z di X e Y.

Esempio di LCS

$$X =< 2, 10, 5, 3, 1, 12, 8, 30, 11, 6, 10, 13 >$$

$$Y =< 2, 5, 10, 2, 3, 1, 30, 12, 6, 8, 10 >$$

$$< 2, 10, 3, 1, 12, 8, 10 > \text{è LCS di X e Y}$$

La LCS è una soluzione ottimale, mentre la sua lunghezza (7) è il valore ottimo.

1.5.2 Istanza del problema

P: date due sequenze $X =< x_1, x_2, ..., x_n > eY =< y_1, y_2, ..., y_n >$, trovare la più lunga sottosequenza comune Z di X e Y.
Abbiamo capito che P è un problema di ottimizzazione di massimo, dove:

- (m,n) $\rightarrow$ è la dimensione del problema (lunghezza stringhe)
- Soluzioni possibili $\rightarrow$ tutte le sottosequenze comuni di X e Y
- Funzione obiettivo $\rightarrow$ lunghezza
- $|Z|$ è il valore ottimo del problema
- Z è una soluzione ottimale

1.6 Procedura LCS

Indichiamo con LCS(A,B) la LCS delle sequenze A e B e di conseguenza $|LCS(A, B)|$ la lunghezza della LCS di A e B.
Procediamo con le seguenti fasi:

1. Individuiamo i sottoproblemi

2. Troviamo le equazioni di ricorrenza

3. Applichiamo una strategia bottom-up con memorizzazione dei risultati

Nota Bene Si deve individuare la sottostruttura ottima del problema. La strategia bottom-up trova l'ottimo (lunghezza di LCS) e in seguito si deve ricostruire una soluzione ottimale (una delle LCS).

1.6.1 Definizione dei sottoproblemi

Sottoproblema di dimensione (i,j). Trovare la LCS dei prefissi X_i e $Y_j \rightarrow LCS(X_i, Y_j)$.

$$i \in \{0, 1, ..., m\}$$

$$j \in \{0, 1, ..., n\}$$

Numero totale sottoproblemi: $(m + 1) \times (n + 1)$
Ricordiamo che $LCS(X_m, Y_n)$ è la soluzione del problema principale.

1.6.2 Equazioni di ricorrenza

Casi base

Tutti i sottoproblemi di dimensione (i,j) tale per cui $i = 0$ oppure $j = 0$.

$$\begin{aligned} i = 0 &\implies LCS(X_0, Y_j) = LCS(\epsilon, Y_j) = \epsilon \\ j = 0 &\implies LCS(X_i, Y_0) = LCS(X_i, \epsilon) = \epsilon \\ i = 0,\, j = 0 &\implies LCS(X_0, Y_0) = LCS(\epsilon, \epsilon) = \epsilon \end{aligned}$$

Passo ricorsivo

Tutti i sottoproblemi di dimensione (i, j) tale per cui $i > 0$ e $j > 0$.
Introduciamo la sottostruttura ottima del problema:

Esempio di sottostruttura ottima

$$X =< (3, 5, 4, 3, 10, 12, 8, 30) > \quad m = 8$$
$$Y =< (3, 2, 4, 10, 2, 8, 30, 13, 30) > \quad n = 9$$

LCS(X,Y) $\rightarrow Z =< z_1, z_2, \ldots, z_{k-1}, z_k >=< 3, 4, 10, 8, 30 >$.
Sicuramente $z_k = 30$. Come mi comporto per $k-1$? $z_1, z_2, ..., z_{k-1} >\rightarrow LCS(?, ?)?$.
Avrò sicuramente che sarà la LCS di $(k-1) + 30$, quindi:

$$LCS(X, Y) = LCS(X_7, Y_8)+ < 30 >$$

1.6.3 Sottostruttura ottima

Date:

- $X =< x_1, x_2, \ldots, x_{m-1}, x_m >$
- $Y =< y_1, y2, \ldots, y_{n-1}, y_n >$
- $LCS(X, Y) =< z_1, z_2, \ldots, z_{k-1}, z_k >$

La sottostruttura ottima sarà data da:
$x_m = y_n \implies LCS(X, Y) = LCS(X_{m-1}, Y_{n-1})+ < x_m >$

1. $z_k = x_m = y_n$
2. $< z_1, z_2, \ldots, z_{k-1} >= LCS(X_{m-1}, Y_{n-1})$

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

1. $z_k \neq x_m \implies LCS(X, Y) = LCS(X_{m-1}, Y_n)$
2. $z_k \neq y_n \implies LCS(X, Y) = LCS(X_m, Y_{n-1})$

1.6.4 Equazioni di ricorrenza

i=0 $\vee$ j = 0 (CASI BASE)

$$LCS(X_i, Y_j) = \epsilon$$

$i > 0 \wedge j > 0$ **(PASSO RICORSIVO)**

$$x_i = y_j \implies LCS(X_i, Y_j) = LCS(X_{i-1}, Y_{j-1})+ < x_i >$$
$$x_i \neq y_j \implies LCS(X_i, Y_j) = max\{LCS(X_{i-1}, Y_j), LCS(X_i, Y_{j-1})\}$$

Applichiamo la funzione "lunghezza" alle equazioni

La funzione lunghezza è definita come segue:

$$c_{i,j} = |LCS(X_i, Y_j)|$$

i=0 $\vee$ j = 0 (CASI BASE)

$$|CS(X_i, Y_j)| = |\epsilon| = 0$$

$i > 0 \wedge j > 0$ **(PASSO RICORSIVO)**

$$x_i = y_j \implies |LCS(X_i, Y_j)| = |LCS(X_{i-1}, Y_{j-1})| + | < x_i > |$$
$$x_i \neq y_j \implies |LCS(X_i, Y_j)| = max\{|LCS(X_{i-1}, Y_j)|, |LCS(X_i, Y_{j-1})|\}$$

Calcoliamo la lunghezza di ciò che è noto

i=0 $\vee$ j = 0 (CASI BASE)

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

Dato che ho i=0 e j=0.

$i > 0 \wedge j > 0$ **(PASSO RICORSIVO)**

$$x_i = y_j \implies c_{i,j} = c_{i-1,j-1} + 1 \quad x_i \neq y_j \implies c_{i,j} = max\{c_{i-1,j}, c_{i,j-1}\}$$

Dato che $< x_i >$ è un singolo carattere, quindi ha lunghezza 1. Le sostituzioni delle funzioni LCS sono la semplice sostituzione della definizione di funzione lunghezza sopra riportata.

Quanti coefficienti/variabili $c_{i,j}$? (m+1)(n+1) coefficienti/variabili.

$c_{m,n} = |LCS(X_m, Y_n)| = |LCS(X, Y)|$, **è il valore ottimo del problema principale**.

Soluzione del problema

- Calcolo del valore ottimo (top-down oppure bottom-up?)
- Ricostruzione di una soluzione ottimale

Perchè io tramite la procedura effettuo il riempimento della matrice, devo capire quindi dove si trova il valore (in questo caso abbiamo visto in $c_{m,n}$) e devo ricostruire la sottosequenza dato che la matrice contiene lunghezze, non stringhe.

```
int ottimo-ricorsivo(i,j)
    if i = 0 || j = 0 then
        return 0
    else
        if x_i = y_j then
            c_{i,j} = ottimo-ricorsivo(i-1,j-1)+1
            return c_{i,j}
        else
            c_{i-1,j} = ottimo-ricorsivo(i-1,j)
            c_{i,j-1} = ottimo-ricorsivo(i,j-1)
            return max{c_{i-1,j}, c_{i,j-1}}
```

Valore ottimo → ottimo-ricorsivo(m,n).

Complessità in tempo nel caso migliore? $\Omega(n)$ nel caso migliore in cui $X = Y$ e $|X| = n$.

Complessità in tempo nel caso peggiore? Esponenziale, perchè continua a creare dei rami per testare la stringa diminunedo di 1 prima a sinistra e poi a destra. Quindi non è una buona idea in questo caso applicare questa strategia.

1.6.5 Riassumendo - Per calcolare l'ottimo

1. Definizione dei sottoproblemi
2. Equazioni di ricorrenza
   - Casi base
   - Passo ricorsivo (sottostruttura ottima)
3. Definizione dei coefficienti/variabili (valori ottimi dei sottoproblemi)
4. Individuazione del coefficiente ottimo (valore ottimo dle problema principale)
5. Equazioni di ricorrenza in termini dei coefficienti
6. Calcolo dei coefficienti seguendo una strategia bottom-up
7. Determinazione del valore ottimo

1.6.6 Dimostrazione matematica per assurdo della sottostruttura ottima

Attualmente non ho molta sbatti di trascriverla, quindi rimanderò a domani o dopo questa infausta operazione.

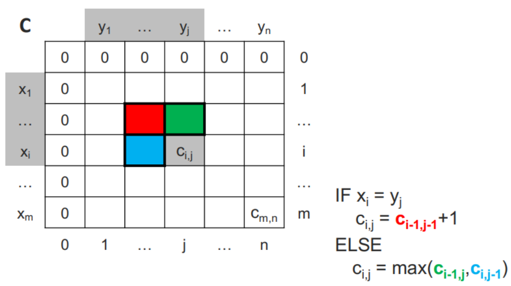
1.6.7 Risoluzione Bottom-Up

- Si calcolano i coefficienti $c_{i,j}$ per dimensione (i, j) crescente a partire dai casi base $(0, j)$ e $(i, 0)$
- Si memorizza $c_{i,j}$ ogni volta che si risolve il sottoproblema (i, j)
- Quando si arriva a calcolare $c_{m,n}$ si ha il valore ottimo

Algoritmo DP (bottom-up)

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

Matrice



Notiamo subito che la prima riga e prima colonna vengono inizializzate a 0, dato che la LCS tra una qualsiasi sequenza e una nulla è 0.

Codice riempimento matrice

```
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]
```

Complessità in tempo $\Theta(mn)$

Esempio di matrice riempita

C		2	5	12	2	3	12	1	30	
	0	0	0	0	0	0	0	0	0	0
2	0	1	1	1	1	1	1	1	1	1
10	0	1	1	1	1	1	1	1	1	2
5	0	1	2	2	2	2	2	2	2	3
3	0	1	2	2	2	3	3	3	3	4
1	0	1	2	2	2	3	3	4	4	5
12	0	1	2	3	3	3	4	4	4	6
	0	1	2	3	4	5	6	7	8	

$c_{6,8} \rightarrow 4$

X = ⟨2, 10, 5, 3, 1, 12⟩

Y = ⟨2, 5, 12, 2, 3, 12, 1, 30⟩

1.6.8 Ricostruzione della soluzione ottimale

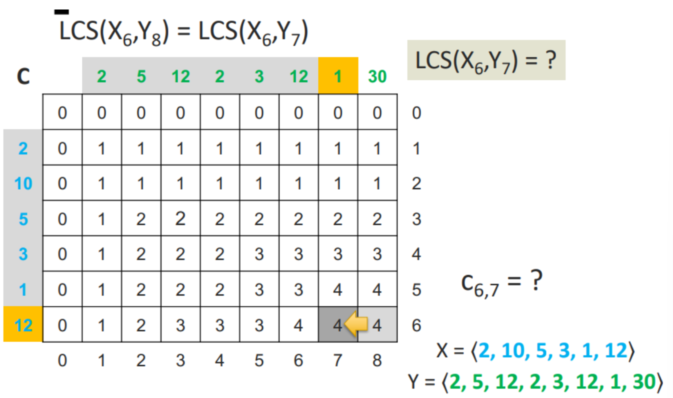
Ora che abbiamo la matrice dei coefficienti abbiamo la LCS, o meglio il valore della LCS (quanto è lunga la più lunga sottosequenza di caratteri in comune fra le 2 sequenze), ma non abbiamo la stringa! É necessario ricostruire tramite la matrice la mia soluzione ottimale.

La procedura Partiamo dall'ultima cella in basso a destra (cella del valore ottimo), che nell'esempio riporta il valore di 4:
Dobbiamo scegliere fra 3 possibili celle dove spostarci:

- La cella sinistra - Se contiene il valore maggiore fra i 3
- La cella in alto - Se contiene il valore maggiore fra i 3
- La cella diagonale - Se le celle hanno lo stesso valore decrementato di 1 rispetto a quello della cella di partenza

Quando mi muovo in diagonale salvo il carattere di riferimento della colonna (o riga, tanto saranno uguali) e proseguo con la procedura.
La procedura consiste nel calcolare la LCS guardando i coefficienti della matrice.

Inizio ricostruzione Partendo da 4 mi chiedo: Quale è la $LCS(X_6, Y_8)$? Guardo i coefficienti della matrice, quale valore fra 4, 4 scelgo? Qua non posso andare in diagonale perchè non ho un decremento del coefficiente di 1, posso andare solo a sinistra o in alto. Scelgo per esempio di andare a sinistra. Non essendoci stato un decremento del coefficiente, cioè non ho valori uguali sugli indici della matrice, infatti $X_6 \neq X_8$, non ho un carattere appartenente alla soluzione ottima, quindi avrò $LCS(X_6, Y_8) = LCS(X_6, Y_7)$, quindi mi sposto e cerco la $LCS(X_6, X_7)$.



Nel caso in cui la cella di partenza e le altre 3 fossero uguali si può scegliere di andare o in alto o a sinistra, si ottiene comunque la stessa soluzione, nel caso in cui si ottengono spostamenti diagonali differenti significa che si avrà un'altra soluzione ottima.
Proseguo e noto che anche qua ho due coefficienti uguali, scelgo di andare in alto, $LCS(X_6, X_7) = LCS(X_5, Y_7)$.

Osservazione Se fossi andato a sinistra al passaggio successivo avrei avuto un passaggio a una diagonale che non percorrerò andando in alto, questa è un'altra soluzione ottima.
Ora mi trovo a calcolare $LCS(X_5, X_7)$. Qua ho tutti e 3 i valori uguali decrementati di 1, infatti ho $X_5 = X_7$, posso effettuare uno spostamento diagonale! Si tratta di un carattere della soluzione ottima quindi avrò che:

$$c_{5,7} = c_{4,6} + 1$$
$$LCS(X_5, Y_7) = LCS(X_4, Y_6) + < x_5 >$$
$$LCS(X_6, Y_8) = LCS(X_5, Y_7)$$
$$LCS(X_6, Y_8) = LCS(X_4, Y_6) + < x_5 >$$
$$LCS(X_6, Y_8) = LCS(X_4, Y_6) + < 1 >$$

Dove 1 è l'ultimo carattere della LCS.
Proseguo a ritrovo fino a quando arriverò in cima alla matrice e avrò il caso base $LCS(X_0, Y_0) = \epsilon$.
Ora ho la sottosequenza ottima!

1.6.9 Algoritmo ricorsivo ricostruzione soluzione ottima

Input Matrice C e indici i e j di una cella di C.

Output Una soluzione ottimale per i prefissi X_i e Y_j.

```
Procedura ricostruisci_LCS(C, i, j)
    if i>0 and j>0 then
        if x_i = y_j then
            ricostruisci_LCS(C, i-1, j-1)
            print x_i
        else
            if C[i,j] = C[i-1,j] then
                ricostruisci_LCS(C, i-1, j)
            else
                ricostruisci_LCS(C, i, j-1)
```

Algoritmo di ricostruzione LCS iterativo

```
List Ricostruisci_LCS(C, m, n)
    j = m
    j = n
    LCS = empty list
    while i > 0 and j > 0 do
        if x_i = y_i then
            aggiungi x_i in testa a LCS
            i = i-1
            j = j-1
        else
            if C[i,j] = C[i-1,j] then
                i = i-1
            else
                j = j-1
    return LCS
```

Capitolo 2

LIS - Longest Increasing Subsequence

Una Increasing Sequence (IS) è definita nel seguente modo:

$$Z =< z_1, z_2, \dots, z_k >$$

Tale che $z_i < z_{i+1}$ per ogni indice i da 1 a k-1.

Esempi

$< 2, 4, 7, 13, 21 > \rightarrow$ è una sequenza crescente
$< 2 > \rightarrow$ sequenza crescente
$< 2, 4, 13, 13, 21 > \rightarrow$ NON è una sequenza crescente
$< 10, 10, 10 > \rightarrow$ NON è una sequenza crescente

LIS di $X =< x_1, x_2, \dots, x_m >$ più lunga sottosequenza di X che sia crescente.
$Z = LIS(X)$.

2.1 Esempio di LIS di X

$$X =< 14, 2, 4, 2, 7, 0, 13, 21, 11 >$$

$$LIS(X) =< 2, 4, 7, 13, 21 >$$

LIS è la più lunga sottosequenza di X che sia crescente.

2.1.1 Altro esempio

$$X =< 5, 4, 3, 2, 1 >$$
$$LIS(X) =< 5 > \text{ or } < 4 > \text{ or } < 3 > \text{ or } < 2 > \text{ or } < 1 >$$
$$X =< 10, 10, 10, 10 >$$
$$LIS(X) = 10$$

2.2 Definizione formale e identificazione tipologia problema

P: Data una sequenza $X =< x_1, x_2, \ldots, x_m >$ trovare la più lunga sottosequenza crescente $Z = LIS(X)$.
P è un problema di ottimizzazone di massimo, dove:

- (m) $\rightarrow$ dimensione del problema
- Soluzioni possibilie $\rightarrow$ tutte le sottosequenze crescenti di X
- Funzione obiettivo $\rightarrow$ lunghezza
- $|Z|$ è il valore ottimo del problema
- Z è una soluzione ottimale

2.3 Soluzione

1. Calcolo dell'ottimo (lunghezza di Z)
2. Ricostruzione di una soluzione ottimale

2.3.1 Definizione dei sottoproblemi - Primo tentativo

Sottoproblema di dimensione (i)
Trovare la LIS del prefisso $X_i \rightarrow LIS(X_i)$.
Numero di sottoproblemi: m+1. Mentre per $i = m$ abbiamo il problema principale.

2.3.2 Casi Base

Sottoproblema di dimensione 0.

$$i = 0 \implies LIS(X_0) =<>$$

2.3.3 Passo ricorsivo?

$\rightarrow$ Tutti i sottoproblemi di dimensione (i) tale per cui $i > 0$.
Qual è la sottostruttura ottima del problema?
Proviamo la seguente sottostruttura:

$$LIS(X_m) = max\{LIS(X_{m-1}), LIS(X_{m-1}) + < x_m >\}$$

$$X =< x_1, x_2, \ldots, x_{m-1}, x_m >$$

$LIS(X_m) = LIS(X_{m-1}) + < x_m >?$

- Sì, se $LIS(X_{m-1})$ finisce con un elmento $< x_m >$
- No, se $LIS(X_{m-1})$ finisce con un elemento $\geq x_m$

2.3.4 Test sottostruttura - Primo tentativo

Verifichiamo se la sottostruttura definita poco fa ci permette di trovare le LIS in una stringa.

Esempio $X =< 14, 2, 4, 2, 7, 14, 15, 0, 13 >$ $m = 9$
$LIS(X_m) = LIS(X_{m-1}) + < x_m >?$ $x_m = 13$
$LIS(X_9) = LIS(X_8) + < 13 >?$ $X =< 14, 2, 4, 2, 7, 14, 15, 0, 13 >$
$LIS(X_9) = LIS(X_8) + < 13 >?$
NO! Perchè risulta uguale a $< 2, 4, 7, 14, 15 > + < 13 >=< 2, 4, 7, 14, 15, 13 >$ e $15 > 13$, questo significa che la sottostruttura non mi permette di avere una LIS.
Questo significa indica che mi manca dell'informazione, è necessario definire un problema ausiliario che ci permette di creare una sottostruttura che trovi una LIS valida.

2.4 Problema Ausiliario

Si tratta di un sottoproblema di dimensione (i) che ha lo scopo di trovare la LIS_v del prefisso $X_i \rightarrow LIS_v(X_i)$, $i \in \{1, 2, \ldots, m\}$.
Numero di sottoproblemi: m.
$LIS_V(X_m)$ $\rightarrow$ è la soluzione ottimale del problema ausiliario.
$LIS(X) \rightarrow max\{LIS_V X_i$ t.c. $1 \leq i \leq m\}$
Sostanzialmente cerco la LIS partendo dalla stringa base di 1 carattere, mano a mano incremento di 1 e ricalcolo la LIS, fino ad arrivare a calcolare la LIS di tutta la stringa.

Esempio

$$
\begin{aligned}
&X =< 14, 2, 4, 2, 7, 14, 15, 0, 13 > \\
&LISV(X1) =< 14 > \\
&LISV(X3) =< 2, 4 > \\
&LISV(X2) =< 2 > \\
&LISV(X5) =< 2, 4, 7 > \\
&LISV(X6) =< 2, 4, 7, 14 > \\
&LISV(X7) =< 2, 4, 7, 14, 15 > \quad \text{LIS(X)} \\
&LISV(X8) =< 0 > \\
&LISV(X9) =< 2, 4, 7, 13 > \\
&LISV(X4) =< 2 >
\end{aligned}
$$

2.4.1 Casi Base

→ Sottoproblema di dimensione (1).

$$i = 1 \implies LIS_V(X_1) =< x_1 >$$

2.4.2 Passo Ricorsivo

→ Tutti i sottoproblemi di dimensione (i) tale per cui $i > 1$.
Qual è la sottostruttura ottima del problema Ausiliario?

Esempio sottostruttura ottima

$$X =< 14, 2, 4, 2, 7, 14, 15, 0, 13, 21, 8 > m = 11$$

$LIS_V(X_{11}) \rightarrow Z =< z_1, z_2, \dots, z_{k-1}, z_k >=< 2, 4, 7, 8 >$
$z_k = x_{11} = 8$
$< z_1, z_2, \dots, z_{k-1} > \rightarrow$ sottosequenza di X_{10} per cui $z_{k-1} = x_5 = 7, \quad z_{k-1} < z_k$

Ottengo quindi che $< z_1, z_2, \dots, z_{k-1} >= LIS_V(X_5) + z_k =< 2, 4, 7, 8 >$

$$LIS_V(X_5) = max\{LIS_V(X_h) \text{ t.c } 1 \leq h \leq 11, x_h < x_1 1\}$$
$$LISV(X_1 1) = max\{LIS_V(X_h) \text{ t.c. } 1 \leq h < 11, x_h < x_1 1\} + < x_1 1 >$$

2.4.3 Definizione sottostruttura ottima

Data la sequenza $X =< x_1, x_2, \ldots, x_m >$ e $LIS_V(X_m) =< z_1, z_2, \ldots, x_{k-1}, z_k >= Z_{k-1}+ < z_k >$:

$$
\begin{aligned}
&z_k = x_m \\
&Z_{k-1} = max\{LIS_V(X_h) \text{ t.c. } 1 \leq< m, x_h < x_m\} \\
&\text{ con max}\{\emptyset\} =<>
\end{aligned}
$$

2.4.4 Passo ricorsivo (Problema Ausiliario - PA)

Data la sequenza $X =< x_1, x_2, \ldots, x_m >$:

$$
\begin{aligned}
&LIS_V(X_m) = max\{LIS_V(X_h) \text{ t.c } 1 \leq h < m, x_h < x_m\}+ < x_m > \\
&\text{ con max}\{\emptyset\} =<> \text{ che implica } LIS_V(X_m) =< x_m >
\end{aligned}
$$

Dato il prefisso $X_i =< x_1, x_2, \ldots, x_i >$:

$$
\begin{aligned}
&LIS_V(X_i) = max\{LIS_V(X_h) \text{ t.c } 1 \leq h < i, x_h < x_i\}+ < x_i > \\
&\text{ con max}\{\emptyset\} =<> \text{ che implica } LIS_V(X_i) =< x_i >
\end{aligned}
$$

2.4.5 Equazioni di ricorrenza - PA

i=1 (Caso Base) $LIS_V(X_1) =< x_1 >$

$i > 1$ **(Passo Ricorsivo)**

$$
\begin{aligned}
&LIS_V(X_i) = max\{LIS_V(X_h) \text{ t.c. } 1 \leq h < i,\, x_h < x_j\}+ < x_i > \\
&\text{ con max}\{\emptyset\} =<> \text{ che implica } LIS_V(X_i) =< x_i >
\end{aligned}
$$

Definizione dei coefficienti - PA

Coefficiente c_i del sottoproblema (i):

$$
\begin{aligned}
&c_i = |LIS_V(X_i)| \\
&i \in \{1, 2, \ldots, m\}
\end{aligned}
$$

Numero di coefficienti: m.
$c_m = |LIS_V(X_m)| \rightarrow$ valore ottimo di PA.

2.4.6 Sostituzione coefficienti nelle equazioni di ricorrenza

i=1 (Caso Base) $c_i = 1$

$i > 1$ **(Passo Ricorsivo)**

$$c_i = max\{c_h \text{ t.c. } 1 \leq h < i,\ x_h < x_i\} + 1$$
$$\text{con } \max\{\emptyset\} = 0 \text{ che implica } c_i = 1$$

2.4.7 Calcolo dell'ottimo (PA)

- Si calcolano i coefficienti c_i per dimensione (i) crescente a partire dal caso base per $i = 1$
- Si memorizza c_i ogni volta che si risolve il sottoproblema (i)
- Quando si arriva a calcolare c_m si ha il valore ottimo del problema ausiliario $\rightarrow |LIS_V(X)|$
- Il valore ottimo del problema principale è dato da $max\{c_i$ t.c. $1 \leq i \leq m\} \rightarrow |LIS(X)$

2.4.8 Algoritmo DP (Bottom-up)

- Costruzione del vettore $C[1 \dots m]$
- Riempimento di C in modo tale che $C[i] = c_i$
- Valore ottimo $\rightarrow$ massimo in C

2.4.9 Algoritmo DP (bottom-up)

```
int calcolo_ottimo_LIS(X)
     C[1] = 1
     H[1] = 0
     valore_ottimo = C[1]
     for i from 2 to m do
          max = 0
          H[i] = 0
          for h from 1 to i-1 d
               if x_h < x_i and C[h] > max then
```

```
                max = C[h]
                H[i] = h
        C[i] = 1 + max
        if C[i] > valore_ottimo then
            valore_ottimo = C[i]
    return valore_ottimo
```

2.4.10 Ricostruzione soluzione ottimale

Mi basta leggere la matrice ausiliaria a partire dal valore massimo e vedere quando non ho 0 scrivere quello che ho in corrispondenza della matrice.

X = ⟨14, **2**, **4**, 2, **7**, 0, **13**, **21**, 20⟩

	14	**2**	**4**	2	**7**	0	**13**	**21**	20
C	1	1	2	1	3	1	4	5	5
H	0	0	2	0	3	0	5	7	7
	1	**2**	3	4	5	6	7	8	9

LIS(X) = ⟨2, 4, 7, 13, 21⟩

2.4.11 Algoritmo di Ricostruzione - Ricorsivo

```
Procedura ricostruisci_LIS_V(H, i)
    if H[i] != 0 then
        ricostruisci_LIS_V(H, H[i])
    print x_i
```

Chiamando la procedura per i_{max} (posizione della cella di C che contiene il valore massimo) si ottiene la stampa di una soluzione ottimale di LIS(X).
Il caso peggiore sarà quando ho una stringa crescente, il tempo sarà di $O(m)$.
Il caso migliore invece è quando la stringa è decrescente, si ferma subito alla prima cella. Avrò infatti T costante.

Capitolo 3

Knapsack Problem 0/1 - Il Problema dello Zaino 0/1

Questione da risolvere: trovare il subset di oggetti di massimo valore complessivo che non superi la capacità C.

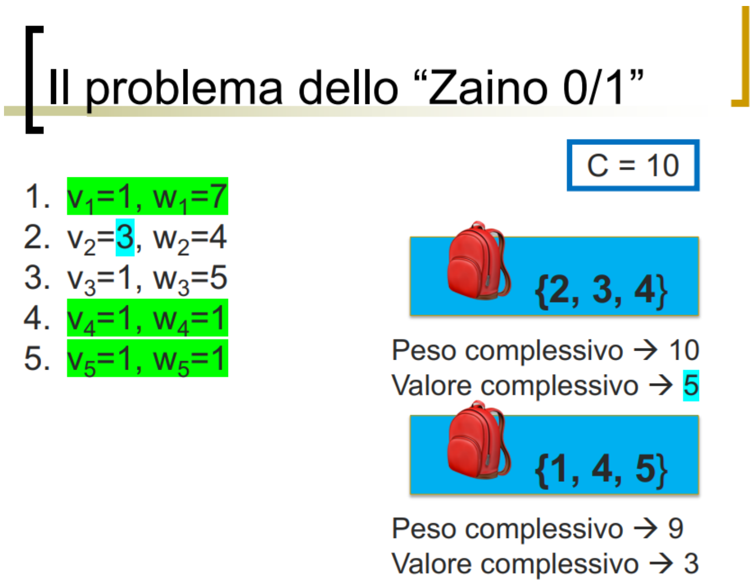
Oggetti Ad ogni oggetto viene associato un peso e un valore, quindi il problema consiste nel inserire nello zaino il massimo valore possibile senza superare il peso massimo.

3.1 Istanza del problema

Insieme di n oggetti $\{1, 2, \ldots, i, \ldots, n\}$:

- C $\rightarrow$ Capacità dello zaino
- v_n $\rightarrow$ Valore dell'oggetto n
- w_n $\rightarrow$ peso/ingombro dell'oggetto n

Esempio



3.2 Definizione formale

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

- $V : X \to N$ tale che $V(i) = v_i$ è il valore dell'oggetto i
- $W : X \to N$ tale che $W(i) = w_i$ è il peso dell'oggetto i

si vuole trovare un sottoinsieme $S = \{i_1, i_2, \ldots, i_k\}$ di X tale per cui:

$$W_S = \sum_{j=1}^{k} w(i_j) \leq C$$

$$V_S = \sum_{j=1}^{k} v(i_j)$$

Dove V_S è il massimo tra tutti i valori dei possibili sottoinsiemi di X che sono "compatibili" con lo zaino.
Si tratta dunque di un problema di ottimizzazione di massimo, dove:

- (n,C) $\to$ dimensione del problema
- Soluzioni possibili $\to$ tutti i sottoinsiemi S' di X il cui peso totale $W_{S'}$ è al più la capacità C dello zaino

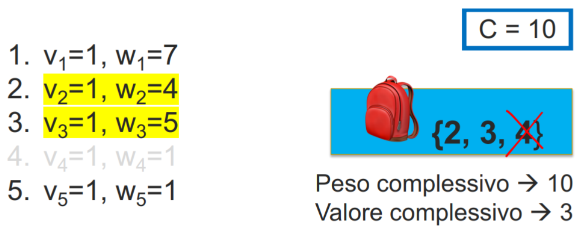
- Funzione obiettivo $\rightarrow$ valore totale $V_{S'}$ della soluzione possibile S'.
- Valore totale di S $\rightarrow$ valore ottimo
- S $\rightarrow$ Soluzione ottimale

3.2.1 Soluzione del problema DP

1. Calcolo del valore ottimo (valore totale di S)
2. Ricostruzione di una soluzione ottimale (un insieme S)

3.3 Sottostruttura Ottima

Consideriamo l'esempio di inizio capitolo:



$\{2,3\}$ è una soluzione ottimale di $X \setminus \{4\}$? NO.
$\{2,3,5\}$ è una soluzione ottimale di $X \setminus \{4\}$!

3.3.1 Diamo un ordine agli oggetti

Diamo un ordine agli oggetti all'interno di X, cioè:
1 viene prime di 2 che viene prima di 3, etc. che viene prima dell'ultimo oggetto n.

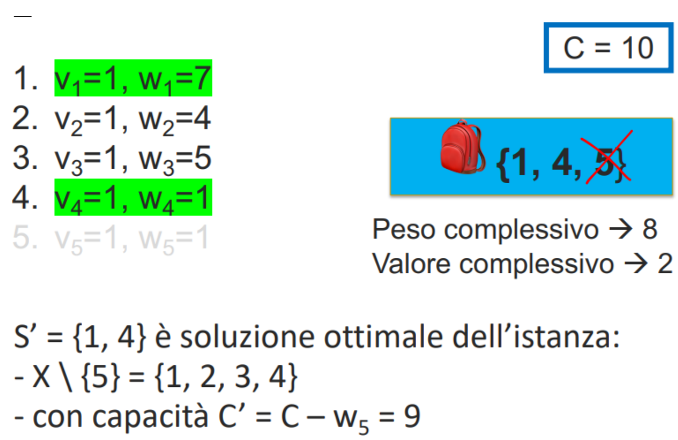
Data una soluzione ottima S si può verificare

- **CASO 1**: l'oggetto n appartiene a S
- **CASO 2**: l'oggetto n NON appartiene a S

CASO 1 $C \geq w_n$ e l'oggetto n appartiene a S

NB: $S' = S \setminus \{n\}$ non è necessariamente la soluzione ottimale dell'istanza per $X \setminus \{n\}$ e capacità C.
Infatti, se esiste $i \in X \setminus \{n\}$ t.c $i \notin S'$ e $w_i \leq C - W_{S'} \implies S' \cup \{i\}$ ha valore totale maggiore di quello di S'.
Tornando a noi, CASO 1 implica che $\implies S' = S \setminus \{n\}$ è soluzione ottimale per l'istanza data da:

- insieme di oggetti $X \setminus \{n\} = \{1, 2, \ldots, n-1\}$
- Zaino di capacità C' pari a $C - w_n$



PROOF

1. S' è compatibile con la capacità $C - w_n$
   $W_S \leq C \implies W_S - w_n \leq C - w_n \implies W_{S'} \leq C - w_n$

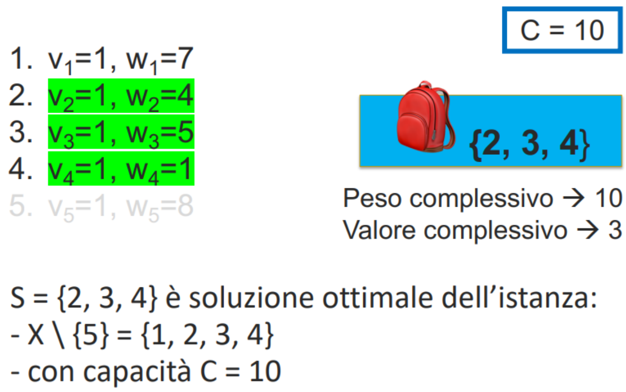
2. S' ha valore totale $V_{S'}$ massimo
   Se esiste S'' tale che $V_{S''} > V_{S'}$ e $W_{S''} \leq C - w_n$
   $\implies S'' \cup \{n\}$ è soluzione ottimale per l'istanza X e C di valore totale maggiore di V_S (contro l'ipotesi)

CASO 2 - L'oggetto n NON appartiene a S

$\implies$ S è soluzione ottimale per l'istanza data da:

- insieme di oggetti $X \setminus \{n\} = \{1, 2, \ldots, n-1\}$
- Zaino di capacità pari a C

Non avendo aggiunto l'oggetto allo zaino la capacità totale rimane invariata.



PROOF

Se esiste S' che è soluzione ottimale per $X \setminus \{n\}$ e capacità C e con valore $V_{S'} > V_{S'}$ allora S' sarà soluzione ottimale per X e capacità C (contro l'ipotesi).

3.3.2 Sottostruttura ottima

Dato un insieme $X = \{1, 2, \ldots, n\}$ di n oggetti, uno zaino di capacità C e una soluzione ottimale S:
$C \geq w_n$, $n \in S \implies S' = S \setminus \{n\}$ è soluzione ottimale per:

- insieme di oggetti $\{1, 2, \ldots, n-1\}$
- capacità $C - w_n$

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

$S = S' \cup \{n\}$ con S' soluzione ottimale per le condizioni viste prima del box.
$n \notin S \implies S$ è soluzione ottimale per:

- insieme di oggetti $\{1, 2, \ldots, n-1\}$
- capacità C

Problema $(n-1, C)$

$S = S''$ con S'' soluzione ottimale per le condizioni viste prima del box.

3.3.3 Passo ricorsivo per (n,C)

Dato un insieme $X = \{1, 2, \ldots, n\}$ di n oggetti e uno zaino id capacità C, la soluzione ottimale $S_{n,C}$ è:
Se $C \geq w_n$ **allora**

- $S_{n,C} = max_V\{S' \cup \{n\}, S''\}$

altrimenti

- $S_{n,C} = S''$

S' soluzione ottimale per il problema $(n-1, C-w_n) \rightarrow S_{n-1,C-w_n}$
S'' soluzione ottimale per il problema $(n-1, C) \rightarrow S_{n-1,C}$ Sostituisco nelle equazioni S' e S'': **Se** $C \geq w_n$ **allora**

- $S_{n,C} = max_V\{S_{n-1,C-w_n} \cup \{n\}, S_{n-1,C}\}$

altrimenti

- $S_{n,C} = S_{n-1,C}$

3.3.4 Definizione dei sottoproblemi

Sottoproblema di dimensione (i,c) Riempire uno zaino di capacità c con oggetti dall'insieme $\{1, 2, \ldots, i\} \rightarrow S_{i,c}$
$i \in \{1, 2, \ldots, n\}$
$c \in \{0, 1, \ldots, C\}$

Numero di sottoproblemi: $(n + 1)(C + 1)$

$i = n$, $c = C$ $\rightarrow$ problema principale

3.4 Equazioni di ricorrenza

CASI BASE $\rightarrow$ Tutti i sottoproblemi di dimensione (i, c) tale per cui i = 0, oppure C = 0.

$$S_{i,C} = \emptyset$$

PASSO RICORSIVO $\rightarrow$ Tutti i sottoproblemi di dimensione (i, c) tale per cui $i > 0$ e $c > 0$.
Se $C \geq w_n$ **allora**

- $S_{n,C} = max_V\{S_{n-1,C-w_n} \cup \{n\}, S_{n-1,C}\}$

altrimenti

- $S_{n,C} = S_{n-1,C}$

Convertito in Equazioni di Ricorrenza:
$i = 0 \vee c = 0$ **(CASI BASE)**
$S_{i,c} = \emptyset$ $i > 0 \wedge c > 0$ **(PASSO RICORSIVO)**
$c \geq w_i \implies S_{i,c} = max_V\{S_{i-1,C-w_i} \cup \{i\},\ S_{i-1,C}\}$
$c < w_i \implies S_{i,c} = S_{i-1,C}$

$d_{i,c} \rightarrow$ valore totale di $S_{i,c}$

3.4.1 Sostituzione coefficienti

Sostituendo quindi $d_{i,c}$ a $S_{i,c}$ otteniamo: $i = 0 \vee c = 0$ **(CASI BASE)**
$d_{i,c} = \emptyset$ $i > 0 \wedge c > 0$ **(PASSO RICORSIVO)**
$c \geq w_i \implies d_{i,c} = max_V\{d_{i-1,C-w_i} \cup \{i\},\ d_{i-1,C}\}$
$c < w_i \implies d_{i,c} = d_{i-1,C}$

3.4.2 Calcolo dell'ottimo

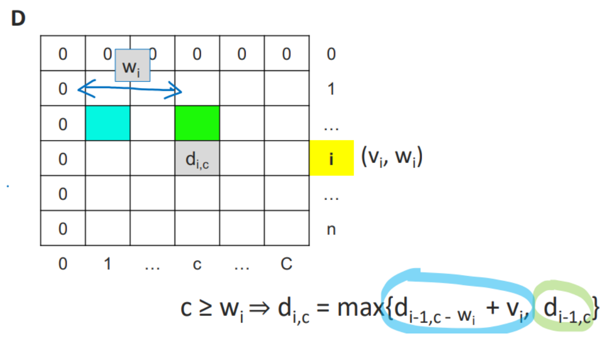
- Si calcolano i coefficienti $d_{i,c}$ per dimensione (i, c) crescente a partire dai casi base per $i = 0$ e $c = 0$
- Si memorizza $d_{i,c}$ ogni volta che si calcola l'ottimo per il sottoproblema (i, c)
- Quando si arriva a calcolare $d_{n,C}$ si ha il valore ottimo del problema (n, C)

3.5 Algoritmo DP (bottom-up)

1. Costruzione di una 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 $\rightarrow D[n, C]$

Avrò quindi la seguente matrice:



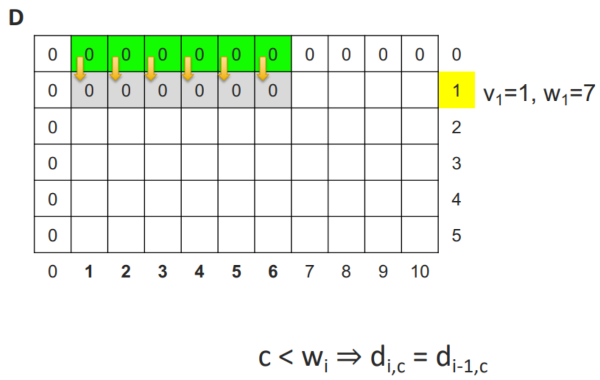
Dove sulle righe avrò il numero di oggetto con associato relativo valore v_i e peso w_i.
Sulle colonne avrò capacità da 0 a C, che rappresenteranno i vari sottoproblemi, gli zaini più piccoli.
Ricordiamo che abbiamo i seguenti oggetti:

1. $v_1 = 1$, $w_1 = 7$
2. $v_2 = 1$, $w_2 = 4$
3. $v_3 = 1$, $w_3 = 5$
4. $v_4 = 1$, $w_4 = 1$
5. $v_5 = 1$, $w_5 = 1$

Capacità $C = 10$.

3.5.1 Riempimento matrice

Prima di tutto riempio la prima riga e colonna con 0 dato che rappresentano l'oggetto 0 e il sottoproblema con $C = 0$, sono i casi base.
Prendo il primo oggetto:

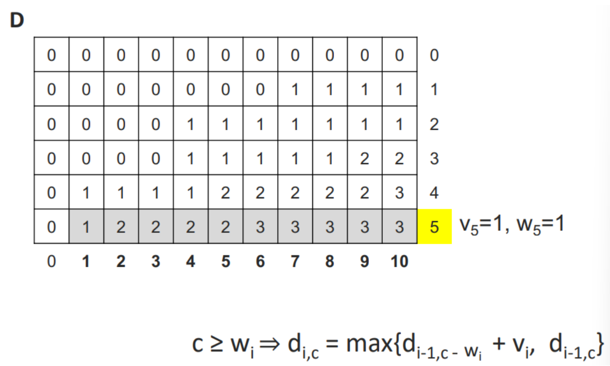


D

0	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0					1	v_1=1, w_1=7
0											2	
0											3	
0											4	
0											5	
0	1	2	3	4	5	6	7	8	9	10		

$$c < w_i \Rightarrow d_{i,c} = d_{i-1,c}$$

Notiamo che riempiamo di 0 la riga fino a quando non abbiamo $w_1 = c_i = 7$. Qua controllo quale sia il massimo tra $d_{i-1,c-w_i} + v_i$ e $d_{i-1,c}$, cioè controllo se mi conviene riempire lo zaino inserendo l'oggetto che sto considerando sommato con il valore dell'oggetto precedente (considerando $c - w_i$, quindi un peso che non superi la capacità), oppure mi conviene riempire lo zaino con la riga precedente perchè ha valore maggiore rispetto all'aggiunta del mio oggetto.
Proseguo così fino a riempire tutta la matrice, il valore ottimo sarà in fondo alla matrice a destra (come per LCS).



D

0	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	1	1	1	1	1	
0	0	0	0	1	1	1	1	1	1	1	2	
0	0	0	0	1	1	1	1	1	2	2	3	
0	1	1	1	1	2	2	2	2	2	3	4	
0	1	2	2	2	2	3	3	3	3	3	5	v_5=1, w_5=1
0	1	2	3	4	5	6	7	8	9	10		

$$c \geq w_i \Rightarrow d_{i,c} = \max\{d_{i-1,c-w_i} + v_i,\ d_{i-1,c}\}$$

Capitolo 4

Problema dei cammini minimi - Floyd-Warshall

Come al solito diamo qualche definizione per poter lavorare successivamente in maniera agile.

4.1 Definizioni

4.1.1 Grafo

Un Grafo viene definito come $G = (V, E)$ dove:

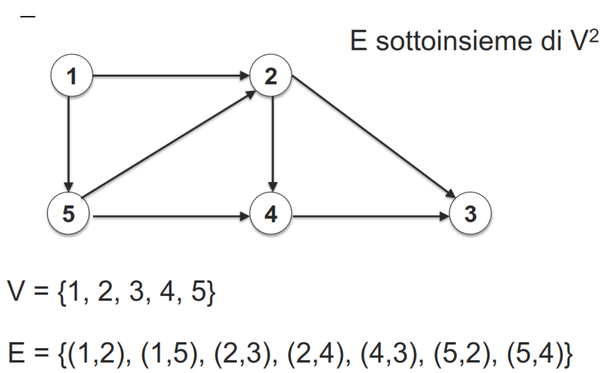
- $V = \{v_1, v_2, v_3, ..., v_n\}$ insieme di vertici
- $E = \{e_1, e_2, e_3, ..., e_m\}$ insieme di archi

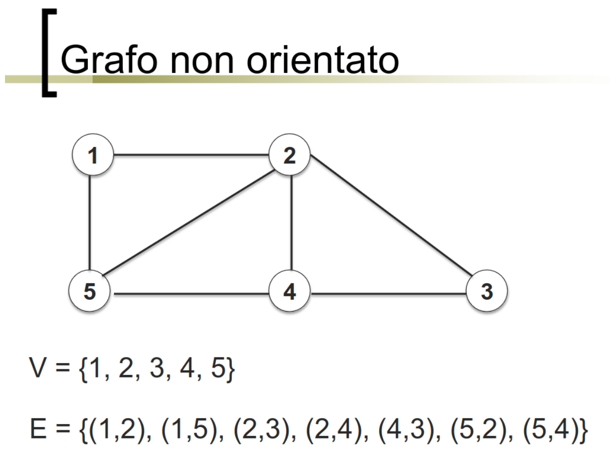
Dimensione di G $\rightarrow$ (n,m). Arco $e_k \rightarrow$ relazione R tra due vertici v_i e v_j

R può essere

- Simmetrica - Grafo NON Orientato - cioè $v_i \, R \, v_j \Leftrightarrow v_j \, R \, v_i$
- Asimmetrica - Grafo Orientato (o diretto) - cioè $v_i \, R \, v_j \not\Leftrightarrow v_j \, R \, v_i$

Un grafo orientato è caratterizzato da un verso di percorrenza degli archi unidirezionale. In questo caso E è sottoinsieme di V^2.





4.1.2 Adiacenza

Un vertifica v è adiacente a un vertice u se $(u, v) \in E$.
Per esempio nella rappresentazione del grafo orientato il vertice **1** è adiacante ai vertici **2 e 5**, infatti notiamo che in E è presente $(1, 2), (1, 5)$.

4.1.3 Rappresentazione di un grafo

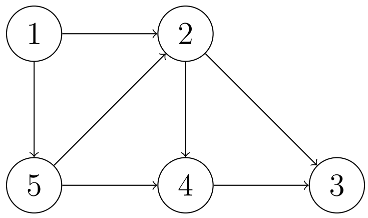
Abbiamo 2 rappresentazioni possibili:

- Liste di adiacenza
- Matrice di adiacenza

1. Le liste di adiacenza utilizzano un vettore L_v di dimensione $|V|$ tale che $V[i]$ è la lista degli adiacenti del vertice v_i. Ogni vertice del grafo avrà un vettore.

2. La matrice di adiacenza è una Matrice M_v di dimensione $n \times n$ tale che $M[i, j] = 1$ se il vertice j è adiacente del vertice i, altrimenti $M[i, j] = 0$. A differenza delle liste in questo caso ho una sola matrice.

4.1.4 Esempio grafo orientato

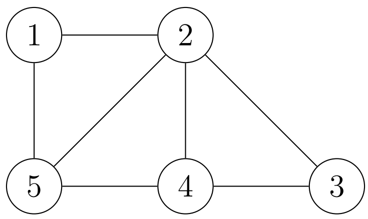


$$M = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 1 & 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 & 1 & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 \\ 4 & 0 & 1 & 0 & 0 & 0 \\ 5 & 1 & 0 & 1 & 0 & 0 \end{array}$$

Dimensione $|V|^2 = n^2$

Numero di celle con 1 $|E|$

4.1.5 Esempio grafo non orientato



$$M = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 1 & 0 & 0 & 1 \\ 2 & 1 & 0 & 1 & 1 & 1 \\ 3 & 0 & 1 & 0 & 1 & 0 \\ 4 & 0 & 1 & 1 & 0 & 1 \\ 5 & 1 & 1 & 0 & 1 & 0 \end{array}$$

Dimensione $|V|^2 = n^2$

Numero di celle con 1 $2|E|$

4.1.6 Liste VS Matrice (memoria)

Liste di adiacenza Sono ottime dal punto di vista dell'occupazione dello spazio nel caso di Grafi sparsi con $|E|$ molto minore di $|V|^2$.

Matrici di adiacenza Risultano migliori nei grafi densi quindi quando ho $|E|$ che si avvicina a $|V|^2$.

4.1.7 Liste VS Matrice (tempo)

(u,v) Intendiamo se i 2 vertici sono collegati. Come tempo intendiamo il tempo per stabilire se (u,v) appartiene ad E e i tempi sono i seguenti:

- Liste di adiacenza $\rightarrow O(|E|) = O(m)$
- Matrice di adiacenza $\rightarrow O(1)$

4.1.8 Cammino in un grafo orientato

Definizione di cammino Sequenza $P =< v_{i_1}, v_{i_2}, ..., v_{i_{k-1}}, v_{i_k} >$ tale che v_{i_k} appartiene a V per $1 \leq j \leq k$ e $(v_{i_j}, v_{i_{j+1}})$ appartiene ad E per $1 \leq j < k$.

Lunghezza del cammino $k - 1$ (numero di archi)

Ciclo Cammino in cui v_{i_1} coincide con v_{i_k}

Cammino semplice Cammino in cui ogni vertice è presente una volta sola (cioè non contiene cicli)

Predecessore di v_{i_k} in P Vertice di $v_{i_{k-1}}$

4.1.9 Grafo orientato pesato

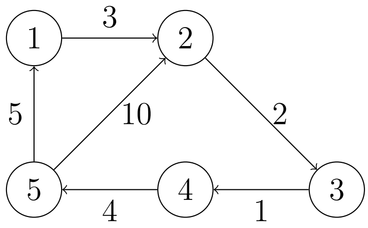
Grafo $G = (V, E, W)$

- $V = \{v_1, v_2, v_3, ..., v_n\}$ insieme di vertici
- $E = \{e_1, e_2, e_3, ..., e_m\}$ insieme di archi

- $W : E \rightarrow R$ tale che $W(v_i, v_j) = w_{ij}$ è il peso dell'arco (v_i, v_j)

Peso di un cammino Si tratta della somma dei pesi di tutti gli archi, formalmente: $P =< v_{i_1}, v_{i_2}, ..., v_{i_k} > \rightarrow \sum_{j=1}^{k-1} w(v_{i_j}, v_{i_{j+1}})$

4.1.10 Esempio grafo orientato pesato



4.2 Il problema dei cammini minimi

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

Output Per ogni coppia di vertici i e j, trovare il cammino di peso minimo (cammino minimo) che parte da i e finisce in j.
Si tratta di un problema di ottimizzazione di minimo, dove

- (n) $\rightarrow$ dimensione del problema
- Soluzioni possibili per una coppia di vertici i e j sono tutti i cammini da i a j
- Funzione obiettivo è il peso del cammino
- Peso del cammino minimo da i a j è il valore ottimi (per i e j)
- Un cammino minimo tra i vertici i e j è la soluzione ottimale

4.2.1 L'input

Funzione peso W $W : E \rightarrow R^+$ tale che $W(i, j) = w_{ij}$ = peso dell'arco (i, j).

Funzione peso W - Versione estesa $W : V \times V \to R^+$ tale che $W(i,j) = w_{ij}$ con:

- $w_{ij} = 0$ se $i = j$
- $w_{ij} =$ peso dell'arco (i,j), se $(i,j) \in E$
- $w_{ij} = \infty$, se $i \neq j$ e $(i,j) \notin E$

Matrice $W = [w_{ij}]$ di n righe e n colonne.

4.2.2 L'output Matrici D e Π

- Matrice $D = [d_{ij}]$ di n righe e n colonne, dove $[d_{ij}]$ è il peso del cammino minimo da i e j
- Matrice $\Pi = [\pi_{ij}]$ di n righe e n colonne, dove π_{ij} è il predecessore di j nel cammino minimo da i a j

Matrice D

- $d_{ij} = 0$ se $i = j$
- $d_{ij} =$ peso del cammino minimo, se esiste un cammino da i a j
- $d_{ij} = \infty$ se non esiste un cammino da i a j

Matrice Π

- $\pi_{ij} = NIL$, se $i = j$
- $\pi_{ij} = u$ appartenente al cammino minimo da i a j, tale che $(u,j) \in E$, se esiste un cammino da i a j
- $\pi_{ij} = NIL$, se non esisto un cammino da i a j

Dopo aver riempito entrambe le matrici mi rendo conto che:
La riga i d Π fornisce l'albero dei predecessori relativo al vertice i.

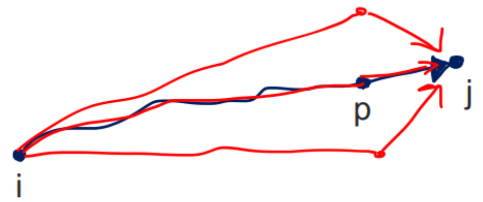
Albero dei predecessori del vertice i (riga i di Π)

- $\{j \in V | \pi_{ij} \notin NIL\} \cup \{i\} \to$ insieme dei vertici
- $(\pi_{ij}, j) | \pi ij \notin NIL$

4.3 Sosttostruttura ottima (primo tentativo)

Consideriamo come P_{ij} il **Cammino minimo da i a j** e p è il predecessore di j.
Sicuramente $P_{ij} = P_{ip} + < j >$, con P_{ip} cammino minimo da i a p.

Con P_{ip} cammino minimo da i a p. Come potrei trovare P_{ij}?



1. Considero tutti i vertici p' tali che $(p', j) \in E$
2. Per ogni vertice p' determino il cammino dato da: $P_{ip} + < j >$
3. Seleziono il cammino di peso minimo

Attenzione! Non è sicuro che quando si calcola P_{ij} si abbiano già a disposizione i cammini $P_{ip'}$. Si deve parametrizzare rispetto alla lunghezza I del cammino:

1. Prima calcolo tutti i cammini minimi a lunghezza $0 \rightarrow P^0_{ij}$
   $P^0_{ij} =< i >$ se $i = j$, altrimenti $P^0_{ij} = \infty$
2. Poi calcolo tutti i cammini minimi a lunghezza $1 \rightarrow P^1_{ij}$
   $P^1_{ij} =< i, j >$ se $i \neq j$ e $(i, j) \in E$, altrimenti $P^1_{ij} = \infty$
3. Poi calcolo tutti i cammini minimi a lunghezza $2 \rightarrow P^2_{ij}$
4. Poi calcolo tutti i cammini minimi a lunghezza $3 \rightarrow P^3_{ij}$
5. ...
6. Ci si ferma per $l = |E| = m$ (l è lunghezza)
7. Per ogni coppia i e j scelgo tra i cammini $P^0_{ij}, P^1_{ij}, \ldots, P^m_{ij}$, quello di peso minimo

Friendly Reminder $< i, j >$ Significa perorso con i vertici i e j.

Domande Come calcolo P_{ij}^l con $l \geq 2$?
E qual è il tempo nel caso peggiore dell'algoritmo DP che sfrutta questa struttura ottima?

4.3.1 Diamo un ordine ai vertici del grafo

1 viene prima di 2 che viene prima di 3 etc. che viene prima dell'ultimo vertice n.
Parametriziamo rispetto ai vertici intermedi del cammino:

- Trovo $P_{ij}^0 \rightarrow$ cammino minimo senza vertici intermedi
- Trovo $P_{ij}^1 \rightarrow$ cammino minimo con vertici intermedi $\in \{1\}$
- Trovo $P_{ij}^2 \rightarrow$ cammino minimo con vertici intermedi $\in \{1, 2\}$
- Trovo $P_{ij}^3 \rightarrow$ cammino minimo con vertici intermedi $\in \{1, 2, 3\}$
- Trovo $P_{ij}^n \rightarrow$ cammino minimo con vertici intermedi $\in \{1, 2, \ldots, n\}$

Analizziamo nel dettaglio i cammini minimi intermedi

$P_{ij}^0 \rightarrow$ cammino minimo senza vertici intermedi.

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

Per $k > 0$, $P_{ij}^k \rightarrow$ cammino minimo con vertici intermedi $\in \{1, 2, \ldots, k\}$.
Per $k = n$, $P_{ij}^n \rightarrow$ cammino minimo con vertici intermedi $\in \{1, 2, \ldots, n\}$.
Quindi $P_{ij}^n \rightarrow$ cammino minimo P_{ik}.

Sottoproblema di dimensione k Per ogni coppia (i,j), trovare il cammino minimo P_{ij}^k dal vertice i al vertice j che ha vertici intermedi $\in \{1, 2, \ldots, k\}$ se $k > 0$, oppure non ha vertici intermedi se $k = 0$.

$$k \in \{0, 1, \ldots, n\}$$
$$i \in \{1, \ldots, n\}$$
$$j \in \{1, \ldots, n\}$$

Numero di sottoproblemi $n \times n \times (n+1)$.
$k = n \rightarrow P_{ij}^n = P_{ij}$.

4.3.2 Equazioni di ricorrenza

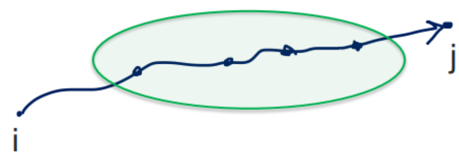
Casi base Sottoproblema di dimensione (0)

$$\begin{aligned} &P_{ij}^0 =< i > \quad \text{se } i = j \\ &P_{ij}^0 =< i, j > \quad \text{se } i \neq j \text{ e } (i,j) \in E \\ &P_{ij}^0 = NIL \quad \text{se } i \neq j \text{ e } (i,j) \notin E \end{aligned}$$

Passo ricorsivo Tutti i sottoproblemi di dimensione (k) tale che $k > 0$

$$P_{ij}^k =?$$

Ricerchiamo la sottostruttura ottima.



Data una soluzione ottimale $P_{ij} = P_{ij}^n$ si possono verificare due casi:

1. Il vertice n NON è uno dei vertici intermedi

2. Il vertice n è uno dei vertici intermedi

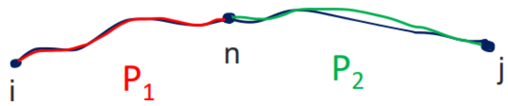
Caso 1 - Il vertice n NON è uno dei vertici intermedi

- P_{ij}^n coincide con P_{ij}^{n-1}

- Predecessore di j in P_{ij}^n coincide con predecesore di j in P_{ij}^{n-1}

Caso 2 - Il vertice n è uno dei vertici intermedi

$$P_{ij}^n = P_1 + P_2$$



$$P_1 = P_{in}^{n-1} \rightarrow P_{ij}^n = P_1 + P_2 = P_{in}^{n-1} + P_2$$

Mentre per quanto riguarda P_2 avrò che

$$P_2 = P_{nj}^{n-1}$$

Quindi sostituendo P_2 all'interno dell'equazione avrò che:

$$P_2 = P_{nj}^{n-1} \rightarrow P_{ij}^n = P_1 + P_2 = P_{in}^{n-1} + P_{nj}^{n-1}$$

Abbiamo quindi che il predecessore di j in P_{ij}^n coincide con il predecessore di j in P_{nj}^{n-1}.

Passo ricorsivo per P_{ij}^n

La soluzione ottimale $P_{ij}^n = P_{ij}$ è data da:

$$P_{ij}^n = min_p\{P_{ij}^{n-1}, P_{in}^{n-1} + P_{nj}^{n-1}\}$$

$i = n$ oppure $j = n \rightarrow P_{ij}^n = P_{ij}^{n-1}$.

Passo ricorsivo per P_{ij}^k

La soluzione ottimale $P_{ij}^k(k > 0)$ è data da:

$$P_{ij}^k = min_p P_{ij}^{k-1}, P^{k-1}ik + P_{kj}^{k-1}$$

$i = k$ oppure $j = k \rightarrow P_{ij}^k = P_{ij}^{k-1}$.

4.3.3 Equazioni di ricorrenza

Riassumendo abbiamo le sequenti equazioni di ricorrenza:

k=0 (CASI BASE)

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

$k > 0$ **(PASSO RICORSIVO)**

$$P_{ij}^k = min_p P_{ij}^{k-1}, P^{k-1}ik + P_{kj}^{k-1}$$

Definizione dei coefficienti

Coefficienti d_{ij}^k dei sottoproblemi.
$d_{ij}^k \rightarrow$ peso del cammino P_{ij}^k

$$k \in \{0, 1, \ldots, n\}$$
$$i \in \{1, \ldots, n\}$$
$$j \in \{1, ..., n\}$$

Quindi abbiamo **Numero di coefficienti** uguale a $n \times n \times (n + 1)$.
$k = n \rightarrow d_{ij}^n$ è il preso d_{ij} di P_{ij}.
Ricordiamo che la funzione obiettivo è trovare il peso del cammino, definiamo quindi i coefficienti nella seguente maniera:

k=0 (CASI BASE)

$$d_{ij}^0 = 0 \quad \text{se } i = j$$
$$d_{ij}^0 = w_{ij} \quad \text{se } i \neq j \text{ e } (i, j) \in E$$
$$d_{ij}^0 = \infty \quad \text{se } i \neq j \text{ e } (i, j) \notin E$$

$k > 0$ **(PASSO RICORSIVO)**

$$d_{ij}^k = min_p d_{ij}^{k-1}, d^{k-1}ik + d_{kj}^{k-1}$$

Predecessori π_{ij}^k

$\pi_{ij}^k \rightarrow$ predecessore del vertice j in P_{ij}^k

$$k \in \{0, 1, \ldots, n\}$$
$$i \in \{1, \ldots, n\}$$
$$j \in \{1, ..., n\}$$

Numero di predecessori:: $n \times n \times (n + 1)$.
$\pi_{ij}^n \rightarrow$ predecessore π_{ij} di j in P_{ij}.
Aggiungiamo alle equazioni di ricorrenza anche i predecessori.

k=0 (CASI BASE)

$$d^0_{ij} = 0 \quad \pi^0_{ij} = NIL \quad \text{se } i = j$$
$$d^0_{ij} = w_{ij} \quad \pi^0_{ij} = i \quad \text{se } i \neq j \text{ e } (i,j) \in E$$
$$d^0_{ij} = \infty \quad \pi^0_{ij} = NIL \quad \text{se } i \neq j \text{ e } (i,j) \notin E$$

$k > 0$ **(PASSO RICORSIVO)**

$$d^k_{ij} = min_p d^{k-1}_{ij}, d^{k-1} ik + d^{k-1}_{kj}$$
$$\pi^k_{ij} = \pi^{k-1}_{ij} \text{ se } d^k_{ij} = d^{k-1}_{ij} \text{ altrimenti } \pi^k_{ij} = \pi^{k-1}_{kj}$$

4.4 Algoritmo bottom-up

Per ogni valore di k da 0 a n, vengono calcolate due matrici $(n \times n)$:

$$D^k = [d^k_{ij}]$$
$$\Pi^k = [\pi^k_{ij}]$$

Il numero totale di matrici è $2(n+1)$.

Caso Base Ho che:

$$D^0 = [d^0_{ij}] = W \text{ matrice dei pesi input}$$
$$\Pi^0 = [\pi^0_{ij}]$$

Passo Ricorsivo In questo caso ho:

$D^k = [d^k_{ij}]$ e $\Pi^k = [\pi^k_{ij}]$
Sono calcolate usando le matrici $D^{k-1} = [d^{k-1}_{ij}]$ e $\Pi^{k-1} = [\pi^{k-1}_{ij}]$

Avrò quindi che le matrici $D^n = [d^n_{ij}]$ e $\Pi^n = [\pi^n_{ij}]$ sono le matrici di output!

4.4.1 Algoritmo bottom-up - Codice

```
Procedura calcola_valori_ottimi_FW(V,E,W)
  D^0 = W
  Π^0 = (n x n) matrix of NIL values
  for i = 1 to n do
    for j = 1 to n do
```

```
    if i != j and w_ij != ∞ then
        Π^0 [i,j] = i
for k = 1 to n do
  for i = 1 to n do
      for j = 1 to n do
          D^k[i,j] = D^{k-1}[i,j]
          Π^k[i,j] = Π^{k-1}[i,j]
          if i != k and j != k then
              if D^k[i,j] > D^{k-1}[i,k] + D^{k-1}[k,j] then
                  D^k[i,j] = D^{k-1}[i,k] + D^{k-1}[k,j]
                  Π^k[i,j] = Π^{k-1}[k,j]
```

Tempo $\Theta(n^3)$.

Rappresentazione esecuzione algoritmo

Link al video Youtube.