Algoritmica 14/15

EDIT DISTANCE

Il problema della *edit distance* (distanza di edizione, in una versione in italiano scarsamente usata) è alla base dei problemi di **confronto fra sequenze** perché il meccanismo algoritmico per affrontarlo è il medesimo con cui si risolvono diversi problemi di questo genere. Il paradigma di base è quello della programmazione dinamica introdotto nella dispensa 6. Definiamo dunque il problema e vediamo come affrontarlo. Lo schema algoritmico potrà poi essere facilmente modificato per risolvere altri problemi sulle sequenze.

Edit distance. Date due sequenze di caratteri X,Y trovare un *allineamento ottimo* delle due che corrisponda alla *minima distanza* tra X e Y, ove la distanza è definita secondo le regole seguenti:

• si ammette che nelle due sequenze possano essere inseriti spazi indicati con - e che, per ogni posizione nelle due sequenze, il carattere o lo spazio che appare nella X sia posto in corrispondenza (allineato) con il carattere o lo spazio nella stessa posizione della Y;

• la distanza è la somma delle distanze tra coppie di caratteri o spazi, uno in X e l'altro in Y, in posizioni corrispondenti nell'allineamento: caratteri uguali hanno distanza zero, indicata con Ø (**match**); caratteri diversi hanno distanza 1 (**mismatch**); un carattere allineato a uno spazio ha distanza 1 (**space**);

• la presenza di uno spazio si interpreta come l'avvenuta **cancellazione** di un carattere dalla sequenza, o come l'**inserzione** di un carattere nell'altra in posizione corrispondente: per questo motivo non si considerano spazi in posizioni corrispondenti di X,Y;

• la **edit distance** tra X e Y è la distanza relativa all'allineamento (o agli allineamenti) che minimizza tale distanza.

Valga come esempio la distanza tra le sequenze alfabetiche X = ALBERO e Y = LABBRO. Due allineamenti con distanza uguale a 3 (che come vedremo è la minima possibile, cioè la edit distance) sono riportati qui sotto. Il segno • indica le coppie nell'allineamento che danno luogo a distanza locale 1; per tutte le altre coppie si verifica un match e la distanza locale è Ø.

```
A L B E R O       A L - B E R O
L A B B R O       - L A B B R O
• •   •           •   •   •
```

Indicate le sequenze come $X = x_1\, x_2 \ldots x_n$, $Y = y_1\, y_2 \ldots y_m$, il calcolo impiega una matrice M di dimensioni $(n+1)\times(m+1)$ ove **M[i,j] indica la edit distance tra il prefisso $x_1\, x_2 \ldots x_i$ di X e il prefisso $y_1\, y_2 \ldots y_j$ di Y.** Dunque i caratteri di X corrispondono alle righe di M, i caratteri di Y alle colonne, e l'ultimo valore M[n,m] indica la edit distance tra le due sequenze. La riga e la colonna Ø corrispondono a prefissi vuoti, cioè X e Y non sono ancora stati esaminati, quindi la M si inizializza sulla riga Ø e la colonna Ø ponendo:

$M[Ø,j]=j$ per $Ø\le j\le m$, $M[i,Ø]=i$ per $Ø\le i\le n$,

valori che corrispondono ad affermare che il prefisso vuoto di X allineato con il prefisso $y_1\, y_2 \ldots y_j$ di Y corrisponde a una distanza j, e che il prefisso vuoto di Y allineato con il prefisso $x_1\, x_2 \ldots x_i$ di X corrisponde a una distanza i. Per le sequenze viste sopra il valore M[Ø,3] = 3 corrisponde all'allineamento del prefisso LAB di LABBRO con tre spazi --- inseriti prima di ALBERO.

Indicata con p(i,j) la distanza locale tra x_i e y_j, cioè posto **$p(i,j)=Ø$ se $x_i=y_j$ (match), $p(i,j)=1$ se $x_i\neq y_j$ (mismatch),** gli elementi M[i,j] con $1\le i\le n$ e $1\le j\le m$ si calcolano progressivamente, riga per riga, con la formula ricorsiva:

$$\mathbf{M[i,j] = \min\{M[i,j-1]+1,\ M[i-1,j]+1,\ M[i-1,j-1]+p(i,j)\}} \qquad (1)$$

che si spiega notando che l'allineamento ottimo tra i prefissi $x_1\, x_2 \ldots x_i$ e $y_1\, y_2 \ldots y_j$ si può costruire dagli allineamenti ottimi per i prefissi privati dell'ultimo carattere, confrontando $x_1\, x_2 \ldots x_i$ con $y_1\, y_2 \ldots y_{j-1}$-; $x_1\, x_2 \ldots x_{i-1}$- con $y_1\, y_2 \ldots y_j$; e $x_1\, x_2 \ldots x_i$ con $y_1\, y_2 \ldots y_j$; e scegliendo tra essi quello che genera distanza minima. Avremo per il nostro esempio la seguente matrice:

	Ø	L	A	B	B	R	O
Ø	Ø	1	2	3	4	5	6
A	1	1	1	2	3	4	5
L	2	1	2	2	3	4	5
B	3	2	2	2	2	3	4
E	4	3	3	3	3	3	4
R	5	4	4	4	4	3	4
O	6	5	5	5	5	4	3

L'ultimo valore M[6,6] = 3 indica la edit distance tra le due sequenze, che come già affermato è uguale a tre. Si noti che la

relazione (1) è ricorsiva ma l'algoritmo di costruzione di M è iterativo e segue uno schema di programmazione dinamica. Tale algoritmo ha **complessità quadratica** $\Theta(nm)$ perché richiede di costruire una matrice M di dimensioni $(n+1)\times(m+1)$, e il valore in ciascuna cella è calcolato in tempo costante, perché i tre valori che appaiono nella formula ricorsiva indicata sopra sono stati già calcolati in passi precedenti e memorizzati in M.

La formulazione dell'algoritmo in forma di un programma che chiameremo ED, fa uso di due vettori X[1:n], Y[1:m] che contengono le sequenze da confrontare (si noti che gl'indici partono da 1), e costruisce la matrice M secondo lo schema già descritto a parole:

```
ED(X,Y)
// Calcolo della edit distance tra due sequenze contenute nei
vettori X,Y di dimensioni n,m, costruendo la matrice M[0:n,0:m]

   for (i=0,i≤ n,i++) {M[i,0]=i;}  //inizializza la colonna 0
   for (j=0,j≤ m,j++) {M[0,j]=j;}  //inizializza la riga 0
   for (i=1,i≤ n,i++)
       { for (j=1,j≤ m,j++)
           { if (X[i]=Y[j]) {p=0;} else {p=1;}
             M[i,j]=min(M[i-1,j]+1,M[i-1,j-1]+p,M[i,j-1]+1); } }
   returnM[n,m];
```

Gli allineamenti che danno luogo alla edit distance M[n,m] (M[6,6] nell'esempio) si ricostruiscono risalendo all'indietro nella matrice, dalla casella [n,m] fino alla casella [Ø,Ø]. Da M[i,j] si risale a M[i-1,j-1] se questi due valori sono uguali e $x_i=y_j$ (per esempio si risale da M[6,6] a M[5,5] entrambi = 3, che indica un match tra caratteri); oppure si risale al valore di M uguale a M[i,j]-1 tra i tre adiacenti che lo precedono: in questo caso vi possono essere più alternative, corrispondenti ad allineamenti di uguale distanza. Nell'esempio si ricostruiscano i due allineamenti ottimi tra ALBERO e LABBRO già indicati in precedenza, partendo da M[6,6]: ci proponiamo di trovarne uno solo perché le soluzioni potrebbero essere moltissime.

Se si richiede di ricostruire un solo allineamento ottimo l'algoritmo richiede tempo $\Theta(n+m)$ per tracciare il percorso all'indietro dalla casella [n,m] alla [Ø,Ø]: infatti n+m è il numero massimo di passi che l'algoritmo può compiere sulla matrice (nel caso che questi avvengano sempre in orizzontale o in verticale) e ciascun passo richiede un numero costante di confronti.

Un programma per eseguire questo compito, che chiameremo ALLINEA, fa uso come il precedente ED dei due vettori X[1:n], Y[1:m] che contengono le sequenze da confrontare e della matrice M prodotta da ED, e costruisce l'allineamento ottimo in due nuovi

vettori ALX,ALY che conterranno le due sequenze inclusi gli spazi che devono apparire in tale allineamento. Per esempio, per il secondo allineamento ottimo visto sopra avremo:

```
ALX:    A L - B E R O
ALY:    - L A B B R O
```

I due vettori contengono lo stesso numero di elementi (sette, nell'esempio), ma questo numero non è noto a priori; quindi essi saranno definiti per k=n+m elementi, ALX[1:k], ALY[1:k], che sono certamente sufficienti. Si noti che i due vettori verranno riempiti a partire dall'ultima casella k (in corrispondenza all'esame di M[n,m]), per posizioni decrescenti e fino a una casella h, $1 \leq h \leq k$, che conterrà l'inizio delle sequenze allineate: gli elementi nelle caselle 1…h-1 non avranno alcun significato.

ALLINEA(X,Y,M)

// Costruzione dell'allineamento ottimo tra due sequenze contenute nei vettori X,Y di dimensioni n,m, utilizzando la matrice M costruita dal programma ED. Il valore di h, restituito dall'algoritmo attraverso la frase return, indica da che punto interpretare il contenuto dei vettori ALX, ALY //

```
k=n+m; h=k; i=n; j=m;
while ((i>0)||(j>0))
    {if (((i>0)&&(j>0))
          &&((M[i,j]==M[i-1,j-1])&&(X[i]==Y[j])
             OR ((M[i,j]==M[i-1,j-1]+1)&&(X[i]!=Y[j])))
                 {ALX[h]=X[i]; ALY[h]=Y[j]; i=i-1; j=j-1;}
     else {if j>0&&(M[i,j]==M[i,j-1]+1)
                {ALX[h]= -; ALY[h]=Y[j]; j=j-1;}
           else //deve essere M[i,j]==M[i-1,j]+1
                {ALX[h]=X[i]; ALY[h]= -; i=i-1;}
          }
      h=h-1;
     }
return h;
```

Esercizio 1. Si esamini attentamente il programma ALLINEA per comprendere come funziona e con quale criterio viene scelto l'allineamento tra i tanti possibili. Nell'esempio ALBERO-LABBRO quale allineamento sceglierà ALLINEA?

A titolo di ulteriore esempio sull'uso e sul significato della matrice M relativamente alla edit distance si consideri l'elemento M[3,6]=4 corrispondente al confronto tra ALB e LABBRO cui corrisponde l'allineamento ottimo:

```
- A L B - -
L A B B R O
•   •   •  •
```

Oppure l'elemento M[4,3]=3 corrispondente al confronto tra ALBE e LAB cui corrispondono gli allineamenti ottimi:

```
A L B E     A L B E     - A L B E
- L A B     L A B -     L A - B -
•   • •     • • •       •   •   •
```

Il calcolo di distanza mediante la matrice M può essere facilmente trasformato per risolvere problemi diversi dalla edit distance. Anzitutto si può considerare lo stesso problema ponendo che il costo di mismatch tra caratteri sia diverso da quello di cancellazione o inserzione, cioè il valore di p(i,j) per $x_i \neq y_j$ nella ricorrenza vista sopra non sia uguale a quello di allineamento tra un carattere e uno spazio.

Esercizio 2. Si calcoli nuovamente la distanza tra ALBERO e LABBRO ponendo p(i,j)=2 per $x_i \neq y_j$.

Un altro problema di grande importanza, sia per gli editori di testo che per la biologia molecolare, è quello di trovare tutte le **apparizioni approssimate** di una sequenza corta (la X, detta ora **pattern**) in una sequenza molto più lunga (la Y, detta ora **testo**), ammettendo che queste apparizioni possano contenere qualche differenza, ovvero che la distanza tra la X e la sottosequenza di Y con cui si confronta la X possa non essere zero. A tale scopo si può impiegare un algoritmo che ha struttura identica a quello per l'edit distance, notando però che **M[i,j] dovrà ora indicare la edit distance minima tra il prefisso $x_1\, x_2 \ldots x_i$ di X e tutte le sottosequeze di Y che terminano in posizione j.** Per ottenere questo risultato occorre unicamente inizializzare la riga zero con tutti Ø, cioè porre:

$M[Ø,j]=Ø, \quad \text{per } Ø \leq j \leq m$

che corrisponde ad assegnare edit distance zero all'allineamento tra il prefisso vuoto della X e qualunque prefisso della Y.

Chiariamo questi punti con un esempio. La seguente matrice è relativa alla ricerca delle apparizioni di X = RAT in Y = SERRATURA.

	Ø	S	E	R	R	A	T	U	R	A
Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
R	1	1	1	Ø	Ø	1	1	1	Ø	1
A	2	2	2	1	1	Ø	1	2	1	Ø
T	3	3	3	2	2	1	Ø	1	2	1

I valori nell'ultima riga indicano la minima edit distance tra la X e tutte le sottosequenze di Y che terminano quella posizione. Per esempio il valore M[3,6]=Ø corrisponde all'apparizione esatta di RAT nelle posizioni 4,5,6 di SERRATURA. Il valore M[3,4]=2 corrisponde al migliore allineamento tra RAT e tutte le sottosequenze di Y che terminano in posizione 4 (cioè tutti i suffissi di SERR), cioè ai tre allineamenti:

R	A	T		R	A	T		R	A	T
R	R	-		R	-	R		R	-	-
	•	•			•	•			•	•

Per determinare gli allineamenti tra la X e le sottosequenze di Y si procede come nel caso precedente, risalendo dalla posizione considerata nell'ultima riga fino alla prima riga, ma senza dover poi recedere fino alla posizione [Ø,Ø]. Per esempio l'ultimo elemento M[3,9]=1 corrisponde all'allineamento di RAT con RA- (il calcolo di M[3,9] è stato eseguito scendendo in verticale dalla casella M[2,9]). L'elemento M[3,4]=2 corrisponde ai tre allineamenti di RAT con RR-, R-R, R--.

Esercizio 3. Ponendo un limite massimo al numero di differenze ammesse tra il pattern e le sue apparizioni nel testo, cioè al valore della edit distance, indicare che vantaggi potrebbero essere ottenuti nel calcolo.