Problem set 3
Esercizio 4

Alessandro Straziota

Esercizio 4. Una sequenza ordinata di simboli di un alfabeto Σ si dice *palindroma* se resta invariata leggendola da destra verso sinistra. Più formalmente, $x_1x_2\dots x_n \in \Sigma^n$ è palindroma se $x_1x_2\dots x_n = x_n\dots x_2x_1$. Per esempio, `angelalavalalegna` è una sequenza palindroma. Una *sottosequenza* di una sequenza di simboli $x_1x_2\dots x_n$, è una sequenza $x_{i_1}x_{i_2}\dots x_{i_k}$ con $1 \leqslant i_1 < i_2 < \dots < i_k \leqslant n$. Per esempio, `cba` è una sottosequenza della sequenza `abcbca`.

Progettare un algoritmo che prende in input una sequenza di simboli e restituisce una sottosequenza palindroma di lunghezza massima. Per esempio, se la sequenza in input è `abcbca` l'algoritmo deve restituire `abcba`.

Progettazione algoritmo Risolvere il problema in questione, per una stringa x, può essere visto semplicemente come la ricerca di un *allineamento ottimale* tra x e la sua inversa, ovvero un allineamento tra la stringa $x_1x_2...x_n$ e $x_n...x_2x_1$.

Definiamo cose è un allineamento:
date due stringe $x = x_1x_2...x_n$, $y = y_1y_2...y_m \in \Sigma^*$ diremo che un *allineamento* tra x e y è un inseme $M \subset \mathbb{N}^2$ della forma

$$M = \{(i,j) \in \mathbb{N}^2 : i \leq n \wedge j \leq m \wedge \nexists(i',j')[i' \geq i \wedge j' \leq j]\}$$

In poche parole un qualsiasi elemento $(i,j) \in M$ sta a indicare il fatto che nel nostro allineamento di x e y avremo che i simboli x_i e y_j sono allineati, o analogamente che c'è un *match* tra x_i e y_j.
Quando invece per un certo carattere non è previsto un matching allora si dice che in quella data posizione c'è un *gap*.

Esempio: si vuole trovare un allineamento tra la stringa $x =$ `STOP` e $y =$ `TOPS`.
Un possibile matching può essere

S T O P -
- T O P S

con $M = \{(2,1),(3,2),(4,3)\}$. Notiamo per esempio che il primo elemento di M è $(2,1)$ e ciò vuol dire che stiamo allineando il secondo carattere di x con primo carattere di y. Ancora, si può notare che i caratteri x_1 e y_4 non hanno un matching, e che quindi avremo dei gap nelle rispettive posizioni, rappresentati col simbolo "$-$".

Ai fini invece di definire cosa è un *allineamento ottimale* è doveroso definire il **valore** di una soluzione ammissibile. Il valore di una soluzione si misura in base al numero gap presenti e al tipo di matching tra i vari caratteri.
Diciamo che un gap avrà un certo valore $\delta > 0$. Invece per i matching tra due simboli x_i e y_j possiamo distinguere due casi:

- quando $x_i = y_j$ allora il costo per il loro allineamento avrà valore $\alpha_{i,j} = 0$
- quando $x_i \neq y_j$ allora si dice che c'è un *missmatch*, e il suo costo sarà un certo valore $\alpha_{i,j} > 0$

A questo punto possiamo definire il valore di una soluzione ammissibile M come segue

$$COST(M) = \sum_{i \leq n : \forall j, (i,j) \notin M} \delta + \sum_{j \leq m : \forall i, (i,j) \notin M} \delta + \sum_{(i,j) \in M} \alpha_{i,j}$$

Perciò una soluzione ottima al problema è quella soluzione M che *minimizza* il suo valore $COST(M)$.

Per trovare quindi il valore di una soluzione ottima ricorriamo sempre metodo della definizione di una equazione ricorsiva. Se partiamo da gli ultimi simboli x_n, y_m possono accadere tre distinte situazioni:

- Se in un allineamento ottimo x_n e y_m sono allineati allora il valore dell'ottimo sarà $\alpha_{i,j}$ più il valore dell'ottimo sull'istanza $x_1, x_2, ..., x_{n-1}$, $y_1, y_2, ..., y_{m-1}$

- Se in un allineamento ottimo non c'è un matching per x_n allora il valore dell'ottimo sarà δ più il valore dell'ottimo sull'istanza $x_1, x_2, ..., x_{n-1}, y$
- Se in un allineamento ottimo non c'è un matching per y_m allora il valore dell'ottimo sarà δ più il valore dell'ottimo sull'istanza $x, y_1, y_2, ..., y_{m-1}$.

Più in generale, indicando con i, j l'istanza $x_1, x_2, ..., x_i, y_1, y_2, ..., y_j$, possiamo definire il valore dell'ottimo con l'equazione ricorsiva

$$OPT(i,j) = \begin{cases} j \times \delta & \text{se } i = 0 \\ i \times \delta & \text{se } j = 0 \\ \min \left\{ \begin{array}{l} \alpha_{i,j} + OPT(i-1, j-1), \\ \delta + OPT(i-1, j), \\ \delta + OPT(i, j-1) \end{array} \right\} & \text{altrimenti} \end{cases}$$

Siamo pronti ora per definire l'algoritmo per risolvere il problema esposto nell' **Esercizio 4**. Osserviamo che se una stringa x è palindroma allora se cerchiamo un allineamento ottimale con la sua inversa, questo avrà come valore ottimo 0. L'idea quindi è quella di cercare una allineamento ottimale tra x e la sua inversa, col vincolo però che non sono tollerati i missmatch (ovvero quando allineo due simboli diversi). Invece ogni carattere che non è allineato con nessun altro carattere dell'altra stringa (ovvero al quale è associato un gap) possiamo tranquillamente rimuoverlo dalla soluzione finale. Ricapitolando l'algoritmo esegue questa sequenza di passi:

1. Trovo un allineamento ottimale tra la stringa x e la sua inversa, tollerando solamente allineamenti tra simboli uguali, altrimenti poniamo dei gap. Per fare questo poniamo $\alpha_{i,j} = +\infty$ se $x_i \neq y_j$, altrimenti $\alpha_{i,j} = 0$ se $x_i = y_j$
2. Dalla stringa x rimuovo tutti e soli i caratteri a cui corrisponde un gap
3. La nuova stringa ottenuta sarà la sottostringa palindroma di x con lunghezza massima

Esempio: si consideri la stringa $x =$ `abaabca`.
Per prima cosa troviamo un allineamento ottimale tra x e la sua invera, secondo i vincoli imposti

```
a - b a a b c a
a c b a a b - a
```

Adesso notiamo che al carattere $x_6 =$ `c` è associato un gap, perciò rimuoviamolo. Adesso la nuova stringa $x' =$ `abaaba` è la sottostringa di x palindroma e di lunghezza massima.

Algorithm 1: OPT

Data: $x = x_1 x_2 ... x_n \in \Sigma^*$
Result: Il valore dell'ottimo $OPT(1, n) = \min COST(M)$

begin

$\overline{x} \leftarrow x_n ... x_2 x_1$;

$\alpha_{i,j} \leftarrow \begin{cases} 0 & \text{if } x_i = \overline{x}_i \\ +\infty & \text{otherwise} \end{cases}$;

$\delta \leftarrow 1$;

Sia C una matrice $(n+1) \times (n+1)$;

for $i = 0$ **to** n **do**

  $C_{i,0} \leftarrow i \times \delta$;

  $C_{0,i} \leftarrow i \times \delta$;

for $i = 1$ **to** n **do**

  for $j = 1$ **to** n **do**

$$C_{i,j} \leftarrow \min \left\{ \begin{array}{l} \alpha_{i,j} + OPT(i-1, j-1), \\ \delta + OPT(i-1, j), \\ \delta + OPT(i, j-1) \end{array} \right\};$$

return C;

Adesso la ricerca della soluzione ottima è il punto dolente dell'algoritmo. Infatti per ottenere l'allineamento M, da cui poi sarà facile ricavare la sottostringa palindroma di x, dobbiamo prima di tutto trasformare la matrice C ottenuta dalla procedura OPT in un $D.A.G$ pesato.

Consideriamo come esempio il caso in cui $x =$ `abbca`; la matrice C risultante da OPT sarà la seguente

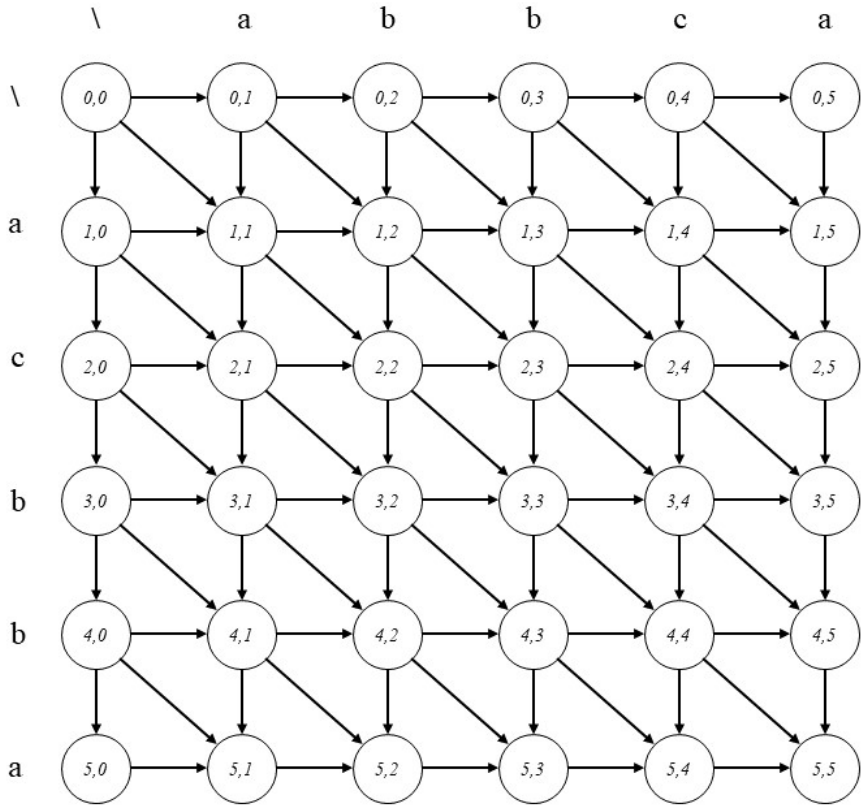
	\	a	b	b	c	a
\	0	1	2	3	4	5
a	1	0	1	2	3	4
c	2	1	2	3	2	3
b	3	2	1	2	3	4
b	4	3	2	1	2	3
a	5	4	3	2	3	2

Adesso per ogni casella della matrice creiamo in nodo del grafo e lo etichettiamo con le sue relative coordinate rispetto alla matrice.

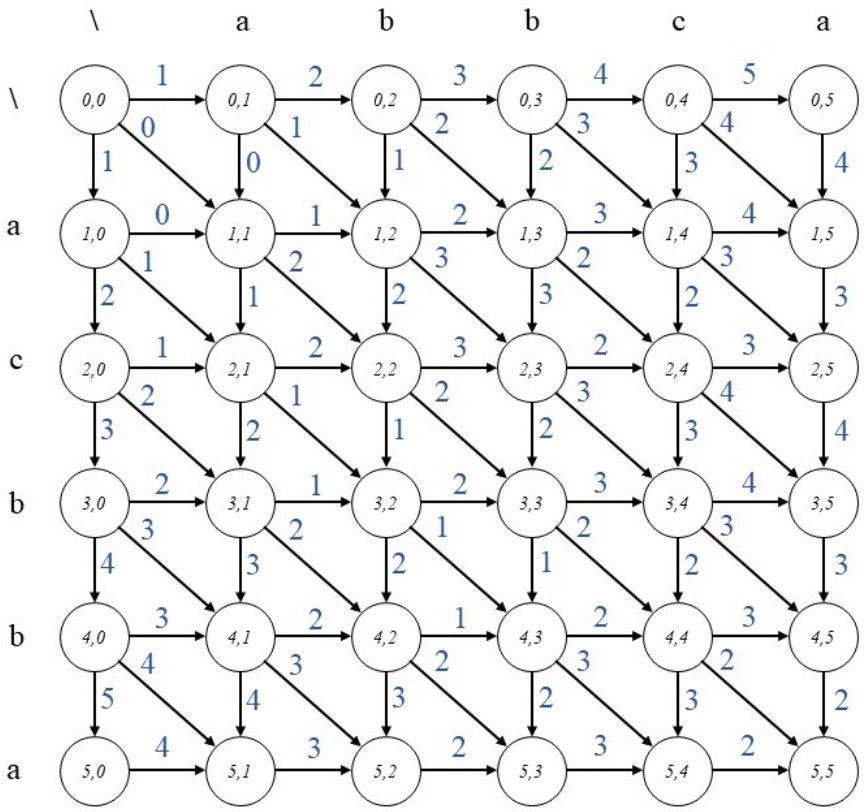
Adesso inseriamo gli archi come descritto:

- per $0 \leq i \leq n-1$ e per $0 \leq j \leq n-1$ inserisco gli archi tra i nodi con etichette:
  - da (i, j) a $(i+1, j)$ (archi verticali)
  - da (i, j) a $(i, j+1)$ (archi orizzontali)
  - da (i, j) a $(i+1, j+1)$ (archi obliqui)
- per $j = n$ e per $0 \leq i \leq n-1$ inserisco l'arco da (i, j) a $(i+1, j)$ (bordo destro del grafo)
- per $i = n$ e per $0 \leq j \leq n-1$inserisco l'arco da (i, j) a $(i, j+1)$ (bordo inferiore del grafo)

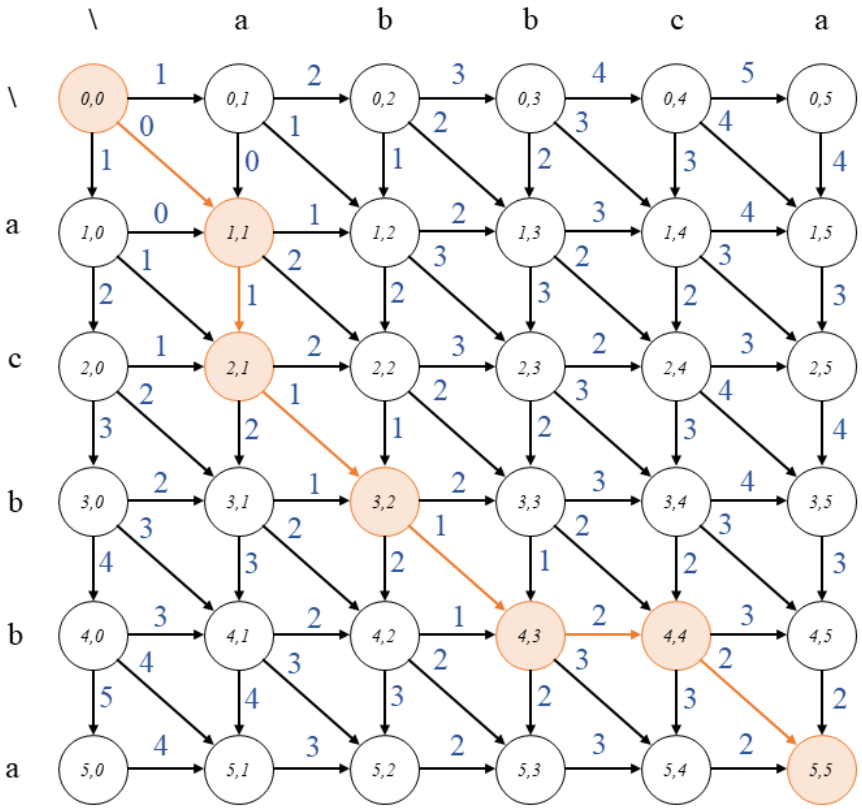
Il risultato sarà un grafo del tipo



In fine per definire i pesi degli archi basterà inserire per ogni arco entrante nel nodo etichettato con (i, j) il valore $C_{i,j}$.
Il risultato finale sarà il seguente



Adesso per costruire M bisogna in primis trovare un cammino minimo dall'unico nodo sorgente all'unico nodo pozzo.



A questo punto inseriamo dentro M soltanto le coppie (i, j) lungo il cammino minimo che sono raggiunte da un arco obliquo, ovvero quelle coppie (i, j) il cui predecessore lungo il cammino è il nodo etichettato con $(i-1, j-1)$. M risulterà in fine

$$M = \{(1,1), (3,2), (4,3), (5,5)\}$$

Il risultato finale è infatti quello atteso

```
x =  a c b b - a
y =  a - b b c a
```

Per concludere la sottostringa palindroma x' di x con lunghezza massima sarà composta da tutti e soli i caratteri di x che avranno un matching in M.

$$x' = \texttt{abba}$$

Analisi della complessità È facile notare che la costruzione della matrice C ha complessità $O(n^2)$, in quanto bisogna "riempire" $(n+1)\times(n+1)$ caselle, ognuna delle quali richiederà la ricerca del minimo tra esattamente tre valori (quindi con costo costante).
Si può dimostrare che anche la costruzione del grafo ha complessità $O(n^2)$. Infatti i nodi da inserire sono esattamente $(n+1)\times(n+1) \in O(n^2)$, mentre si può osservare che per ogni nodo bisogna inserire **al più** 3 archi uscenti, per un massimo di $O\big(3 \times \big((n+1)\times(n+1)\big)\big) \in O(3n^2) \in O(n^2)$.
Per quanto riguarda la ricerca del cammino minimo dal nodo sorgente al nodo pozzo, se si ricorre all'algoritmo di Dijkstra per la ricerca dei cammini minimi su grafo pesato con pesi non negativi, si può ottenere il risultato in tempo $O(|E| + |V|\log|V|)$, e dato che $|E|, |V| \in O(n^2)$ (dove ricordiamo n *è la lunghezza della stringa* x) la ricerca del cammino avrà complessità $O(n^2 + n^2 \log n^2) \in O(n^2 \log n)$. Ora poniamo un upperbound alla lunghe del cammino minimo, che è facile notare che $O(n)$. Da questo possiamo quindi affermare che costruire M (dato il cammino minimo dalla sorgente al pozzo) avrà complessità $O(n)$, e analogamente in tempo lineare possiamo ricavare la sottostringa soluzione del problema.

Ricapitolando, la complessità dell'algoritmo è composta da:

1. $O(n^2)$ per costruire C
2. $O(n^2)$ per ricavare il $D.A.G.$ pesato
3. $O(n^2 \log n)$ per trovare il cammino minimo dalla sorgente al pozzo, che avrà una lunghezza di $O(n)$
4. Dal cammino minimo ricavo M in tempo lineare, e in fine, sempre in tempo lineare ricavo la sottostringa soluzione del problema

Conclusione, l'algoritmo che data una stringa x restituisce la sottostringa x' palindroma e di lunghezza massimale ha complessit $O(n^2 \log n)$.