1 Tracce esercizi

Es.1 Es.5 p 316 Kleinberg-Tardos
As some of you know well, and others of you may be interested to learn, a number of languages (including Chinese and Japanese) are written without spaces between the words. Consequently, software that works with text written in these languages must address the *word segmentation problem*—inferring likely boundaries between consecutive words in the text. If English were written without spaces, the analogous problem would consist of taking a string like "meetateight" and deciding that the best segmentation is "meet at eight" (and not "me et at eight," or "meet ate ight," or any of a huge number of even less plausible alternatives). How could we automate this process?
A simple approach that is at least reasonably effective is to find a segmentation that simply maximizes the cumulative "quality" of its individual constituent words. Thus, suppose you are given a black box that, for any string of letters $x = x_1x_2...x_k$, will return a number $quality(x)$. This number can be either positive or negative; larger numbers correspond to more plausible English words. (So quality("me") would be positive, while quality("ght") would be negative.) Given a long string of letters $y = y_1y_2...y_n$, a segmentation of y is a partition of its letters into contiguous blocks of letters; each block corresponds to a word in the segmentation. The total quality of a segmentation is determined by adding up the qualities of each of its blocks. (So we'd get the right answer above provided that quality("meet") + quality("at") + quality("eight") was greater than the total quality of any other segmentation of the string.)
Give an efficient algorithm that takes a string y and computes a segmentation of maximum total quality. (You can treat a single call to the black box computing $quality(x)$ as a single computational step.)
(A final note, not necessary for solving the problem: To achieve better performance, word segmentation software in practice works with a more complex formulation of the problem—for example, incorporating the notion that solutions should not only be reasonable at the word level, but also form coherent phrases and sentences. If we consider the example "theyouthevent," there are at least three valid ways to segment this into common English words, but one constitutes a much more coherent phrase than the other two. If we think of this in the terminology of formal languages, this broader problem is like searching for a segmentation that also can be parsed well according to a grammar for the underlying language. But even with these additional criteria and constraints, dynamic programming approaches lie at the heart of a number of successful segmentation systems.)

Es.2 C'è una scacchiera $n \times n$, ad ogni casella ci sono dei valori reali, si parte da una qualsiasi casella nella riga più bassa e si vuole arrivare a una qualsiasi casella nella riga più alta, ci si può muovere o di una casella in alto o di una casella nella diagonale alto-destra o di una casella diagonale alto sinistra. Ogni volta che si visita una casella si guadagna il valore reale che ha quella casella. Bisogna progettare un algoritmo che trovi il percorso che

massimizzi il guadagno.
Nell'esempio sotto in grassetto e in sottolineato sono due migliori percorsi. (Partono entrambi dalla casella con il 4 che è in grassetto e sottolineato, comunque ricordo che nella riga più bassa si può scegliere qualsiasi casella si vuole per iniziare, non ce ne è una da cui partire per forza)

3	**4**	4	3
5	-1	-2	0
0	**0**	5	2
3	1	**4**	0

2 Soluzioni

Es.1 Data la stringa $y = y_1..y_n$, (ad esempio "*meetateight*") iniziamo a ragionare come se ci fosse solo il primo carattere in input $y = y_1$ (quindi la stringa in input è "m"), quale sarebbe in questo caso la partizione in blocchi ottima? Beh ce ne è una sola, ed è quella con l'unico blocco "m". Quindi il valore ottimo con un solo carattere, che chiamo $Opt[1]$, è uguale alla qualità di y_1, che chiamo $Q(y_1)$.
Ora ragioniamo come se la parola in input fosse solo $y = y_1y_2$ (ossia "me"), quale sarebbe la partizione in blocchi ottima in questo caso? Ce ne sono solo due: una è "y_1" , "y_2" e l'altra è formata dall'unico blocco "y_1y_2" (rispettivamente "m", "e" oppure "me".) L'ottima sarebbe quella che mi farebbe guadagnare di più come somma delle qualità dei singoli blocchi. Quindi $Opt[2] = max\{Q(y_1) + Q(y_2), Q(y_1y_2)\}$; dato che $Opt[1] = Q(y_1)$ posso scrivere: $Opt[2] = max\{Opt[1] + Q(y_2), Q(y_1y_2)\}$
Se invece la stringa in input fosse costituita dai soli tre primi caratteri $y_1y_2y_3$ (quindi "mee")? Avremmo quattro possibilità: una y_1, y_2, y_3; l'altra y_1y_2 e y_3; un'altra ancora y_1 e y_2y_3 e poi $y_1y_2y_3$. Quanto vale $Opt[3]$?
$Opt[3] = max\{Q(y_1) + Q(y_2) + Q(y_3), Q(y_1y_2) + Q(y_3), Q(y_1) + Q(y_2y_3), Q(y_1y_2y_3)\}$ (*)
Un'osservazione: togliendo ai primi due termini di (*) l'elemento $Q(3)$ si otterrebbe, rispettivamente, $Opt[1] + Q(y_2)$ e $Q(y_1y_2)$. Dato che $Opt[2]$ è uguale al più grande tra questi due posso sostituire i primi due elementi di (*) con l'unica espressione $Opt[2] + Q(3)$. Inoltre un'osservazione sul terzo elemento di (*): posso sostituire $Q(1)$ con $Opt[1]$. Diventa quindi: $Opt[3] = max\{Opt[2] + Q(y_3), Opt[1] + Q(y_2y_3), Q(y_1y_2y_3)\}$. Per comodità pensiamo ad $Opt[0] = 0$, diventa quindi: $Opt[3] = max\{Opt[2] + Q(y_3), Opt[1] + Q(y_2y_3), Opt[0] + Q(y_1y_2y_3)\}$.
Proviamo a generalizzare: quanto vale il valore ottimo se in input avessi solo i primi i caratteri (ossia $y_1y_2..y_i$)? $Opt[i] = max_{0\leq j\leq i}\{Opt[j-1] + Q(y_j..y_i)\}$.
Per risolvere il task possiamo quindi costruirci un matrice ausiliaria A di dimensione $2\times(n+1)$. Per non creare confusione con quanto detto prima mi riferisco agli elementi $(1,1)$ e $(2,1)$ della matrice come con, rispettivamente, $A_{1,0}$ e $A_{2,0}$, gli elementi $(1,2)$ e $(2,1)$ con $A_{1,1}$ e $A_{2,1}$, e

così via fino algi elementi $(1, n+1)$ e $(2, n+1)$ con $A_{1,n}$ e $A_{2,n}$. Come è fatta questa matrice? La costruiamo in modo tale che $A_{1,0} = Opt[0] = 0$, $A_{1,1} = Opt[1] = Q(y_1)$ e in $A_{1,i} = Opt[i] = max_{0 \leq j \leq i}\{Opt[j-1] + Q(y_j..y_i)\}$, il $j \leq i$ che è stato scelto in questa ultima equazione sarà salvato nella riga 2; così da poter ricostruire successivamente la partizione in blocchi.
Qui sotto un esempio di come è fatta la matrice A con l'istanza "*meetateight*" supponendo che Q("*me*") = 3, Q("*m*") = Q("*e*") = Q("*a*") = Q("*i*") = Q("*g*") = Q("*h*") = Q("*t*") = 0 , Q("*meet*") = 4, Q("*at*") = 4, Q("*eight*") = 4, qualsiasi altro blocco viene valutato negativamente.

0	0	3	3	4	4	8	8	8	8	8	12
0	0	0	2	0	4	4	6	7	8	9	6

Le varie partizioni in blocchi calcolare passo per passo sono quindi state:

- *m*
- *me*
- *me e*
- *meet*
- *meet a*
- *meet at*
- *meet at e*
- *meet at e i*
- *meet at e i g*
- *meet at e i g h*
- *meet at eight*

Partendo dall'elemento $A_{2,n}$ posso ricostruire la partizione in blocchi ottimale: vedo che è uguale a 6, allora vado a vedere l'elemento in $A_{2,6}$, che è 4, vedo di conseguenza $A_{2,4}$ che è 0 e ho finito. La partizione in blocchi ottimale sarà quindi $y_1y_2y_3y_4$, y_5y_6, $y_7y_8y_9y_{10}y_{11}$ che è proprio *meet at eight*.

Es.2 Chiamo S la matrice che rappresenta la scacchiera.
Costruiamoci una matrice Opt tale che nell'elemento $Opt_{i,j}$ c'è scritto quanto si guadagna con il cammino ottimo da una qualsiasi casella della prima riga alla casella i, j.
Nella prima riga gli elementi di Opt quanto varranno? Beh ad esempio in $Opt_{1,2}$ ho che l'unico modo per arrivare alla casella 1, 2 è proprio partire dalla casella 1, 2 (con le mosse lecite posso solo andare a caselle di righe superiori); quindi per la riga 1: $\forall j \in \{1, ..., n\} \quad Opt_{1,j} = S_{1,j}$.
Per una riga i qualsiasi quanto vale $Opt_{i,j}$? Beh se la posizione (i, j) non è all'estrema sinistra o all'estrema destra io posso raggiungere la casella $S_{i,j}$ o dalla casella sotto $(S_{i-1,j})$, o dalla casella sotto-diagonale sinistra $(S_{i-1,j-1})$ o dalla casella sotto-diagonale destra $(S_{i-1,j+1})$. Se è all'estrema sinistra non la posso raggiungere dalla casella sotto-diagonale sinistra, se all'estrema destra non la posso raggiungere dalla casella sotto-diagonale destra.
Basterà selezionare il più grande elemento di Opt che appartiene all'ultima riga per sapere a quanto ammonta il guadagno ottimo, infatti cerchiamo il cammino migliore da una qualsiasi casella della prima riga a una qualsiasi casella della riga destra.
Come ricostruire il cammino? Usiamo la stessa idea precedente solo che usiamo una matrice in cui un elemento è un array di due elementi, in un elemento ci salviamo $Opt_{i,j}$ e nell'altro elemento la casella da cui siamo partiti per arrivare alla posizione (i, j) (se siamo alla prima riga possiamo mettere uno 0).
Istanza:

3	4	4	3
5	-1	-2	0
0	0	5	2
3	1	4	0

Matrice Opt relativa solo ai guadagni:

12	13	13	12
9	8	7	9
3	4	9	6
3	1	4	0

Matrice Opt relativa solo all'informazione per ricostruirsi il percorso:

1	1	4	4
2	3	3	3
1	3	3	3
0	0	0	0

Pseudocodice:

for $j = 1$ *to* n **do**
  $Opt_{1,j}[0] = S_{1,j}$
  $Opt_{1,j}[1] = 0$
end
for $i = 2$ *to* n **do**
  for $j = 1$ *to* n **do**
    if $j \neq 1 \wedge j \neq n$ **then**
      $Opt_{i,j}[0] = max\{Opt_{i-1,j-1}[0] + S_{i,j}, Opt_{i-1,j}[0] + S_{i,j}, Opt_{i-1,j+1}[0] + S_{i,j}\}$
      $Opt_{i,j}[1]$ =la colonna del massimo di prima (se ad esempio è $Opt_{i-1,j-1}$ ci sarà j-1)
    end
    if $j = 1$ **then**
      $Opt_{i,j}[0] = max\{Opt_{i-1,j}[0] + S_{i,j}, Opt_{i-1,j+1}[0] + S_{i,j}\}$
      $Opt_{i,j}[1]$ = la colonna del massimo di prima (se ad esempio è $Opt_{i-1,j}$ ci sarà j)
    end
    if $j = n$ **then**
      $Opt_{i,j}[0] = max\{Opt_{i-1,j-1}[0] + S_{i,j}, Opt_{i-1,j}[0] + S_{i,j}\}$
      $Opt_{i,j}[1]$ =la colonna del massimo di prima (se ad esempio è $Opt_{i-1,j-1}$ ci sarà j-1)
    end
  end
end
$S \leftarrow \emptyset$
$k \leftarrow max\{Opt_{n,j}\}[1]$
$S \leftarrow S \cup k$
for *i = n-1 to 2* **do**
  $k \leftarrow Opt_{i,k}[1]$
  $S \leftarrow S \cup k$
end
return S