Capitolo 4

Esercizio 4

Il problema della **longest common substring** si pone l'obiettivo di estrarre, a partire da due stringhe di lunghezza arbitraria, anche non uguale, la sottostringa, comune ad entrambe, più lunga.
Dal punto di vista della griglia si assegnano i seguenti pesi (come da figura 4.1):

- $w(diagonale) = 1$
- $w(verticale) = -i$
- $w(orizzontale) = -j$

Dove gli archi diagonali rappresentano un match ed i e j sono gli indici che scorrono le due stringhe, ovvero rispettivamente sulle righe e sulle colonne.
Si assume che i nodi abbiano valore $x \geq 0$ quindi **qualora si ottenga un valore negativo come risultato dopo l'attraversamento dell'arco viene messo il valore 0**.
Quando si ha un match quindi il nodo finale ha valore pari a uno più il valore

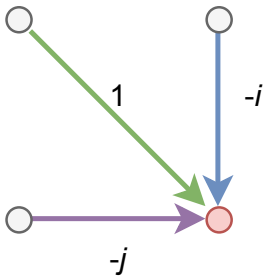


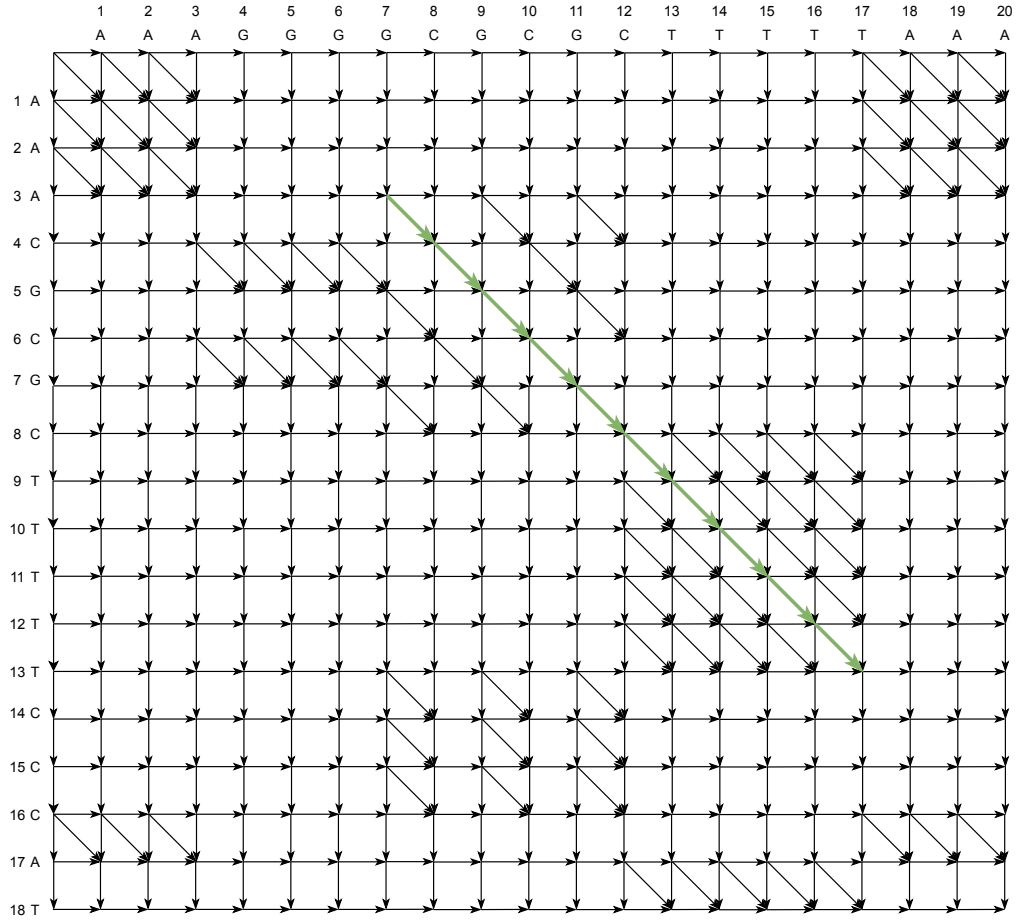
Figura 4.1: Rappresentazione grafica dei pesi degli archi. Si ricorda che nel nodo finale, in rosso, posso comunque avere solo valori ≥ 0

del nodo sorgente dell'arco diagonale.
Si ragiona quindi tenendo traccia del valore massimo raggiungo, tenendo traccia, per una maggior efficienza, anche delle coordinate. Una volta completata la griglia si avrà che il valore massimo corrisponde al nodo *sink* del cammino composto dalla più lunga sequenza possibile di archi diagonali consecutivi. Qualsiasi altra "operazione", con archi orizzontali o diagonali, comporta infatti una perdita di punteggio e ogni volta che un cammino diagonale termina viene azzerato il punteggio, tramite pesi negativi pesati sugli indici. Azzerando ogni volta si permette di avere la costruzione di un eventuale cammino diagonale che assegna man mano il valore corrispondente alla lunghezza del cammino stesso, avendo che la diagonale pesa 1. Il valore massimo quindi altro non è che la lunghezza della *longest common substring*. Sapendo che archi diagonali corrispondono a match tra le due sequenze si ottiene che tale cammino corrisponde alla *longest common substring*.
Volendo si può scegliere di salvare in un vettore i valori massimi qualora coincidano, per ottenere eventuali più *longest common substring* qualora ce ne siano.
In altri termini si cerca il sottocammino più pesante.
ipotizzando di non "azzerare" ogni volta in caso di mismatch otterrei comunque una griglia dover il nodo massimo mi permetterebbe di risalire la diagonale ritrovando la *longest common substring* ma tale nodo non potrebbe avere come valore la lunghezza della stessa, in quanto non ricomincerei il conto da 0 ogni volta che creo un nuovo cammino fatto di diagonali. Per ricostruire la *longest common substring* si parte dal nodo *sink*, come detto definito dal valore massimo calcolato. Si aggiunge il carattere corrispondente e si risale tutto il cammino diagonale, aggiungendo di volta in volta in testa il carattere letto. Ci si ferma quindi quando non si ha più un nodo il cui punteggio è stato ottenuto tramite un arco diagonale.

Vediamo quindi un esempio pratico. Siano date:

- AAACGCGCTTTTTCCCAT
- AAAGGGGCGCGCTTTTTAAA

Si costruisce quindi la griglia e si identifica il cammino diagonale più lungo:



Ricostruendo si ha che tale cammino identifica un massimo pari a 10. Il sottocammino più pesante è quindi quello rappresentato in verde. Ipotizzando quindi di poter 'risalire" la dialogale si ricostruisce:

CGCGCTTTTT

che è appunto la *longest common substring* delle due sequenze in input.