Bioinformatica

UniShare

Davide Cozzi
@dlcgold

Indice

Capitolo 1

Introduzione

Questi appunti sono presi a lezione. Per quanto sia stata fatta una revisione è altamente probabile (praticamente certo) che possano contenere errori, sia di stampa che di vero e proprio contenuto. Per eventuali proposte di correzione effettuare una pull request. Link: `https://github.com/dlcgold/Appunti`.

Capitolo 2

Introduzione alla bioinformatica

La genomica ha dimostrato negli ultimi anni ha dimostrato una capacità incredibile di produrre dati e questo ha portato alla nascita del bioinformatico, che diventa un esperto della gestione di questi dati sia dal punto di vista algoritmico che dal punto vista sistemistico.
A partire dal 2000/2001 ma sopratutto poco prima del 2010 si ha una crescita dei **dati genomici** non indifferente. I dati genomici sono quelli provenienti dal sequenziamento del DNA. Negli ultimi anni questa crescita ha superato la curva della **legge di Moore** quindi la crescita in termini di hardware (che si stima migliorare ogni 18 mesi) non riesce più a soddisfare la stima di richiesta di hardware necessario per il sequenziamento. Questa stima di sequenziamento è basata su Illumina, che produce le più diffuse macchine fisiche per il sequenziamento. Case farmaceutiche e laboratori che studiano il sequenziamento hanno almeno una macchina Illumina. La quantità di dati ha raggiunto i livelli dei petabyte e quindi ci si aspetta (e in parte già è così) che l'hardware non sia più in grado di elaborare tali dati.
La bioinformatica riceve quindi questa tipologia di dati. La bioinformatica è cruciale nell'ambito della ricerca in biologia molecolare (riguardante prettamente DNA), dove sempre più si ha necessità dell'appoggio dell'informatica, avendo a che fare con dati, nel dettaglio grandi dati.
Un altro aspetto è quello legato alle nanotecnologie e alla così detta **DNA-based computation**. Un esempio è legato al fatto che ormai si è in grado di manipolare il DNA al punto di essere in grado di assemblarlo in laboratorio, tramite un meccanismo a *tiling (tasselli)*, dove il tiling tendenzialmente è una figura regolare (triangolare, rettangolare, esagonale, etc...) con cui si compone del materiale biologico. Si riescono a fare letteralmente figure con il DNA (anche stelle, smile etc...) ma, sopratutto di questi tempi, vaccini,

che sono appunto manipolazione genetica di DNA o RNA. **Questa parte non è trattata nel corso**.

2.1 Breve introduzione biologica

Nel corso tratteremo prevalentemente sequenze di DNA. All'interno della cellula si hanno i **cromosomi** e un **genoma** altro non è che la collezione di cromosomi all'interno di un individuo. Il singolo cromosoma è rappresentato da filamenti di DNA "attorcigliati". Il cromosoma sostanzialmente è formato dalla coppia di due filamenti che si uniscono in una parte centrale detta **centromero**. I cromosomi, dal punto di vista informatico, sono vere e proprie sequenze (con i 4 nucleotidi, adenina, citosina, guanina e timina, ricordando la complementarietà delle basi A-T C-G), anche se si hanno varie regole per gestire questa "semplificazione". Un altro aspetto è il passaggio dal DNA alle **proteine**, anche se nel corso non verrà trattata la **proteomica**, ovvero lo studio delle proteine in se. In merito al passaggio da DNA a proteine si ha che il DNA contiene i **geni** da cui poi derivano le proteine. Un gene può portare a più di una proteina e questo si è scoperto grazie al sequenziamento. Allo stato attuale per "leggere" il DNA di un individuo dobbiamo passare per macchine di sequenziamento che però non possono leggerlo interamente ma, prendendo il DNA da una provetta (anche a partire da una singola cellula nel **sequenziamento single-cell**), si ha in output un file con dei frammenti del DNA originale, replicati in coppie, dette **read**. Tramite vari algoritmi siamo poi in grado di arrivare a capire e studiare il DNA per poi arrivare, si spera, ad uno dei principali fini della bioinformatica, quello di curare la vita, tramite terapie mediche (si parla di **medicina traslazionale**, ovvero non curo un paziente tramite protocolli generali ma sulla base del DNA del paziente, che viene studiato ai fini di stabilire la migliore terapia, che diventa personalizzata per l'individuo). Le scoperte biologiche più attuali sono ottenute praticamente sempre grazie all'intervento anche dell'informatica e della bioinformatica.
Un esempio di uso delle sequenze è confrontare regioni genomiche di varie specie per valutare eventuali somiglianze. Un primo modo è diretto, un secondo è confrontare dopo l'allineamento, con l'inserimento di gap (studieremo la cosa nel dettaglio).
Il bioinformatico fornisce al biologo/biotecnologo la strumentazione necessaria per fare le varie analisi.

2.2 Progetto Genoma Umano

Un elemento chiave nella bioinformatica è il **Human Genome Project (*progetto genoma umano*)**, progetto partito prima del 2000 (la prima base è del 1990) con vari obiettivi:

- identificare tutti i circa 30.000 geni nel DNA umano
- determinare le sequenze dei 3 miliardi di coppie di basi chimiche che compongono il DNA umano
- memorizzare queste informazioni in banche dati/db
- migliorare gli strumenti per l'analisi dei dati

La bioinformatica è andata avanti quasi sempre con progetti globali e il Progetto Genoma Umano è stato il primo di questi progetti, diciamo che lì nacque la bioinformatica. Si hanno vari *milesstones*:

- *1990:* progetto avviato come sforzo congiunto del U.S. Department of Energy e del National Institutes of Health (NIH)
- *Giugno 2000:* completamento di una bozza di lavoro dell'intero genoma umano
- *Febbraio 2001:* vengono pubblicate le analisi della bozza di lavoro
- *Aprile 2003:* Il sequenziamento del Progetto Genoma Umano è completato e il progetto è dichiarato finito due anni prima del previsto

Quest'anno, nel 2020, è stato lanciato un progetto ulteriore in quanto ora si è anche in grado di sequenziare il DNA nei pressi dei **telomeri**, ovvero le terminazioni dei cromosomi, che sono le regioni più difficili da ricostruire tramite il sequenziamento. Per farlo si hanno algoritmi e software davvero molto sofisticati.
Vediamo qualche numero:

- il genoma umano contiene 3 miliardi (3×10^9) di basi nucleotidiche chimiche che sono 4:
  - adenina (A)
  - citosina (C)
  - guanina (G)

  - timina (T)

- il gene mediamente è composto da 3000 basi, ma le dimensioni variano molto, con il più grande gene umano noto che è la Distrofina con 2.4 milioni di basi
- il numero totale di geni è stimato a circa 30000, molto inferiore alle stime precedenti da 80000 a 140000 (in quanto prima c'era il dogma che un gene codificasse una sola proteina, e si avevano circa 140000 proteine, che si conoscevano anche solo per le analisi del sangue)
- quasi tutte (99.9%) le basi nucleotidiche sono esattamente le stesse in tutte le persone. Basta lo 0.1% di differenze tra basi per "fare la differenza", anche differenziando predisposizioni geniche per una certa malattia
- le funzioni sono sconosciute per oltre il 50% del gene scoperto

Vediamo anche qualche numero (in stima) in merito agli organismi più studiati dai bioinformatici (spesso organismi con poche basi), più l'attualissimo *sars-cov-2*:

organismo	numero basi	numero di geni
uomo (Homo sapiens)	3 miliardi	30000
topo di laboratorio (M. musculus)	2.6 miliardi	30000
arabetta comune (A. thaliana)	100 milioni	25000
nematoda (C. elegans)	97 milioni	19000
mosca della frutta (D. melanogaster)	137 milioni	13000
lievito (S. cerevisiae)	12.1 milioni	6000
batterio (E.coli)	4.6 milioni	3200
Human immunodeficiency virus (HIV)	9700	9
sars-cov-2	$\sim$27 milioni	$\sim$15

2.3 Variazioni

Una volta conosciuta la sequenza dell'uomo si è cercato di studiare quello 0.1% di differenze tra vari esseri umani. Queste differenze sono dette **SNPs (*single nucleotide polymorphisms*)** (*detti a voce "snips"*) che rappresentano la variabilità nella popolazione umana. Sono le differenze a livello di singolo nucleotide. Subito dopo il Progetto Genoma Umano è partito, sempre

tramite il National Institutes of Health (NIH), un progetto che confrontasse popolazione africana, asiatica e statunitense per calcolare queste differenze, individuate tramite tool informatici, tramite il cosiddetto **assemblaggio di aplotipi**, che è prettamente un problema informatico, *NP-complete*, la cui soluzione più recente è data da un **algoritmo parametrico**. Dagli aplotipi vengono estratti gli SNPs e questo sarà visto tra qualche lezione. Gli SNPs sono serviti a determinare differenze tra le varie popolazioni campione in merito, ad esempio alla predisposizione alla Talassemia nelle popolazioni mediterranee. Questi studi servono appunto capire le predisposizioni delle varie popolazioni. Se una popolazione ha, nella maggior parte dei casi, una certa base in una certa posizione allora si ha uno SNPs. Il famoso 0.1% forma questi SNPs, il 99.9% della popolazione porta il cosiddetto **allele di maggioranza** mentre lo 0.1% l'**allele di minoranza**.
Uno studio ha dimostrato che, in Italia, solo i Sardi hanno un profilo genetico ben definito, tutti gli altri sono dei "mix genetici" e questo si è scoperto studiando gli SNPs.
Dal Progetto genoma Umano si è poi passati a confrontare il genoma di piccolissimi campioni, ad esempio 1000 individui, con il 1000 Genomes Project, un altro progetto con sforzi internazionali, fatto per mappare le variazioni su una popolazione di 1000 individui. Si segnala che per sequenziare un individuo ci sono voluti 10 anni nel primo caso ma poi ci è voluto molto meno. Ora un singolo individuo si sequenzia in qualche ora, a costi molto ridotti.
Dal DNA si sono anche ricavati i flussi migratori avvenuti nel corso della storia.

2.4 Pangenoma

Si vedrà, durante il corso, che dire **il genoma è una singola sequenza**, è ormai sostanzialmente errato. Avendo sequenziato milioni di individui si parla di **pangenoma** e le analisi devono ormai essere fatte non su un singolo genoma di riferimento ma si usa quello abbinato a tutta la serie di 0.1% di SNPs individuati finora. Nel dettaglio un pangenoma è una collezione di genomi multipli che sono correlati tra loro (variando solo in pochi punti). Si ha il pangenoma dell'uomo, di un batterio etc...
Dal punto di vista informatico diciamo comunque che il DNA è una sequenza sotto l'assunzione della **complementarietà delle basi**:

- adenina e timina sono complementari
- citosina e guanina sono complementari

e questo mi permette di poter studiare solo uno dei due filamenti del DNA.

Esempio 1. *Sia data la sequenza:*

$$S = acctacga$$

la complementare è:

$$S' = tggatgct$$

Se prendo la sequenza (o meglio una porzione di essa) di S_1 di un individuo h_1 e la sequenza S_2 di un individuo h_2 avrò un alta somiglianza con eventualmente uno o più SNPs.

La posizione dello SNP è detto **locus**. Uno SNP si ha quando nel 99.9% dei casi tutti gli individui hanno una certa base in una data posizione, avendo l'*allele di maggioranza*, mentre lo 0.1% degli individui ne ha una diversa, avendo l'*allele di minoranza* (e lo rilevo confrontando una popolazione).

Esempio 2. *Si hanno:*

$$S_1 = acc\mathbf{t}acga$$

$$S_2 = acc\mathbf{g}acga$$

ho uno SNP nel locus 4. Ipotizzando che il 99.9% degli individui siano come l'individuo con la sequenza s_1 ho che la base t è un allele di maggioranza mentre la base g è un allele di minoranza.

L'uomo si dice essere **biallelico** in quanto le "opzioni" per una certa posizione sono solo due. Alcuni cambiamenti possono anche essere del tipo *inserzione/delezione* (anche per sequenze di più basi contigue), parlando di **variazioni strutturali** (che sono comunque più complesse e meno tipiche). Per rappresentare il fatto che si hanno più sequenze con queste variazioni, soprattutto se sono inserimenti e delezioni, ma considerando che il 99.9% delle basi è uguale (cercando quindi una rappresentazioni che ottimizzi questa cosa), rappresentando quindi un pangenoma, dal punto di vista computazionale è un **grafo**. Ogni sequenza identica collassa in un solo nodo, avendo poi singoli nodi per le variazioni.

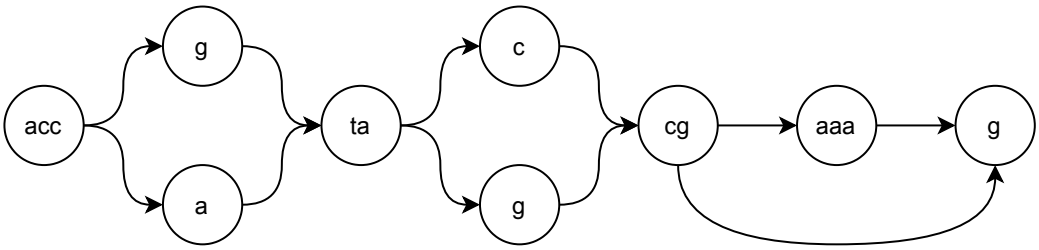
Esempio 3. *Ipotizzo di avere (con $-$ per indicare delezioni):*

$$S_1 = acc\mathbf{g}ta\mathbf{c}cg\mathbf{aaa}g$$

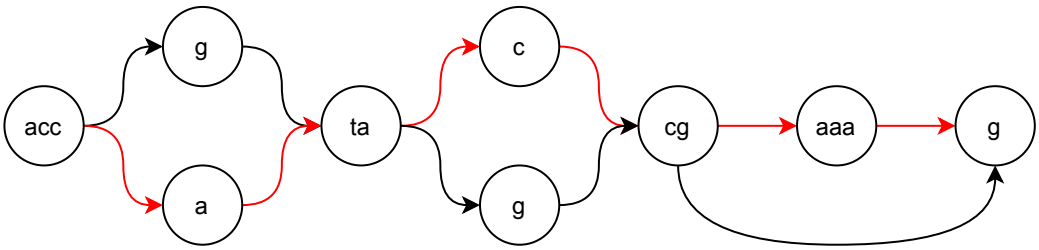
$$S_2 = acc\mathbf{a}ta\mathbf{g}cg\mathbf{aaa}g$$

$$S_3 = acc\mathbf{g}ta\mathbf{c}cg\texttt{---}g$$

E ottengo un grafo del tipo:



Studiando i cammini dei grafi ottengo tutte le rappresentazioni.
Questa rappresentazione però ha dei difetti, in quanto potrei avere cammini che non rappresentano nessuna sequenza di partenza. Pensando all'esempio sopra potrei avere il cammino in rosso che non rappresenta nessuna delle tre sequenze:



Rappresento quindi più di quello che voglio rappresentare.
Un pangenoma è un grafo che rappresenta una popolazione senza fare grandi distinzioni, avendo percorsi che non sono riscontrabili in nessun individuo della popolazione. Si ha comunque che il concetto di sequenza non è più adeguato. Il grafo di una popolazione è enorme e comunque, tramite colori, si possono distinguere i vari percorsi della popolazione (distinguendo facilmente "tracce comuni"). Parlando quindi di **genoma di riferimento** o si parla di quello specifico di un individuo o si parla del pangenoma di una popolazione, con le varianti.
Dal punto di vista di *file* le varianti vengono date in un file **Variant Call Format (*VCF*)**. L'input classico dei software è quindi spesso un VCF, così come l'output.

2.5 Progetti attuali

Vediamo ora quali sono i grandi progetti su larga scala attualmente in corso:

- **The Cancer Genome Atlas Pan-Cancer Analysis Project (*TCGA*)**, che cerca di costruire un catalogo delle caratteristiche

genomiche dei tumori, ovvero un catalogo delle mutazioni genomiche associate a tumori (ad esempio quello del seno si sa che è legato alla mutazione del gene BRCA che si sa bene dov'è)

- **The 1000 Genomes Project Consortium: A global reference for human genetic variation**, che cerca di ricostruire e raffinare un sequenziamento di diversi genomi per costruire un genoma di riferimento per una popolazione, nel dettaglio umana, (in formato VCF)
- **Trans-Omics for Precision Medicine**, il progetto per la medicina traslazionale
- **The Computational Pangenome Consortium**, che mira a studiare nuovi strumenti software che possano trattare il grafo del pangenoma visto che la maggioranza del software attuale ancora funziona su sequenze e non su grafi

2.6 Sequenziamento del DNA

Il sequenziamento (che letteralmente significa "produrre la sequenza") solitamente si svolge concatenando diverse operazioni:

1. estrazione del DNA
2. si ha una "libreria preparatoria" dove si mette del materiale genetico su un materiale preparatorio
3. si ha un meccanismo di "copie" tramite PCR o simili
4. si mettono i sample genomici in una macchina di sequenziamento che produce in output i dati

Un genoma non può essere letto "nucleotide per nucleotide" e i biologi, con la tecnologia attuale producono le cosiddette **read** del DNA originali. Si hanno due tipi di read:

- **read**, dette anche **short read**, lunghe circa 100 basi. Illumina produce tendenzialmente 100 o al più 150 basi
- **long read**, lunghe circa 10000 basi (se non di più, anche 20000)

Per ottenere il sequenziamento si ha un processo in cui:

- si divide il genoma in due parti, "aprendo" il filamento di DNA per permetterne la lettura
- si ha la **generazione delle read** da copie multiple del genoma tramite un processo biologico svolto dai macchinari, che sfruttano processi chimici
- si ha poi **l'assemblaggio dei frammenti**, ovvero un processo computazionale dove tramite algoritmi si assemblano le varie read per ottenere il genoma di partenza, avendo che le read hanno pezzi in *overlap*

Il problema del sequenziamento risale alla fine degli anni settanta con Sander e Gilbert che avevano studiato un processo di replicazione dando le basi allo studio del sequenziamento.
Dopo il sequenziamento dell'uomo si è passati a sequenziare molti altri organismi.
Oggi il sequenziamento è reso semplice dalla tecnologia. Un esempio è la tecnologia MinION, così piccola sta stare in una mano, che produce *long read* (anche se comunque con diversi errori). MinION è una tecnologia di *Oxford Nanopore*. MinION è USB ed è fatta per biologi che devono sequenziare in situazioni d'emergenza (esempio banale un biologo in Africa in piena emergenza Ebola). L'elaborazione dati viene fatta da un server.
Il primo sequenziamento è costato 3 miliardi di dollari per diversi anni, ora si fa in meno di 40 ore a 5000 dollari. Di recente si è passati addirittura a poche ore per un costo di circa 1000 dollari. Tornando alla *legge di Moore* si ha che il costo è collassato rispetto alla legge e quindi la capacità delle tecnologie di sequenziamento è molto maggiore della capacità di processare i dati, per quanto visto ad inizio capitolo. Si hanno quindi tanti dati ma non si è in grado di elaborarli.
Si tratterà anche il **confronto di genomi** per studiare poi gli aspetti evoluzionistici, tramite **alberi evolutivi**, anche **alberi evolutivi tumorali**. Il **confronto tra sequenze** permette di studiare le evoluzioni, anche quelle tumorali, dove si hanno mutazioni radicali di DNA. Approfondiremo anche tali mutazioni e il loro effetto (basta il cambio di una base per portare, ad esempio, all'anemia falciforme). Studieremo quindi anche come fare gli **allineamenti**. Approfondiremo il discorso della **filogenesi** e della **filogenesi tumorale**.
Tutto questo, in questo ultimo anno, è stato applicato allo studio di **sars-cov-2**, avendo lo studio delle variazioni.
Verrà approfondito anche il discorso del **riarrangiamento**.

Capitolo 3

Grafi di assemblaggio

La prima tematica che affrontiamo è l'assemblaggio delle read tramite grafi. Per questo problema abbiamo quindi:

- **input**: collezioni di read (short read e/o long read)
- **output**: grafo di assemblaggio da cui estrarre un cammino o un'unica sequenza

Si hanno principalmente due tipi di grafo:

- **grafo di De Bruijn (*DBG*)** (*si legge "grafo di de broin"*), che si prestano più per *short read* (da 100 o 150 basi)
- **grafo di overlap**, più comodo in caso di *long read*

Si useranno per questi scopi varie nozioni, tra cui:

- relazione di prefisso/suffisso tra k-mers
- relazione di prefisso/suffisso tra read
- Longest Common Prefix tra sequenze
- estrazione di cammino di Eulero dal grafo
- estrazione di cammino Hamiltoniano dal grafo
- Maximal Exact Matches (*SMEMs*)
- Burrows Wheeler Transform (*BWT*)
- indici succinti (come FM-Index)

- suffix tree e suffix array
- bloom filters, nati in ambito fisico e usati ora in ambito BigData
- min-hash e min-sketch, usati anche nelle reti neurali e nel Deep Learning quando si ha a che fare con grandi moli di dati

Studiare i grafi di assemblaggio può essere utile anche in ottica di applicare procedimenti simili ad altri problemi posti dai biologi.

3.1 Grafi in bioinformatica

In bioinformatica infatti uno strumento molto usato, anche oltre il sequenziamento, è quello dei **grafi**.
In letteratura la nozione di grafo compare nel 1735 con il **grafo di Eulero**, con Eulero che, si dice, fosse ossessionato dal problema dei **ponti di Königsberg**, volendo trovare il ciclo che attraversasse ogni ponte solo una volta. Ogni isola di Königsberg diventava un nodo e ogni ponte tra isole un arco tra nodi. Da qui la definizione del problema.

Definizione 1. *Il **problema del ciclo Euleriano** consiste nel trovare un ciclo in un grafo tale che visiti ogni arco una e una sola volta prima di tornare al punto di partenza. Si può passare dallo stesso nodo più volte.*
*Questo problema si dimostra risolvibile in **tempo lineare** sull'input $G = (V, E)$.*

Vediamo poi il "problema duale", quello in cui si vuole fare un cilo che non visiti due volte uno stesso nodo.

Definizione 2. *Il **problema del ciclo Hamiltoniano** consiste nel trovare un ciclo in un grafo tale che visiti ogni vertice una e una sola volta prima di tornare al punto di partenza.*
*Questo problema si dimostra essere **NP-complete**.*

La differenza di complessità di questi due problemi sarà qualcosa che bisognerà considerare parlando dello studio dei grafi in bioinformatica. Anche solo il problema dell'assemblaggio si vedrà è riducibile alla visita di un grafo (quindi non potremo formularlo come un problema di ciclo Hamiltoniano, la cui soluzione potrebbe richiedere anni).
La comparsa dei grafi nel mondo chimico è intorno a metà del 1800 con Cayley che li usò per rappresentare strutture chimiche, nel dettaglio usò **alberi** (che ricordiamo esserei grafi connessi aciclici) per contare gli isomeri strutturali.

In biologia l'uso dei grafi è stato introdotto a metà 1900 con l'esperimento di Benzer, che capì l'importanza dei grafi mentre cercava di distinguere quando determinati virus attaccano determinati batteri. Benzer è riuscito a mostrare che il DNA di questi virus era *lineare* mentre prima si congetturava che il DNA avesse delle biforcazioni. Per capire che non avesse delle biforcazioni ha sfruttato la capacità di alcuni geni dei virus di aggredire batteri, rappresentando la cosa coi **grafi ad intervallo**.

Definizione 3. *Nella teoria dei grafi, un* ***grafo d'intervallo*** *è il grafo d'intersezione di un multiinsieme di intervalli sulla linea reale. Ha un solo vertice per ciascun intervallo dell'insieme, e uno spigolo tra ogni coppia di vertici corrispondenti agli intervalli che intersecano.*
In poche parole associo una lettera ad ogni intervallo e collego nel grafo i vertici corrispondenti alla lettera qualora i due intervalli abbiano sovrapposizioni.

Il punto di svolta si ha però nel 1977 col sequenziamento e i due metodi di Sanger (che è tutti gli effetti il primo metodo di sequenziamento) e Gilbert, entrambi chimici. Entrambi i metodi generano frammenti etichettati di lunghezza variabile che vengono "letti" tramite elettroforesi.
Non approfondiamo nel dettaglio i metodi, essendo prettamente chimici e biologici.

3.1.1 Superstringhe e grafo di overlap

Siamo in ottica **short read sequencing** del **fragments assembly**.
L'assemblaggio dei frammenti del DNA è invece un problema prettamente computazionale, avendo l'assemblaggio dei singoli frammenti, ovvero delle **read** prodotte dal sequenziamento, anche in più copie, in un'unica sequenza genomica, detta **superstringa**. *Fino alla fine degli anni '90 l'assemblaggio di frammenti del genoma umano era visto come un problema intrattabile.*

Definizione 4. *Definiamo* ***stringa*** *come la concatenazione di simboli di un alfabeto Σ.*
In bioinformatica spesso si ha $\Sigma = \{a, c, g, t\}$

Definizione 5. *Definiamo il* ***shortest superstring problem (SSP)*** *come la ricerca, dato un insieme di stringhe, di trovare la più corta superstringa che le contiene tutte. So hanno quindi:*

- ***input****: una collezione $s_1, s_2, \ldots, s_n$ di stringhe che possono anche essere lunghe uguali o a lunghezza variabile*

- ***output**: una stringa s che contiene tutte le stringhe $s_1, s_2, \ldots, s_n$ dell'input come sottostringhe tale che $|s|$, ovvero la lunghezza della stringa s, sia **minima***

*Questo problema è **NP-complete** e assume che non ci siano errori di sequenziamento nella produzione delle stringhe $s_1, s_2, \ldots, s_n$.*
***La shortest superstring potrebbe non essere unica**.*

Esempio 4. *Vediamo un esempio di shortest superstring. Si assume per semplicità alfabeto binario $\Sigma = \{0, 1\}$.*
Si ha la collezione di stringhe binarie in input (che nel dettaglio sono tutte le possibili combinazioni di 3 simboli binari):

$$C_I = \{000, 001, 010, 011, 100, 101, 110, 111\}$$

Si può verificare che la shortest superstring è:

$$s = 0001110100$$

Con la shortest superstring ho letteralmente assemblato le stringhe in input.
Le read determinano la **coverage (*copertura*)** del DNA. Per valutare il coverage vado a vedere ogni base da quante read è coperta. Con Illumina ho un coverage di almeno 50x, quindi ogni posizione è coperta da almeno 50 read (lunghe ciascuna $\sim$150 basi). Per poter ricostruire la sequenza di DNA originale serve una certa quantità di coverage. Una coverage bassa potrebbe impedire la ricostruzione. Illumina va dal 50x minimo anche a 80x. MinION, della Oxford Nanopore, produce long read anche di 20000 basi ma con basso coverage, anche 3x, ma avendo read lunghe si riesce comunque ad assemblare. Quindi se ho long read mi basta un basso coverage mentre se ho short read mi serve un elevato coverage, avendo un insieme di read molto "fitto" e con poca "sparsità", in quanto si avrebbero gap, con zone non coperte. Il coverage è comunque dato "per media" e quindi poter comunque avere buchi.
Il punto chiave che mi permette di ricostruire il DNA è la sovrapposizione tra le varie read. Il DNA inoltre ha ripetizioni e questo costituisce, purtroppo, un limite all'assemblaggio e in merito studieremo il **fragment assembly problem**, che serve anche in altri contesti, oltre a quello dell'assemblaggio del DNA. Fin'ora abbiamo anche trascurato anche un altro problema, gli **errori di sequenziamento**, dati dal fatto che il processo di sequenziare non è *ottimo*, ovvero privo di errori, dove con errore si intende che nel DNA si ha una certa base e nella read prodotta dal sequenziamento se ne ha un'altra. In fase di assemblaggio questo tipo di errore comporta che non si riesce a

sovrapporre bene le read, non potendo vedere più alcuni **overlap** tra coppie read. Si ha quindi **perdita di informazione dell'overlap** e diventa più complicato assemblare il DNA, non impossibile ma più complicato.

Esempio 5. *Si hanno un pezzo di DNA e tre read che sono sovrapponibili:*

	1	2	3	4	5	6	7	8
$DNA =$	a	c	c	g	t	a	c	g
$R_1 =$	a	c	c	g	t			
$R_2 =$		c	c	g	t	a		
$R_3 =$				g	t	a	c	g

Possiamo quindi assemblare il pezzo di DNA.
Ma se ipotizziamo di avere un errore di sequenziamento con la terza base della seconda read:

	1	2	3	4	5	6	7	8
$DNA =$	a	c	c	g	t	a	c	g
$R_1 =$	a	c	c	g	t			
$R_2 =$		c	c	**c**	t	a		
$R_3 =$				g	t	a	c	g

Diventa più difficile assemblare.

Il tasso di errore nei macchinari Illumina è dello 0.01%, avendo circa due errori per read lunga 150. Per MinION si ha un tasso d'errore anche di circa il 10%, quindi ogni 50 basi ho una serie d'errore. Di recente, in ambito long read, si stanno progettando i **PacBio HiFi** (con HiFi che qui sta per "high quality framents") che producono long read con tasso d'errore allo 0.1%, facendo ben sperare per il futuro.
Tra i primi informatici che hanno fatto sequenziamento abbiamo Eugene Myers che era un esperto di algoritmi su stringhe e di pattern matching (parte attiva nella creazione dei suffix array), nonché responsabile della creazione dell'algoritmo di assemblaggio (famoso anche per BLAST). Eugene Myers era un esperto del problema della shortest superstring. A partire dalla tecnica di costruzione della shortest superstring ha sviluppato l'algoritmo di assemblaggio. Vediamo quindi, in primis, come costruire la shortest superstring. Per farlo bisogna in primis capire come confrontare le varie stringhe in input e come "foldarle". Per farlo faccio l'overlap che però a questo punto necessita di una definizione formale.

Definizione 6. *Definiamo **overlap** tra una coppia di stringhe s_i e s_j in input come il più lungo prefisso di s_j che ha un match perfetto (coincide) con un suffisso di s_i. Posso anche dire che è il più lungo suffisso di s_i che ha un match perfetto con un prefisso di s_j, ribaltare la definizione non cambia. L'overlap tra le due stringhe si indica con:*

$$ov(s_i, s_j)$$

Ricordiamo che una stringa la posso scrivere in modo scomposto in due modi:

- $s_i = s'_i x$, *con x suffisso*
- $s_j = x s'_j$, *con x prefisso*

*Tendenzialmente si prende l'**overlap più lungo**.*

Esempio 6. *Siano:*

$$s_i = accgtgtgt$$

$$s_j = gtgtgtccaa$$

Allora si ha che:

$$ov(s_i, s_j) = gtgtgt$$

con l'overlap lungo 6.

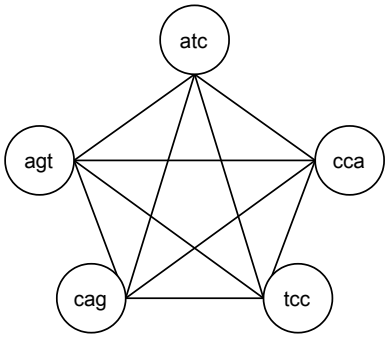
Proseguiamo quindi con il calcolo della shortest superstring dopo aver calcolato l'overlap di tutte le stringhe in input.
Creo un grafo con un nodo per ogni sequenza, etichettato con la sequenza stessa. Tracciamo quindi un arco tra due nodi sse i due nodi sono in overlap, associando all'arco la lunghezza dell'overlap.
Una tecnica per fare il grafo consiste in:

- collegare a priori di tutti i nodi ottenendo un grafo completo non orientato
- per ogni coppia di stringhe s_i e s_j metto l'arco pesato con l'overlap massimo $ov(s_i, s_j)$, dando anche direzione all'arco. Eventualmente posso anche dare doppio peso all'arco in base alla direzione. Si è ottenuto il **grafo di overlap**, che quindi è un grafo orientato (se in entrambi i versi non ho overlap lo lascio per praticità senza orientamento con peso 0)

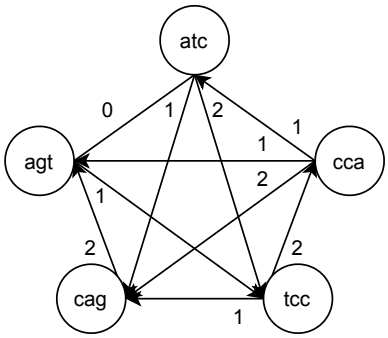
Esempio 7. *Sia la collezione di stringhe in input:*

$$C_i = \{atc, cca, cag, tcc, agt\}$$

e costruisco il grafo completo come detto sopra:



Aggiungo quindi i pesi relativi agli overlap dando l'eventuale orientamento e ottengo il grafo di overlap:



Per calcolare la shortest superstring dovremo calcolare un certo cammino sul grafo di overlap. Sicuramente un cammino che visita tutti i nodi mi porta ad avere una superstringa. Vediamo quindi una prima idea intuitiva:

Esempio 8. *Riprendendo il grafo di overlap dell'esempio precedente faccio:*

- *parto dal nodo atc e lo aggiungo alla superstringa, che per ora è* $s = atc$
- *seguo l'arco di peso 2 e arrivo in tcc*
- *aggiungo c (ovvero la parte non in overlap) alla superstringa, che per ora è* $s = atcc$
- *seguo l'arco di peso 2 e arrivo in cca*

- *aggiungo a (ovvero la parte non in overlap) alla superstringa, che per ora è $s = atcca$*
- *seguo l'arco di peso 2 e arrivo in cag*
- *aggiungo g (ovvero la parte non in overlap) alla superstringa, che per ora è $s = atccag$*
- *seguo l'arco di peso 2 e arrivo in agt*
- *aggiungo t (ovvero la parte non in overlap) alla superstringa, che per ora è $s = atccagt$*
- *mi fermo avendo visitato tutti i nodi*

Alla fine ho:

$$s = atccagt$$

che so essere una superstringa.

Si vede che il cammino, a conferma, tocca ogni vertice una e una sola volta, avendo un cammino Hamiltoniano ma, avendo i pesi, abbiamo a che fare con un **Traveling Salesman Problem (*TSP*)**. Dobbiamo però dimostrare che la superstringa ottenuta è anche la più breve.
Diamo però una piccola definizione formale del grafo di overlap.

Definizione 7. *Definiamo il **grafo di overlap** $G_{ov} = (V, E)$ tale che, data una collezione di stringhe $s_1, \ldots, s_n$:*

- *$V = \{s_1, \ldots, s_n\}$*
- *E è definito in modo che ogni arco $(s_i, s_j) \in E$ è un arco orientato da s_i a s_j di peso $|ov(s_i, s_j)|$ (quindi pesato con la lunghezza dell'overlap)*

Si dimostra poi che il **cammino Hamiltoniano di massimo costo** "produce" una shortest superstring. Facciamo una dimostrazione non formale. Innanzitutto per "produce" si intende che, dato il cammino prodotto da Hamilton di massimo costo, con i vertici etichettati dalle stringhe $s_{i,1}, s_{i,2}, \ldots, s_{i,n}$, la superstringa si ottiene sapendo che una stringa $s_{i,j+1}$ che è ha un prefisso in overlap con al precedente stringa $s_{i,j}$ la si può scrivere come:

$$s_{i,j+1} = ov(s_{i,j}, s_{i,j+1}) \cdot x_{i,j+1}$$

Possiamo anche dire che:

$$r(s_{i,j}, s_{i,j+1}) = x_{i,j+1}$$

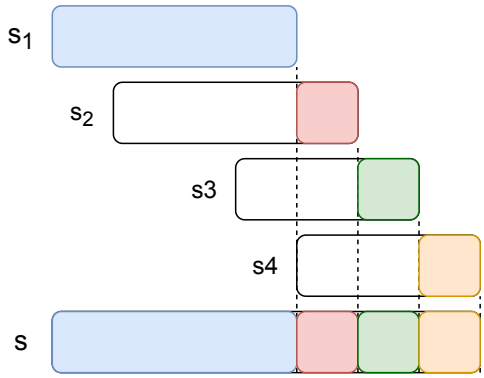


Figura 3.1: Esempio di formazione di una shortest superstring a partire da una collezione di stringhe sfruttando i "resti" degli overlap

indicando con $x_{i,j+1}$ la parte della stringa fuori dall'overlap, il "resto" possiamo dire. A questo punto so che la superstringa parte con $s_{i,1}$ e prosegue concatenando i vari $x_{i,j+1}$ (come si può vedere in figura 3.1):

$$s = s_{i,1} \cdot x_{i,2} \cdot x_{i,3} \cdot \ldots \cdot x_{i,n}$$

Bisogna dimostrare che il cammino Hamiltoniano di massimo peso coincide con la shortest superstring. Bisogna dimostrare che:

- un cammino Hamiltoniano di massimo peso calcolato come sopra è una superstringa, e questo di dimostra perché tocca ogni vertice e quindi ogni stringa che di conseguenza viene inclusa
- la superstringa appena calcolata è la più breve e per dimostrarlo si ha l'intuizione che se massimizzo l'overlap "globale" minimizzo la lunghezza della superstringa

Il calcolo della shortest superstring può quindi risolvere l'assemblaggio di stringhe anche se **si ricorda che il problema del cammino Hamiltoniano è NP-complete**.
La miriade di read però, quando ci lavorò per primo Eugene Myers, rendeva davvero difficile il calcolo (problema NP-complete e hardware storicamente poco potente). Servirono quindi anni per il primo calcolo, circa una quindicina, usando appunto il metodo della superstringa.
Si ha però un'euristica per calcolare la superstringa, usando un **algoritmo 2-approssimante** usando la **tecnica greedy**. In base a questa tecnica si sceglie sempre l'arco che pesa di più nel senso che ordino in ordine di peso tutti gli archi e faccio gli overlap tra le stringhe collegate. Dopo avere selezionato l'arco si prendono i due estremi e se ne fa la superstringa. Si continua

quindi cercando sempre gli archi che pesano di più creando poi la superstringa. **Non si vede nel dettaglio il funzionamento** e ovviamente non si ha la soluzione ottima e in realtà si congettura sia 2-approssimante ma non si ha una dimostrazione in merito, è un problema aperto da trent'anni e solo a livello sperimentale si è ipotizzata la 2-approssimazione.
Si ricorda che con il cammino Hamiltoniano ottimo si ottiene comunque una soluzione che potrebbe non essere unica.

3.1.2 Grafi di De Bruijn e k-mers

Siamo sempre in ottica **short read sequencing** del **fragments assembly**. Il metodo della superstringa è stato quindi usato per l'assemblaggio del primo sequenziamento (quello con il metodo Sanger) ma l'appoggio ad un problema NP-complete (il *ciclo Hamiltoniano*) rendeva il tutto troppo dispendioso. Il primo *assemblatore*, quello di Celera, usava però questo metodo più lento per il *fragment assembly*.
Vediamo ora una soluzione diversa, basata sui **grafi di De Bruijn** che invece come problema sottostante ha il *ciclo Euleriano* che sappiamo avere soluzione lineare.
Vediamo in primis qualche definizione.

Definizione 8. *Definiamo* ***k-mer*** *come è una sottostringa di lunghezza k. I k-mers sono quindi tutte le sottostringhe distinte di lunghezza k, non estraggo più volte lo stesso k-mer. Si segnala però che troppe ripetizioni dello stesso k-mer, che vengono trascurate, possono rendere difficile l'assemblaggio. Quindi il caso ideale è che tutti i k-mer estratti da una stringa siano distinti ma si seleziona il k in modo che ci siano al più due o tre ripetizioni dello stesso k-mer nella sequenza.*

Definizione 9. *Data una stringa s definiamo* ***spettro di s di dimensione/ampiezza*** *l è il* ***multiinsieme, avendo quindi ripetizioni****, di tutte le occorrenze di sottostringhe di lunghezza l (gli l-mers) e si indica con:*

$$spectrum(s, l)$$

Spesso lo spettro poi si rappresenta con i vari l-mer in ordine lessicografico*.*

Esempio 9. *Prendiamo una stringa s:*

$$s = tatggtac$$

Fissiamo $k = 3$ e si ha lo spettro di dimensione 3:

$$spectrum(s, 3) = \{tat, atg, tgg, ggt, gta, tac\}$$

(che in questo caso, non avendo ripetizioni, è un insieme di 3-mers.)

Dati i frammenti/read (che sono short read lunghe circa 150 basi) si estraggono da essi i **k-mers**, con k usualmente pari a 32, 31 o 28 per avere il minor numero di ripetizioni (i numeri sono stati identificati sperimentalmente). Si segnala che questa "proprietà" di avere sequenze circa lunghe 32 che non si ripetono è probabilmente legata anche al fatto che il DNA, preso come sequenza di simboli, è difficile da comprimere con i tool standard (zip, uso della BWT etc...), creando non pochi problemi alle banche dati anche se ancora non si è scoperto bene ne perché ne come risolvere la cosa.
Il problema di assemblaggio diventa quindi ricostruire una stringa da un insieme di k-mer:

- **input**: un insieme di stringhe $s_1, s_2, \ldots, s_n$
- estraggo i k-mer da tutte le s_i in input, per un k fissato
- assemblo i k-mer usando i grafi di De Bruijn

Ma prima di introdurre i grafi di De Bruijn vediamo un esempio di cosa significhi assemblare k-mers, partendo dal caso **senza ripetizioni**, avendo quindi un **insieme di k-mers** e non uno **spettro** (che è un multiinsieme).

Esempio 10. *Si prenda in input una collezione di k-mers con* $k = 3$ *(quindi 3-mers):*

$$C = \{atg, agg, tgc, tcc, gtc, ggt, gca, cag\}$$

Non essendo coincidenti se facessi gli overlap tra ogni coppia avrei al più overlap di lunghezza 2. Ipotizzando di fare il grafo di overlap avrei il seguente cammino Hamiltoniano (uno dei possibili):

$$atg \to tgc \to gca \to cag \to agg \to ggt \to gtc \to tcc$$

che produrrebbe la superstringa:

$$s = atgcaggtcc$$

Vedendo che anche con i k-mer posso ragionare in ottica di superstringa.

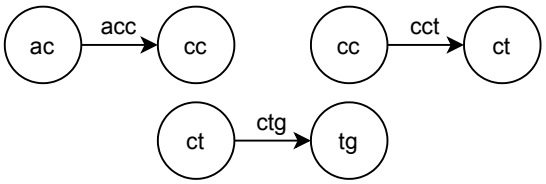
Ma con l'approccio che vogliamo ora non si passa per ogni vertice una e una sola volta ma per ogni arco una e una sola volta, usando il cammino Euleriano.
Dobbiamo fare si che quindi prendere ogni arco una e una sola volta corrisponde a prendere tutti i k-mers, avendo quindi che ogni arco deve essere

associato ad un k-mer. Costruiamo quindi un grafo che soddisfi queste condizioni (grafo che poi definiremo essere un **grafo di De Bruijn**). Si ha quindi un grafo dove gli archi sono i k-mer mentre i vertici all'estremo di un arco sono etichettati con il prefisso di lunghezza $k-1$ e il suffisso di lunghezza $K-1$ del k-mer (quindi i suffissi e i prefissi unici formano l'insieme dei vertici). L'arco è orientato dal prefisso al suffisso.
In altri termini i vertici agli estremi di un arco etichettato con il k-mer x altro non sono che i due unici (k-1)-mer estraibili da x.

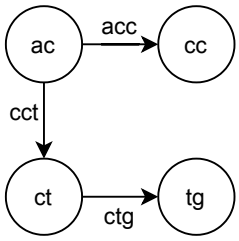
Esempio 11. *Prendendo una collezione di k-mer:*

$$C = \{acc, cct, cgt\}$$

so che, per il grafo $G = (V, E)$, ho $E = C$ mentre per V so che:



Quindi $V = \{ac, cc, ct, tg\}$.
Costruiamo quindi il grafo:



Quindi il cammino di Eulero (qui banale) e otteniamo la superstringa (e quindi l'assemblaggio) concatenando le etichette degli archi nell'ordine in cui vengono visitati, ragionando nello stesso modo in cui si faceva coi grafi di overlap, quindi partendo con la prima etichettata e proseguendo concatenando solo i resti dei vari overlap tra etichette degli archi:

$$s = acctg$$

Si ha quindi che:

Definizione 10. *Un grafo $G = (V, E)$* ***orientato*** *è un* ***grafo di De Bruijn*** *di ordine k se:*

- *i vertici sono un sottoinsieme di* Σ^{k-1}*, ovvero* $V \subseteq \Sigma^{k-1}$
- $\forall u, v \in V$ *si ha che* $(u, v) \in E$ *se esiste una parola* $w \in \Sigma^*$ *tale che* u *è (ha come etichetta) il prefisso di lunghezza* $k-1$ *di* w *e* v *è (ha come etichetta) il suffisso di lunghezza* $k-1$ *di* u

Quindi possiamo dire che, in modo astratto:

- *i vertici sono etichettati con un (k-1)-mer*
- *gli archi sono etichettati con un k-mer*

Tali grafi sono stati introdotti dal matematico De Bruijn nel 1946.
Questa è una definizione ***edge-centric*** *in quanto i* ***k-mer*** *vengono usati per gli archi. Si può avere anche una definizione* ***node-centric*** *dove estraggo i nodi per poi ottenere gli archi, collegando due vertici quando il prefisso nel primo nodo coincide con il suffisso del secondo ma avendo i nodi etichettati coi k-mer. Non necessariamente il k-mer dell'arco potrebbe non esistere. Normalmente per l'assemblaggio si usa la versione edge-centric. L'edge-centric implica il node-centric ma non viceversa. In entrambi i casi etichetto l'arco con il resto/estensione.* ***Capire meglio!***

Definizione 11. *Si ha che un vertice è* ***bilanciato*** *sse,* $\forall v \in V$*:*

$$in(v) = out(v)$$

con:

- $in(v)$ *numero di archi entranti in* v
- $out(v)$ *numero di archi uscenti in* v

Teorema 1 (Teorema di Eulero)**.** *Un grafo* ***connesso*** *è un* ***grafo Euleriano*** *sse suo ogni vertice è* ***bilanciato****.*

Dimostrazione. Vediamo le due direzioni della dimostrazione, avendo il *sse*.
Se si ha che per ogni arco entrante nel vertice v deve esistere almeno un arco uscente da v in quanto altrimenti mi "bloccherei" in un nodo, arrivando ad una contraddizione e non avendo quindi un cammino di Eulero. Quindi se non è bilanciato sicuramente non è Euleriano.
Se si ha un grafo bilanciato allora esiste (facciamo il caso "semplice") un ciclo Euleriano (e di conseguenza il caso "difficile" del cammino Euleriano). Infatti possiamo dare l'algoritmo che trova il ciclo Euleriano. Si ha quindi una **dimostrazione costruttiva**. Si ha quindi che:

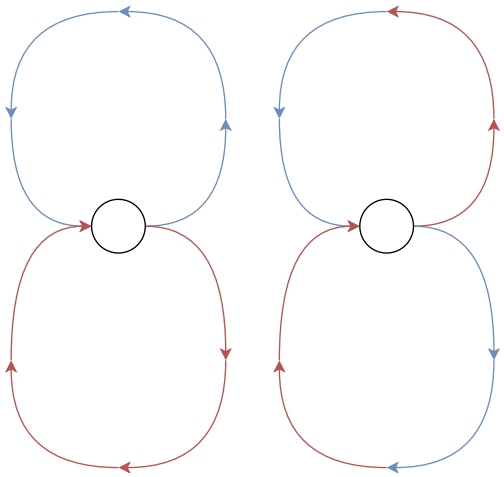


Figura 3.2: Idea del processo di apertura dei cicli dove si hanno due cicli, a sinistra in rosso e blu. Dopo l'apertura il cammino passa dal ciclo sopra a quello sotto, seguendo i colori nell'immagine a destra.

1. si parte da un vertice arbitrario v
2. si connette v ad un altro vertice usando archi, si riparte da tale arco e si fa la stessa operazione. Si ripete l'operazione fino a che non si è formato un ciclo
3. alla fine o uso tutti gli archi o altrimenti chiudo un ciclo lasciando vertici non usati. Nel secondo caso ripeto lo step 1) cercando di usare altri archi non usati. Facendo la **decomposizione del grafo in cicli**. In caso mi fermo quando ho usato tutti gli archi ma mi trovo con tanti cicli. Per derivare un unico ciclo da due cicli sfrutto il vertice di congiunzione in modo che se entro partendo da un ciclo in quel nodo esco con l'arco che mi porta nell'altro ciclo, facendo la cosiddetta **apertura dei cicli** (vedere figura 3.2). Riassumendo si combinano due cicli in uno unico e si itera questo passaggio fino alla creazione di un singolo ciclo. *Per fare questo uso l'ipotesi che G è bilanciato*

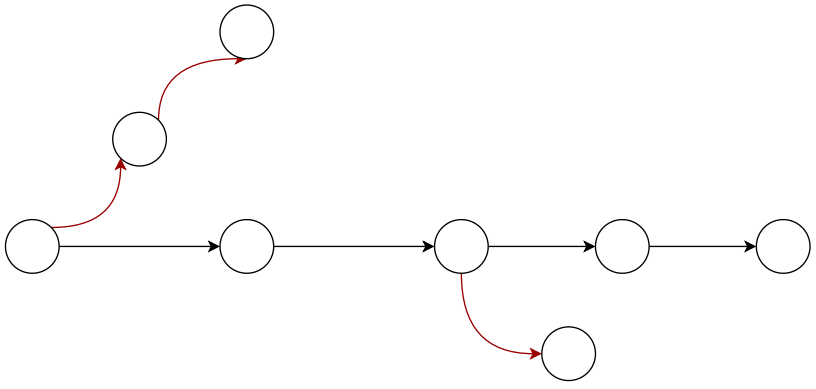
Questo algoritmo è lineare nella dimensione di un grafo, ovvero $O(|V| + |E|)$. □

Quindi per costruire il grafo di De Bruijn:

- **input:** una collezione F di frammenti
- si genera da F l'insieme dei k-mer (non lo spettro)

- per ogni k-mer dell'insieme dei k-mer estraggo il prefisso e il suffisso di lunghezza $k-1$
- si verifica l'esistenza del cammino di Eulero. Quindi:
  - se esiste si prosegue
  - se non è bilanciato si deve passare alla cosiddetta **pulitura del grafo**, eliminando gli archi detti **tip** (comportando la rimozione anche del nodo "destinatario" del grafo). Potrei avere perdita di k-mer. Le tip solamente sono singoli archi ma porrebbero anche essere cammini.
    Si usa anche il concetto di **bubble**, come effetto degli errori di sequenziamento. Una bubble è una situazione in cui si viola il teorema di Eulero ma che si elimina essendo un errore di sequenziamento, **risolvendo la bolla**. Per capire quale sia l'errore di sequenziamento sfrutto il fatto che ho più read che coprono la stessa porzione, scegliendo la base che è più frequente. **Il k condiziona le bolle oltre che alla connettività del grafo (*se è troppo grande rischio di non avere un grafo connesso*)**.
    Spesso si ha anche lo **scaffolding** per ottenere la connettività
- trovo il vertice "*head*" (l'unico che non ha vertici entranti) e il vertice "*tail*" (l'unico che non ha vertici uscenti), che saranno inizio e fine del cammino di Eulero
- si estrae il cammino di Eulero

Esempio 12. *Vediamo un esempio di tip, con le tip in rosso:*



quindi gli archi in rosso e i nodi a cui puntano possono essere rimossi. Vediamo un esempio di bubble. Ipotizzo di avere:

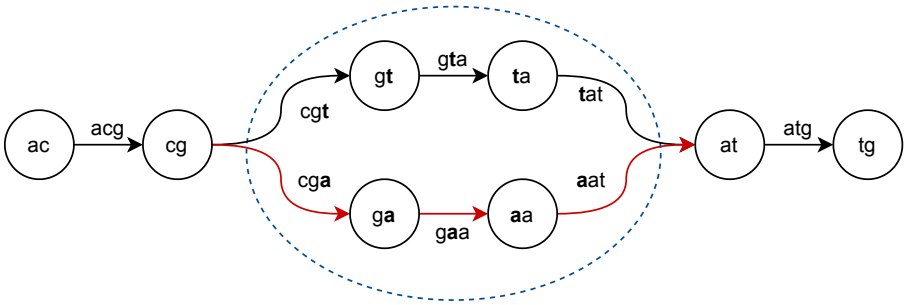
$$F = \{acgtatg, acgaatg\}$$

Notiamo quindi che:

$$acg\mathbf{t}atg$$

$$acg\mathbf{a}atg$$

Ma sappiamo che, tra le varie read, la t occorre molto più frequentemente della a in quella posizione. Si assuma di avere $k = 3$. Ho quindi, con la bolla (notando come la sua presenza impedisce di avere un cammino Euleriano) segnalata dall'ovale:



(dove si nota come la k influisca sulla natura della bolla).
Avendo però che la t occorre più frequentemente in quelle posizioni posso rimuovere il cammino con gli archi rossi e i due nodi centrali.

Esempio 13. *Sia:*

$$F = \{atgcc, caatg, gcgtt\}$$

e sia $k = 3$.
Avendo lo spettro:

$$\{atg, tgc, gcc, caa, aat, atg, gcg, cgt, gtt\}$$

ho l'insieme dei k-mer, ovvero l'insieme degli archi del grafo, è:

$$E = \{atg, tgc, gcc, caa, aat, gcg, cgt, gtt\}$$

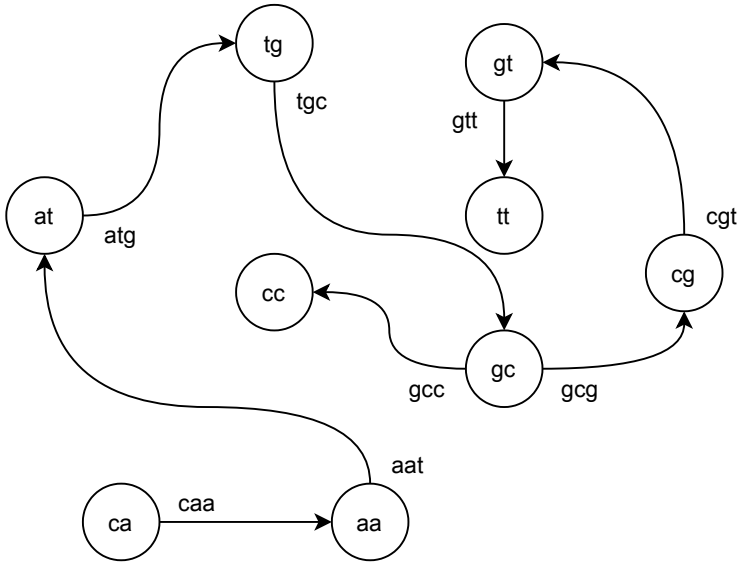
Genero quindi i prefissi e i suffissi di lunghezza $k - 1$ di ogni k-mer:

$$\{at, tg, tg, gc, gc, cc, ca, aa, aa, at, at, tg, gc, cg, cg, gt, gt, tt\}$$

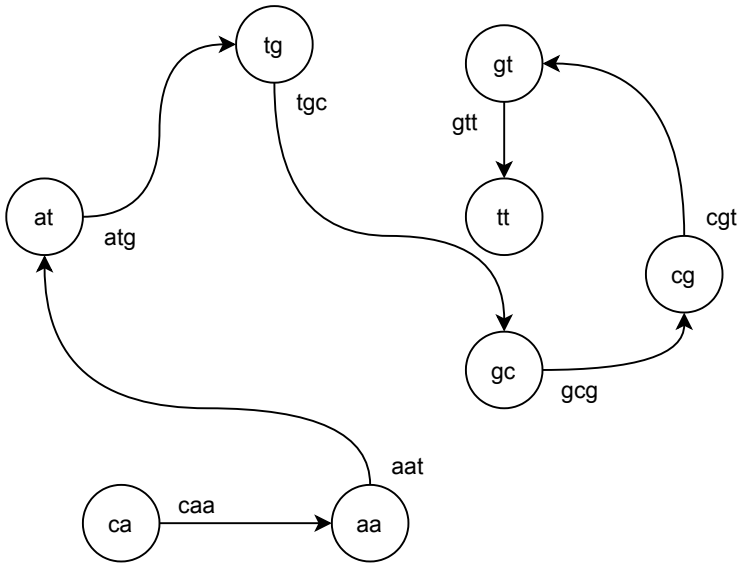
e quindi l'insieme dei vertici è:

$$V = \{at, tg, gc, cc, ca, aa, cg, gt, tt\}$$

Avendo quindi il grafo:



Le etichette degli sono per pura comprensione.
Bisogna quindi capire se esiste un cammino di Eulero. Purtroppo si hanno i vertici con tt e con cc che creano problemi. Inoltre il vertice gc non è bilanciato. Quindi il grafo non ammette cammini di Eulero.
Elimino quindi un arco "problematico", quello tra gc e cc, ottenendo:



Dove ho un cammino Euleriano, partendo dal vertice ca e arrivando al vertice tt. Si compone così la superstringa, ovvero l'assemblaggio:

$$s = caatgcgtt$$

Definizione 12. *Si indica con il termine **de novo** quando si fa una cosa "da zero".*
*Ad esempio si ha il **de novo genome assembly**.*

Definizione 13. *Si indica con il termine **reference bases** quando non si fa una cosa "da zero" ma si parte da una reference già preesistente.*

3.2 Note extra sui Grafi

I grafi di assemblaggio vengono usati anche nella **metagenomica**, ovvero lo studio di campioni di DNA di diversi organismi assieme. Questa cosa può essere utile quando il campione contiene vari organismi, si pensi ad esempio allo studio di un campione d'acqua o allo studio di un tampone faringeo.

Definizione 14. *Definiamo il paradigma **overlap-layout-consesus (OLC)**, proposto da Eugene Myers, come il paradigma per il quale si procede alla costruzione del grafo a partire da un insieme di read. Tramite le read si calcola poi il grafo di overlap per poi inferire il cammino, ovvero la **sequenza di consenso**. Il costo computazionale di calcolo non è indifferente.*
Rispetto ai grafi di De Bruijn è che possono immediatamente disambiguare brevi ripetizioni che i grafici di de Bruijn potrebbero risolvere solo nelle fasi successive.

Definizione 15. *Definiamo formalmente il **grafo di overlap**.*
Dato un insieme di read R un grafo di overlap è un grafo orientato:

$$G = (R, E)$$

dove i vertici sono le read stesse e si ha un arco tra r_i e r_j sse un suffisso di r_i è un prefisso di r_j, essendo in overlap. Il resto dell'overlap è detto e_{ij} ed etichetta l'arco (e si nota che il prefisso di r_i prima dell'overlap può essere usato come etichetta).

Si ha che:

- un cammino nel grafo di overlap rappresenta una stringa che è ottenuta assemblando le read. Tale stringa è ottenuta tramite $r_1 e_{i1} e_{i2} \cdots e_{ik}$

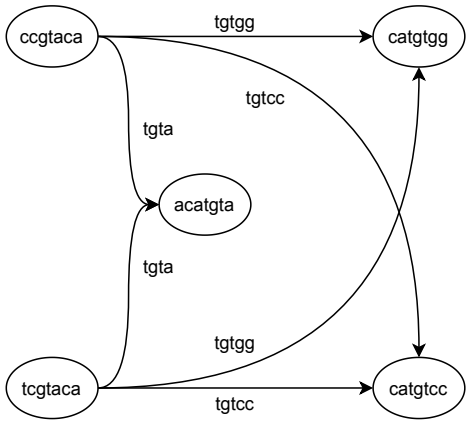


Figura 3.3: Esempio di grafo di overlap

- se un arco (r_i, r_j) è un cammino con solo due vertici è la stringa $s = r_i e_{ij}$
- un arco (r_i, r_j) è detto **riducibile** se esiste un percorso da r_i a r_j, con anche altri vertici, che rappresenta la stessa stringa di (r_i, r_j). Tali archi possono essere eliminati da un grafo di overlap ottenendo uno **string graph**, che offre una struttura più semplice e utile per la ricostruzione del genoma

Il grafo di overlap può essere usato per trovare le **varianti**.

Definizione 16. *Dato un insieme R di read si ha che un* ***edge-centric De Bruijn Graph****:*

$$G = (V, E)$$

di ordine k è un grafo dove i vertici sono i vari (k-1)-mer e si ha un arco tra u e v sse esiste un k-mer $w \in R$ tale che si il prefisso di lunghezza k-1 di w è uguale a u e il suffisso lungo k-1 di w è v. L'arco tra u e v è etichettato con l'ultimo carattere di v.

Se teniamo conto che i k-mer sono ripetuti possiamo usare un **multigrafo**.

Definizione 17. *Un* ***multigrafo*** *è un grafo in cui gli archi sono specificati da un* ***multinsieme****, il che significa che lo stesso arco può essere ripetuto più volte.*

Definizione 18. *Definiamo* ***isola*** *in un grafo di De Bruijn sono un insieme di nodi isolati. Sono generate da k-mer non frequenti e quindi probabilmente sono errori di sequenziamento.*

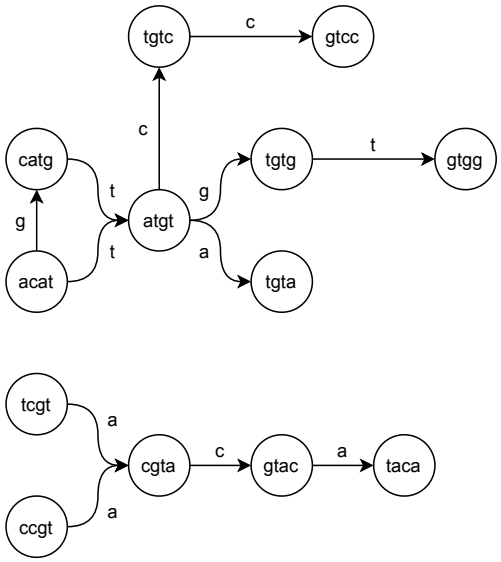


Figura 3.4: Esempio di grafo di De Bruijn con $k = 5$ edge-centric (anche se rappresentato non con l'intero 5-mer ma solo con il resto).

Definizione 19. *Definiamo **tip** in un grafo di De Bruijn come un nodo che si separa singolarmente con un arco dagli altri.*

I k-mer relativi a isole e tip vanno eliminati.
Se si ha che un k-mer è troppo ripetuto si è davanti probabilmente ad una **ripetizione**. L'uomo ha molti geni replicati, avendo magari varie versioni di un certo gene. Posso avere ripetizioni corte o molto lunghe.
Oltre alle bolle legate ad errori ho anche bolle legate magari al fatto che l'uomo è **diploide**, avendo **due copie** di ciascun cromosoma, una ereditata dalla madre e una dal padre, che possono avere comunque il 0.1% di differenza. Le read mi arrivano in eguale proporzione dai due cromosomi, sto assemblando quindi due fonti in una sola sequenza. Nelle banche solitamente si ha comunque solo l'**allele di maggioranza** (l'**allele**, dal punto di vista informatico, è la base variata a livello di cromosoma).
Vanno rimosse anche le **bubble** e i cammini vengono assemblate nelle stringhe dette **contigs**.
Un assemblatore che usa i grafi di De Bruijn è davvero complesso.
Nell'ultimo periodo in realtà si lavora con una reference e con un file VCF contenente le varianti. Le read da cui si parte solitamente sono a lunghezza fissata.
In fase di metagenomica si parte da read "colorate" provenienti da diversi organismi. Si costruisce un **grafo di De Bruijn misto** da cui si riesce ad

estrarre le sequenze delle singole specie. Per farlo si sfruttano varie informazioni, ad esempio la concertazione di copie di una certa read per le varie specie. Come informazione si usa anche il fatto che alcune specie hanno k-mer **unici**.
Diamo uno sguardo anche ai tempi computazionali, con n basi, g lunghezza del genoma e a lunghezza overlap (con ST indichiamo con *suffix array*, con DP *programmazione dinamica* e con NE *senza errori*):

	De brujin	***Overlap***
tempo	$O(n)$	$O(n+a)$ ST, $O(n^2)$ DP
spazio	$O(n)$, $O(\min\{N, g\})$ NE,	$O(n+a)$

Normalmente il sequenziamento viene fatto tramite **elettroforesi** ma picchi irregolari portano sicuramente ad errori di sequenziamento.
Tornando al discorso **ripetizioni** si ha che essere conferiscono "confusione" all'assemblaggio.
Risentire ultimi minuti lezione 16 marzo
Potrei avere, con Illumina, buchi di read non coperti che vengono colmati in modo algoritmico, tramite **contigs** che si collegano in **supercontigs**. Questa operazione è detta **scaffolding**.

3.2.1 Sequenziamento per Ibridazione

Sezione molto oscura.
Per leggere porzioni di DNA si usa il **sequenziamento per ibridazione *SBH***, ad esempio per leggere le porzioni atta a capire il test di paternità. Viene fatta tramite **affymetrix chip**, ovvero una piastra di materiale plastico dove viene depositato, dentro celle, il DNA da studiare. Il DNA sfrutta la complementarietà delle basi e queste celle contengono particolari k-mer così che le celle mi restituiscono i k-mer completi, ottenuti tramite il processo chimico dell'ibridazione che fa attaccare basi complementari.
Si ha quindi il **SBH problem**, che consiste di ricostruire una stringa dai suoi l-mer, con:

- **input**: un insieme S con tutti gli l-mer di una stringa s
- **output**: una stringa s tale che $Spectrum(s, l) = S$

Si usa il cammino Hamiltoniano per risolvere il problema oppure posso usare il cammino di Eulero usando i (k-1)-mer e il grafi di De Bruijn.

Capitolo 4

Allineamento di Sequenze

L'**allineamento** è la procedura principe usata per comparare sequenze biologiche.
Il confronto di sequenze, tramite sequenze di genomi, permette di creare **alberi di filogenesi**, alberi no-rooted con le specie attuali come foglie, che rappresentano l'evoluzione delle varie specie. La ricostruzione di tale albero avviene grazie al confronto, tramite allineamento, di sequenze.
Confrontare le sequenze consiste nello studiare il processo di evoluzione, risalendo al processo evolutivo tramite le varie mutazioni del DNA.
SI hanno due tipi di mutazioni:

- a livello nucleotidico, su singole basi, che sono **operazioni di edit**. Una singola base comporta la creazione di una diversa proteina e potrebbe portare a varie patologie. Le varie forme di una malattia genetica (tipo l'anemia) possono derivare da proteine diverse. A questo tema si affianca la medicina traslazionale

- a livello più ampio, cromosomico, che riguardano **riarrangiamenti**, ovvero scambi di pezzi di DNA o inversione di orientamento di alcune porzioni. Questo è legato magari a fenomeni ambientali e comporta il cambiamento di proteine e caratteristiche della specie, comportando un processo di adattamento e selezione naturale

Con l'allineamento si confrontano sequenze per vedere quanto sono simili/diversi. Si vuole o massimizzare la simiglianza o minimizzare le differenze. Per allineare si inseriscono i **gap**.

v	a	t		g	t	t	a	t	
w	a	t	c	g	t		a		c

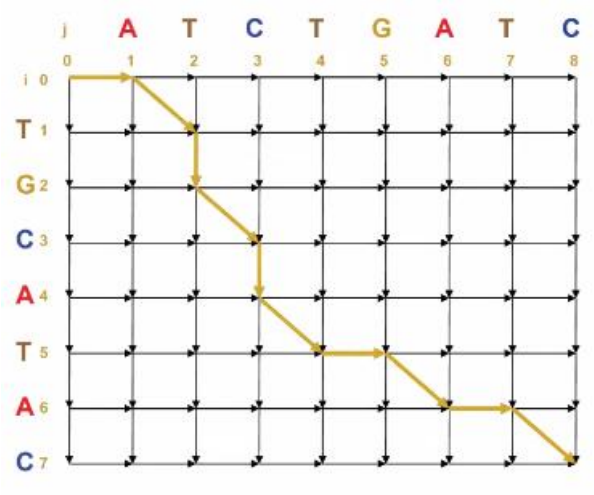


Figura 4.1: Esempio di matrice di programmazione dinamica per calcolare l'allineamento, dove si hanno 4 match, due inserzioni e due delezioni (si ragiona dal punto di vista di v).

Il biologo attribuisce alle mutazioni un punteggio, uno **score**. Tale score è stabilito attraverso uno studio probabilistico sulle mutazione avvenute nel corso dell'evoluzione.
Se allineo più di due sequenze parlo di **allineamento multiplo**. Si segnala che il 98% dei geni è pari tra due mammiferi e si ha più del 70% di similarità nelle sequenze proteiche.
L'allineamento dal punto di vista algoritmico è fatto tramite programmazione dinamica (mentre all'inizio erano fatte a mano dai biologi). Dal punto di vista bioinformatico l'allineamento è il processo base per praticamente qualsiasi software.

Definizione 20. *Definiamo l'allineamento di due sequenze A e B su Σ^* è una matrice $A^{2\times l}$ in cui la prima riga è associata ad A e la seconda a B. Ciascuna riga contiene i caratteri della sequenza associata intervallati da gap/spazi, rappresentati con il trattino.*
***Non è permesso avere due gap nelle due righe per la stessa posizione. In altri termini non posso avere una colonna di gap**.*
Ne segue che al massimo, avendo $|A| = n$ e $|B| = m$ allora:

$$l \leq n + m$$

e la matrice $2 \times n + m$, ovvero quella massima, consiste nell'avere la prima sequenza allineata con soli gap (che saranno sotto) e la seconda allineata con soli gap (che saranno sopra).

Per definire il problema dell'allineamento ottimo ho bisogno di una:

$$\delta : (\Sigma \cup \{-\})^2 \to \mathbb{R}^+$$

Potrei avere anche versioni della delta con pesi negativi:

$$\delta : (\Sigma \cup \{-\})^2 \to \mathbb{R}^-$$

dove δ rappresenta la **funzione di score**, tramite **matrice di punteggio**, che ha i caratteri dell'alfabeto che etichettano righe e colonne. Ogni a_{ij} è lo score. Sulla diagonale avrò soli 0.
Dati in input quindi A, B e δ si ha che il costo della matrice di allineamento M è il minimo (o a seconda massimo) della somma del costo delle colonne:

$$c(M) = \sum_{1 \leq i \leq l} \delta(M_{1i,2i})$$

Dove $A_{1i,2i}$ è una scrittura $M[1,i], M[2,i]$.
In altri termini è la somma delle delta delle colonne (delta prima colonne più delta seconda, più delta terza etc...).
La **distanza di edit** è un caso particolare dell'allineamento.
Il problema **LCS** (che quindi può essere vista come caso particolare dell'allineamento), dove date due sequenze A e B si cerca la lunghezza della più lunga sottostringa comune, è calcolabile tramite la **matrice di allineamento**, dove l'unico elemento della matrice che costa 1 è quando si ha un match ($\delta(\sigma, \sigma) = 1$), avendo alla fine che che $c(M) = |lcs(A, B)|$.
Se passo all'**allineamento multiplo di sequenza** ho in input ho $A_1, A_2, \ldots, A_k$ con k sequenze.

Definizione 21. *Definiamo l'allineamento multiplo di $A_1, A_2, \ldots, A_k$ è una matrice M di k righe e l colonne con:*

$$l \leq k \cdot \max\{|A_i|\}$$

tale che la riga i-esima, $\forall 1 \leq i \leq k$, di M contiene la sequenza A_i intervallata da gap. Anche qui non si possono avere colonne di soli gap. È quindi un'estensione/generalizzazione di quanto detto per due sequenze.

Definizione 22. *Definiamo anche in questo caso lo scoring, avendo che è identico al caso di due colonne. Infatti rappresenta il corso di una operazione tra **coppie** di simboli.*

Definizione 23. *Definiamo il **costo di una matrice di allineamento multiplo**. Tale costo dipende da cosa si vuole evidenziare e si hanno varie*

definizioni alternativa.
La definizione classica è quella della ***sum of pairs****, ma anche le alternative, tipo* ***consesus****, si basa sul dire che il costo è comunque in qualche modo la somma dei costi delle colonne della matrice. Bisogna però capire come stabilire tali costi. Per farlo penso ad ogni cella di una colonne come nodo di un grafo e costruisco un grafo completo. Il* ***sum of pairs*** *è il costo di quel grafo completo. Ogni arco tra coppie σ_1 e σ_2 (con σ_i contenuto della cella i-esima della colonna) è di peso pari al $\delta(\sigma_1, \sigma_2)$. Confronto quindi ogni cella della colonna con ogni altra, calcolando il delta, e faccio la sommatoria:*

$$c(M) = \sum_{1 \leq t,s \leq k} \delta(\sigma_t, \sigma_s)$$

Nel caso del ***consesus*** *dovrei vedere solo le differenze rispetto ad una sequenza reference, anch'essa in input. Confronto tutte le altre sequenze contro questa sequenza reference A^*. Avrei quindi come grafo un albero con la reference come radice. Alla fine sommo i costi degli archi come per il sum of pairs. Tale allineamento è anche detto* ***star alignment****.*
Questi due sono i due metodi principali ma ci possono anche essere altri "mapping" tra cui prendere alberi specifici basati sull'evoluzione delle specie. Questa tecnica è detta ***tree alignment****, confrontando solo coppie di un albero specifico di evoluzione.*
A seconda dello scoring poi si procede massimizzando o minimizzando.

Bisogna quindi capire come calcolare M tale che il costo di M sia ottimo. Per farlo si applica la programmazione dinamica. Per due sequenze lunghe rispettivamente n e m il costo sarebbe $O(n \cdot m)$, $O(n^2)$ se lunghe uguali.
Per k sequenze si ha sempre la programmazione dinamica ma il costo è più elevato. Se le sequenze sono in numero fissato si ha $O(l^k)$ ma se non conosco il numero di sequenze in input il problema diventa **NP-hard** anche quando prendo uno score delta banale che soddisfa la disuguaglianza triangolare su alfabeto binario. Si hanno quindi diverse euristiche per risolvere il problema e si basano sempre sull'idea del **consesus**, con tecniche greedy. Un esempio di software per farlo è **MAFFT**.

Esempio 14. *Date:*

- $A_1 = accgtc$
- $A_2 = ccgcc$
- $A_3 = accttc$

Ho una possibile (sarebbero diverse) matrice di allineamento, fatta per massimizzare i match:

A_1	a	c	c	g	t	c	-
A_2	-	c	c	g	-	c	c
A_3	a	c	c	t	t	c	-

Per esempio per la prima colonne avrei:

- $\delta(a, -)$
- $\delta(a, a)$
- $\delta(-, a)$

coi tre valori che andrebbero sommati nel caso di sum of pairs.

Fondamentalmente allineare sequenze significa trovare un cammino in una griglia, un problema del tipo **Manhattan Tourist Problem (MTP)**. La griglia rappresenta diversi modi per attraversare la città da un punto A ad uno B dove si hanno pesi per ogni arco (i lati dei quadrati della gruglia hanno un peso associato) per specificare il numero di attrazioni. Lo scopo è arrivare alla fine massimizzando il numero di attrazioni viste.
Si cerca quindi il cammino più pesante in una griglia pesata. In input si ha la griglia, il punto di partenza e di arrivo. Una strategia greedy per la quale ad ogni istante scelgo l'arco più pesante non sempre porta alla soluzione ottima. Si ha quindi un algoritmico di programmazione dinamica con il criterio di memorizzazione.
L'algoritmo ricorsivo si preoccupa di capire come arrivare al nodo finale ragionando a partire dal nodo finale stesso, sapendo che posso fare solo certi movimenti sulla griglia. Derivo il costo ottimo di un nodo a partire dai costi ottimi dei nodi che portano a quel nodo (scegliendo il migliore). Con la pro-

Algorithm 1 Algoritmo MTP con ricorsione

if $n = 0 \vee m = 0$ **then**
  return $MT(n, m)$
$x \leftarrow MT(n-1, m) + w(u, v), \;\; u = (n-1, m), v = (n, m)$
$y \leftarrow MT(n, m-1) + w(z, v), \;\; u = (n, m-1), v = (n, m)$
return $\max\{x, y\}$

grammazione dinamica parto dalla sorgente e accumulo il punteggio per ogni percorso possibile, scegliendo poi il miglior percorso. Visito l'intera griglia e

calcolo ogni volta il modo migliore per arrivare ad un nodo, risolvendo ogni volta problemi di massimo. Alla fine vado a ritroso per capire che percorso fare mentre nell'ultimo nodo potrò già vedere il peso ottimo del percorso scelto.

$$s_{i,j} = \max \begin{cases} s_{i-1,j} + w(u,v), & u = (i-1,j), v = (i,j) \\ s_{i,j-1} + w(z,v), & z = (i,j-1), v = (i,j) \end{cases}$$

Si a quindi, per una griglia $n \times m$, un costo pari a $O(n \cdot m)$.

Potrei aggiungere al problema l'uso delle diagonali, dovendo cambiare il sistema, aggiungendo il costo della diagonale:

$$s_{i,j} = \max \begin{cases} s_{i-1,j} + w(u,v), & u = (i-1,j), v = (i,j) \\ s_{i,j-1} + w(z,v), & z = (i,j-1), v = (i,j) \\ s_{i-1,j-1} + w(o,v), & o = (i-1,j-1), v = (i,j) \end{cases}$$

Trovare un cammino ottimo (uno di quelli più pesanti) della griglia corrisponde all'allineamento tra sequenze.
Dato il costo quadratico di confrontare due sequenze si passa a considerare anche metodi di confronto **alignment-free**, dove si decide che due sequenze sono molto simili senza fare l'allineamento.
Per allineare nel modo più veloce possibile si è passati poi dalla programmazione dinamica i metodi **bwt-based**.
L'allineamento può essere usato sui k-mer per eliminare a priori errori prima della costruzione del grafo di De Bruijn.

Definizione 24. *Definiamo il seguente* ***paradigma****. La somiglianza/similarità a livello di sequenza implica "stessa funzione".*
Due regioni genomiche, che si sa essere geni, uguali in due specie si ha che rappresentano lo stesso gene (e questo permette di studiare i topi in ambito medico e farmacologico).
Si rileva che la stessa funzione non implica per forza stessa sequenza e quindi stesso gene. Non vale quindi il viceversa.

Ricordiamo che il problema MTP consisteva nel trovare il cammino che massimizzava il cammino in una grigia. La risoluzione usa appunto la programmazione dinamica.
Interpretiamo il riempimento della griglia in termini di confronto di sequenze.
Un allineamento senza mismatch altro non è che il calcolo della Longest Common Subsequence LCS.

Per trovare l'LCS posso usare un grafo pesato che parte da un *source* e arriva ad un *sink*, trovando all'interno il cammino più pesante dal source al sink. Ogni cammino è tra il source e il sink. I nodi rappresentano i confronti tra due simboli delle due stringhe di partenza. Se ho un arco diagonale mi sto spostando su entrambe le sequenze, che in quel nodo hanno lo stesso simbolo. Con gli archi verticali e orizzontali mi sposto solo in una delle due sequenze, confrontando un simbolo con un gap (se l'arco è verticale ho un gap nella stringa che sta sulle colonne e viceversa se ho un arco orizzontale). Il percorso migliore è quello con più diagonali possibili visto che rappresentano un match. La diagonale ha quindi costo $\delta(\sigma, \sigma)$, mentre le altre due mosse hanno costo $\delta(-, \sigma)$ e $\delta(\sigma, -)$. Una matrice di allineamento è quindi un percorso nella griglia.
Ogni volta che ci si allontana dalla diagonale principale si sta inserendo un gap. Potrei quindi ragionare **allineando per banda**, avendo al più k gap e quindi costruendo solo una banda distante al più k dalla diagonale principale, avendo quindi un algoritmo più veloce per costruire l'allineamento.
Si nota quindi che LCS è un MTP e anche la distanza di edit è calcolabile con la griglia.
Per LCS, per due stringhe v e w, si ha infatti che:

$$s_{i,j} = \max \begin{cases} s_{i-1,j} \\ s_{i,j-1} \\ s_{i-1,j-1} + 1 \text{ se } v_i = w_j \end{cases}$$

Quindi nella griglia le diagonali costano 1 e gli altri costano 0.
Quando si fa l'allineamento si sta calcolando in realtà la **distanza** tra due sequenze, calcolando quanto sono diverse. La distanza di edit è una di queste distanze, che rappresenta il minimo numero di operazioni epr trasformare una stringa in un'altra. Un altra è la **distanza di Hamming**, che si riferisce a sequenze di distanza uguale, senza ammettere spazi, calcolando il numero di posizioni in cui si hanno mismatch. Non si hanno inserzioni e delezioni in Hamming, mentre si hanno entrambe in edit.
Le matrici di confronto più note, per il confronto proteico, sono le:

- **PAM (*point accepted mutations*)**
- **BLOSUM (*blcok substitution*)**

Allineamento Multiplo

Ricordiamo che l'allineamento multiplo permette di confrontare più specie, cercando di studiare i cambiamenti evolutivi. L'allineamento multiplo rivela

similarità che con l'allineamento normale non si riescono a comprendere. Per praticità consideriamo tre sequenze e quindi 3 dimensioni.
Confrontando 3 sequenze ho quindi bisogno di un cammino in uno spazio a tre dimensioni. Si usa il cosiddetto **3-D Manhattan cube** che comporta un **3-D edit graph**.
Pensando a come posso arrivare ad un punto ho quindi non più 3 modi, come nel caso a due dimensioni, ma ne ho 7:

$$s_{i,jk} = \max \begin{cases} s_{i-1,j-1,k-1} + \delta(v_j, w_k, u_k) \\ s_{i-1,j-1,k} + \delta(v_j, w_k, \_) \\ s_{i-1,j,k-1} + \delta(v_j, \_, u_k) \\ s_{i,j-1,k-1} + \delta(\_, w_k, u_k) \\ s_{i-1,j,k} + \delta(v_j, \_, \_) \\ s_{i,j-1,k} + \delta(\_, w_k, \_) \\ s_{i,j,k-1} + \delta(\_, \_, u_k) \end{cases}$$

Si arriva a:

$$O(n^3)$$

mentre per k dimensioni si avrebbe:

$$O(2^k n^k)$$

Per k fissato è quindi parametrico ma se non so quante k sequenze ho è un problema particolarmente complesso, esponenziale.
Ogni allineamento multiplo comporta, nei passaggi interni, allineamenti a coppie, per poi "chiudere" tramite la transitività.
Dall'allineamento multiplo induco all'allineamento a coppie ma non vale il viceversa, avendo che partendo da coppie di allineamento potrei avere tipologie diverse di allineamento a parità di sequenza non potendo quindi usare la transitività. Potrei quindi arrivare a inconsistenze.
Per allineare possiamo usare un approccio greedy. Si considerino 4 sequenze, si hanno quindi:

$$\binom{4}{2} = 6$$

possibilità di allineamento. Si usa il **progressive alignment** e non si usa quindi una strategia di programmazione dinamica.
Un altro metodo è quello già anticipato del **sum of pairs score (*SP-Score*)**.
L'allineamento multiplo può anche essere visualizzato come un grafo, tramite il **grafo del pangenoma**. In tale grafo si ha un nodo in corrispondenza di match perfetto e vari branch in presenza di non match. Posso tenere

traccia dei "collassamenti" in un solo nodo dei vari match e della sequenza di provenienza di ogni nodo. Valutando i percorsi studio gli allineamenti anche se potrei avere un percorso non corrispondente ad alcuna sequenza e questo è un side effect. Questa rappresentazione ha i suoi vantaggi, definendo il **partial order multiple sequence alignment** per cui ogni volta che arriva una nuova sequenza il confronto avviene direttamente con il grafo, arricchendolo, ottenendo un modo efficienze per fare allineamento multiplo in modo incrementale. Il creatore di questo metodo ha anche sviluppato un metodo simile per confrontare più grafi.

4.1 Algoritmi di Allineamento

4.1.1 Allineamento Globale di Needleman-Wunsch

Vediamo quindi l'allineamento globale di due sequenze. Si ricorda che:

- **input:** due sequenze, $A = a_1, \dots a_m$ e $B = b_1, \dots, b_n$, definite su algoritmo Σ, e uno schema di punteggio δ
- **output:** un allineamento M tra A e B che corrisponde al'ottimo (a seconda dell'impostazione del problema massimo/minimo) score D

Si ha che:

- il problema è di massimo se δ fornisce una misura di somiglianza tra simboli
- il problema è di minimo se δ fornisce una misura di diversità tra simboli

Dato $A_i = A[1, i]$, ovvero il prefisso di i caratteri di A, e dato $B_J = B[1, j]$, ovvero il prefisso dei primi j caratteri di B, ho che, $\forall i = 0, \dots, m$ e $\forall j = 0, \dots, n$:

$$D(i, j)$$

è lo **score** di allineamento globale tra A_i e B_j.
Si ha quindi che:

$$D(m, n)$$

è lo **score di allineamento globale** tra A e B.
Si ha quindi che:

$$D(i,j) = opt \begin{cases} D(i-1, j-1) + \delta(a_i, b_j) \\ D(i-1, j) + \delta(a_i, -) \\ D(i, j-1) + \delta(-, b_j) \end{cases}$$

avendo che, per un certo k, s e d (in base ai quali si capisce se è un problema di massimo o minimo):

- $\delta(a_i, b_j) = k$ se $a_i = b_j$
- $\delta(a_i, b_j) = s$ se $a_i \neq b_j$
- $\delta(a_i, -) = \delta(-, b_j) = d$

Come casi base si ha quindi:

- $D(0, 0) = 0$
- $D(i, 0) = i \cdot d = D(i - 1, 0) + d$
- $D(0, j) = j \cdot d = D(0, j - 1) + d$

L'algoritmo consiste quindi in:

1. riempire la matrice di score calcolando tutti i $D(i, j)$
2. individuare la cella ottima, ovvero $D(m, n)$
3. ricostruzione della matrice di allineamento M a partire da tale cella ottima

Bisogna quindi capire come riempire tale matrice D, dati k, s e d:

1. si specifica che D è una matrice $(m + 1) \times (n + 1)$, con righe indicizzate per $i \in [0, m]$ e colonne indicizzate per $j \in [0, n]$. La riga di incide $i, i > 0$, corrisponde ad a_i e la colonne di indice $j, j > 0$ al simbolo b_j
2. inizializzo $D(i, 0) = i \cdot d, \forall i \in [0, m]$ (quindi inizializzo la prima colonna) e $D(0, j) = j \cdot d, \forall j \in [0, n]$ (quindi inizializzo la prima riga)
3. scorro riga per riga (o colonna per colonna) e riempio al matrice secondo il calcolo sopra definito per $D(i, j)$. Tengo traccia della cella di partenza da cui ho ottenuto l'ottimo (se quella sopra, sotto o sulla diagonale). Se ho pari valori posso scegliere di dare priorità a considerare di aver fatto un movimento sula diagonale
4. identifico in $D(m, n)$ la cella ottima

In merito alla ricostruzione dell'allineamento si procede quindi a ritroso:

1. si inizializza una matrice M di allineamento vuota
2. parto dalla cella ottima di D e seguo un percorso a ritroso fino alla cella $(0, 0)$
3. ogni passaggio da una cella aggiungo una coppia di simboli a M. Si hanno quindi tre casi:
    - il punteggio migliore lo ottengo dalla cella sulla diagonale (ovvero l'ottimo è il valore $D(i-1, j-1)+\delta(a_i, b_j)$) e in tal caso aggiungo:
    $$\binom{a_i}{b_j}$$
    alla matrice di allineamento M
    - il punteggio migliore lo ottengo dalla cella in alto (ovvero l'ottimo è il valore $D(i-1, j) + \delta(a_i, -)$) e in tal caso aggiungo:
    $$\binom{a_i}{-}$$
    alla matrice di allineamento M
    - il punteggio migliore lo ottengo dalla cella in basso (ovvero l'ottimo è il valore $D(i, j-1) + \delta(-, b_j)$) e in tal caso aggiungo:
    $$\binom{-}{b_j}$$
    alla matrice di allineamento M

Questo algoritmo ha complessità sia in tempo che in spazio pari a:

$$O(m \cdot n)$$

Su slide esempio pratico.

4.1.2 Allineamento Locale di Smith-Waterman

Vediamo quindi l'allineamento locale di due sequenze. Si ricorda che:

- **input:** due sequenze, $A = a_1, \dots a_m$ e $B = b_1, \dots, b_n$, definite su algoritmo Σ, e uno schema di punteggio δ

- **output:** la coppia di sottostringhe di A e B a cui corrisponde il massimo (l'ottimo in questo caso può solo essere il massimo) score di allineamento globale

Vista la natura del problema, che può essere solo di massimo, si ha che, riprendendo la stessa nomenclatura dell'algoritmo di Needleman-Wunsch, si ha che lo score δ è solo di **similarità**, avendo quindi che:

- $s, d < 0$
- $k > 0$

L'algoritmo di Smith-Waterman permette di identificare una regione comune a due sequenze che non verrebbe evidenziata dall'allineamento globale con Needleman-Wunsch. Un approccio di allineamento globale permette di evidenziare le sottostringhe più simili.
Si procede quindi fissando i e j e, considerando tra tutte le coppie di sottostringhe che finiscono in a_i su A e b_j su B quella che ha il massimo score di allineamento globale $D(i, j)$.
Si ha che:

$$\max\{D(i, j)\}$$

è lo score di allineamento locale tra A e B.
Il calcolo di $D(i, j)$ varia quindi rispetto all'algoritmo di Needleman-Wunsch in quanto, per come sono stati definiti k, s e d, potrei avere valori negativi ma si impone un limite inferiore pari a 0 che tonerà poi utile nell'algoritmo.
Si ha quindi:

$$D(i, j) = \max \begin{cases} D(i-1, j-1) + \delta(a_i, b_j) \\ D(i-1, j) + \delta(a_i, -) \\ D(i, j-1) + \delta(-, b_j) \\ 0 \end{cases}$$

con:

- $\delta(a_i, b_j) = k$ se $a_i = b_j$
- $\delta(a_i, b_j) = s$ se $a_i \neq b_j$
- $\delta(a_i, -) = \delta(-, b_j) = d$

Anche i casi base variano, avendo che la prima riga e la prima colonna sono interamente inizializzate a 0:

- $D(0, 0) = 0$

- $D(i, 0) = 0, \forall i = 1, \ldots m$
- $D(0, i) = 0, \forall j = 1, \ldots n$

L'algoritmo è analogo, a livello di "procedura" (salvo ovviamente l'individuazione della cella ottima), a quello di Needleman-Wunsch e consiste quindi in:

1. riempire la matrice di score calcolando tutti i $D(i, j)$
2. individuare la cella ottima, ovvero la cella con in massimo valore in D
3. ricostruzione della matrice di allineamento M a partire da tale cella ottima

Bisogna quindi capire come riempire tale matrice D, dati k, s e d:

1. si specifica che D è una matrice $(m + 1) \times (n + 1)$, con righe indicizzate per $i \in [0, m]$ e colonne indicizzate per $j \in [0, n]$. La riga di incide $i, i > 0$, corrisponde ad a_i e la colonne di indice $j, j > 0$ al simbolo b_j
2. inizializzo a 0 la prima riga e la prima colonna
3. scorro riga per riga (o colonna per colonna) e riempio al matrice secondo il calcolo sopra definito per $D(i, j)$. Si ricorda che se dalle tre direzioni si ha massimo negativo si inserisce 0
4. identifico $D(j, i)$ la cella ottima $D(j, i)$

In merito alla ricostruzione dell'allineamento si procede quindi a ritroso:

1. si inizializza una matrice M di allineamento vuota
2. parto dalla cella ottima di D e seguo un percorso a ritroso fino alla prima cella contente 0 che si incontra
3. ogni passaggio da una cella aggiungo una coppia di simboli a M. Si hanno quindi tre casi:
   - il punteggio migliore lo ottengo dalla cella sulla diagonale (ovvero l'ottimo è il valore $D(i-1, j-1)+\delta(a_i, b_j)$) e in tal caso aggiungo:
     $$\binom{a_i}{b_j}$$
     alla matrice di allineamento M

- il punteggio migliore lo ottengo dalla cella in alto (ovvero l'ottimo è il valore $D(i-1,j)+\delta(a_i,-)$) e in tal caso aggiungo:
$$\begin{pmatrix} a_i \\ - \end{pmatrix}$$
alla matrice di allineamento M
- il punteggio migliore lo ottengo dalla cella in basso (ovvero l'ottimo è il valore $D(i,j-1)+\delta(-,b_j)$) e in tal caso aggiungo:
$$\begin{pmatrix} - \\ b_j \end{pmatrix}$$
alla matrice di allineamento M

Questo algoritmo ha complessità sia in tempo che in spazio pari a:

$$O(m \cdot n)$$

Su slide esempio pratico.

4.1.3 Allineamento Semiglobale

Vediamo quindi l'allineamento semiglobale tra due sequenze. Si ha che;

- **input:** due sequenze, $A = a_1, \dots a_m$ e $B = b_1, \dots, b_n$, definite su algoritmo Σ, e uno schema di punteggio δ
- **output:** si hanno 8 possibili output:
  1. il prefisso di A più simile a B
  2. i due prefissi più simili
  3. i due suffissi più simili
  4. la sottostringa di A e il suffisso di B più simili
  5. il suffisso di A più simile a B
  6. la sottostringa di A più simile a B
  7. la sottostringa di A e il prefisso di B più simili
  8. il suffisso di A e il prefisso di B più simili

Bisogna quindi determinare opportunamente $D(i,j)$ a partire da:

- le equazioni di ricorrenza per il calcolo di $D(i,j)$

- i valori di $D(i, 0)$ e $D(0, j)$
- particolari valori i e j che risolvano il problema

A questo punto si può:

- riempire la matrice degli score D
- trovare la cella ottima
- ricostruire la matrice di allineamento M andando a ritroso a partire dalla cella ottima di D

Vediamo quindi i vari casi (**per ognuno esempio pratico su slide**) per ottenere gli 8 output:

1. **ricerca del prefisso di A più simile a B**. Per farlo si ricerca tra tutti i prefissi di A quello con il massimo score di allineamento globale con B. In altri termini $D(i, j)$ è lo score di allineamento globale tra il prefisso di A che finisce in posizione i e il prefisso di B che finisce in posizione j e nel dettaglio:
    - $D(i, n)$ è lo score di allineamento globale tra il prefisso di A che termina in i e tutta la sequenza B
    - i_m, tale che $D(i_m, n)$ è massimo, è la posizione di fine su A del prefisso che ha il massimo score di allineamento globale con tutta B. Applico quindi Needleman-Wunsch con ottimo pari al massimo presente nella colonna di indice n di D, ripartendo da lì per ricostruire M (ricostruendo quindi fino a che non arrivo in $D(0, 0)$
2. **ricerca dei due prefissi più simili**, ovvero tra tutte le coppie di prefissi di A e di B cercare quella che ha il massimo score di allineamento globale. In altri termini $D(i, j)$ è lo score di allineamento globale tra il prefisso di A che finisce in posizione i e il prefisso di B che finisce in posizione j e nel dettaglio:
    - i_m, j_m, tali che $D(i_m, j_m)$ è massimo, sono rispettivamente la posizione di fine su A e B dei prefissi che hanno il massimo score di allineamento globale. Applico quindi Needleman-Wunsch con ottimo pari al massimo presente nella matrice D, ovvero tale $D(i_m, j_m)$ e parto da lì per ricostruire M (ricostruendo quindi fino a che non arrivo in $D(0, 0)$

3. **ricerca dei due suffissi più simili**, ovvero tra tutte le coppie di suffissi di A e di B cercare quella che ha il massimo score di allineamento globale. In altri termini $D(i, j)$ è lo score di allineamento globale tra la coppia di sottostringhe più simili che finiscono in posizione i su A e in posizione j su B. Nel dettaglio:

    - $D(m, n)$ è lo score di allineamento globale tra la coppia di suffissi che hanno il massimo score di allineamento globale. Applico quindi Smith-Waterman con cella ottima pari a $D(m, n)$ e ricostruisco, ovvero costruisco M procedendo a ritroso dalla cella ottima fino a che non trovo uno 0 nella matrice D. Tale 0 si trova in un certo $D(i_0, j_o)$ e ho che $i_0 + 1$ e $j_0 + 1$ sono rispettivamente la posizione di inizio del suffisso su A e B

4. **ricerca della sottostringa di A e del suffisso di B che sono più simili**, ovvero tra tutte le coppie formate da una sottostringa di A e un suffisso di B cercare quella che ha il massimo score di allineamento globale. In altri termini $D(i, j)$ è lo score di allineamento globale tra coppie di sottostringhe più simili che finiscono in posizioni i di A e j di B. Nel dettaglio:

    - $D(i, n)$ è lo score di allineamento globale tra la coppia più simile composta da una sottostringa che finisce in posizione i su A e un suffisso di B
    - i_m, tale che $D(i_m, n)$ è massimo, è la posizione di fine su A della sottostringa su A più simile ad un suffisso di B. Applico quindi Smith-Waterman con ottimo pari al massimo presente nella colonna di indice n di D, ripartendo da lì per ricostruire M, ricostruendo quindi fino a che non arrivo in al primo 0. Tale 0 si trova in un certo $D(i_0, j_o)$ e ho che $i_0 + 1$ e $j_0 + 1$ sono rispettivamente la posizione di inizio della sottostringa su A e del suffisso B

5. **ricerca del suffisso di A più simile a B**, ovvero tra tutti i suffissi di A cercare quello ha il massimo scoredi allineamento globale con tutta la sequenza B. In altri termini $D(i, j)$ è lo score di allineamento globale della coppia più simile formata da una sottostringa di A che finisce in posizione i e il prefisso di B che termina in posizione j. Costruisco D in modo che sia inizializzata

come previsto per Needleman-Wunsch ma con la prima colonna pari a 0. Nel dettaglio:

- $D(m, n)$ è lo score di allineamento globale tra tutta la sequenza B e il suffisso più simile su A. Applico quindi Needleman-Wunsch con ottimo pari a $D(m, n)$ e parto da lì per ricostruire M, ricostruendo quindi fino a che non arrivo in $D(i_0, 0)$. Si ha che $i_0 + 1$ è la posizione di inizio del suffisso su A

6. **ricerca della sottostringa di A più simile a B**, ovvero tra tutte le sottostringhe di A cercare quella ha il massimo score di allineamento globale con tutta la sequenza B. In altri termini $D(i, j)$ è lo score di allineamento globale della coppia più simile formata da una sottostringa di A che finisce in posizione i e il prefisso di B che termina in posizione j. La matrice D è la inizializzata come la D di Needleman-Wunsch ma con tutti 0 sulla prima colonna. Nel dettaglio:

   - $D(i, n)$ è lo score di allineamento globale tra tutta B e la sottostringa di A più simile
   - i_m, tale che $D(i_m, n)$ è massimo, è la posizione di fine su A della sottostringa più simile a B. Applico quindi Needleman-Wunsch con ottimo pari a $D(i_m, n)$ e parto da lì per ricostruire M, ricostruendo quindi fino a che non arrivo in $D(i_0, 0)$. Si ha che i_0 è la posizione di inizio della sottostringa su A mentre i_m quella di fine

7. **ricerca della sottostringa di A e del prefisso di B più simili**, ovvero tra tutte le coppie formate da una sottostringa di A e un prefisso di B cercare quella che ha il massimo score di allineamento globale. In altri termini $D(i, j)$ è lo score di allineamento globale tra la coppia formata da una sottostringa di A che finisce in posizione i e il suffisso di B che finisce in posizione j. La matrice D è la inizializzata come la D di Needleman-Wunsch ma con tutti 0 sulla prima colonna e nel dettaglio:

   - i_m, j_m, tali che $D(i_m, j_m)$ è massimo, sono rispettivamente la posizione di fine su A e B della sottostringa e del prefisso più simile su A e B. Applico quindi Needleman-Wunsch con ottimo pari al massimo presente nella matrice D, ovvero tale $D(i_m, j_m)$ e parto

da lì per ricostruire M (ricostruendo quindi fino a che non arrivo in $D(i_0, 0)$. Si ha che i_m e j_m sono rispettivamente la posizione di fine della sottostringa su A e del prefisso su B. Inoltre $i_0 + 1$ è la posizione di inizio della sottostringa su A

8. **ricerca del suffisso di A e del prefisso di B più simili**, ovvero tra tutte le coppie formate da un suffisso di A e un prefisso di B cercare quella che ha il massimo score di allineamento globale. In altri termini $D(i, j)$ è lo score di allineamento globale della coppia più simile formata da una sotto stringa di A che finisce in posizione i e il prefisso di B che termina in posizione j. D è costruita come la matrice di Needleman-Wunsch con però la prima colonna a 0 nel dettaglio:

    - $D(m, j)$ è lo score di allineamento globale tra il prefisso di B che termina in j e il suffisso di A più simile
    - j_m, tale che $D(m, j_m)$ è massimo, è la posizione di fine su B del prefisso più simile ad un suffisso di A. Applico quindi Needleman-Wunsch con ottimo pari al massimo presente nell'ultima riga di D, ripartendo da lì per ricostruire M, ricostruendo fino a che non arrivo in $D(i_0, 0)$. S ha chej_m è la posizione di fine del prefisso su B mentre $i_0 + 1$ è la posizione di inizio del suffisso su A

Capitolo 5

Alberi Filogenetici

Vediamo come ricostruire la storia evolutiva in bioinformatica.
Useremo la nozione di **albero evolutivo**.
L'evoluzione è guidata da, per Darwin:

- **diversità**, con individui diversi che portano variazioni genetiche
- **mutazioni**, ovvero cambiamenti nella sequenze di DNA, con mutazioni nucleotidiche o genomiche
- **selezione naturale**, con mutazioni che favoriscono la sopravvivenza o altre che la impediscono, portando morte o impossibilità di trasmettere la mutazione

Definizione 25. *Definiamo* ***filogenesi****, o* ***albero filogenetico****, come un albero che descrive le sequenze di eventi di speciazione che hanno portato alle specie attuali, rappresentate nelle foglie di questo albero. I nodi interni non rappresentano specie viventi.*

La costruzione di un albero si usano i **caratteri morfologici** (lunghezza, numero di arti etc...) e attualmente anche l'informazione molecolare data da sequenze geniche e proteiche. Un classico tipo di sequenza genomica usato è il gene dell'emoglobina, che è un gene si è evoluto con mutazioni specifiche per le specie che si analizzano.

La **topologia** è la forma dell'albero evolutivo. Ogni nodo interno è un antenato comunque delle specie nelle foglie del sottoalbero che ha tale nodo come radice

La filogenesi dipende dall'input, usando ad esempio il **DNA mitocondriale** (che è un DNA particolarmente conservato) rispetto che altro, come

l'emoglobina, si ottengono alberi diversi, studiando nel primo caso la somiglianza a livello di DNA mitocondriale. Un altro tipo di DNA spesso usato è il **DNA nucleare**.
Con il termine **speciazione** si indica la creazione di specie differenti, portando a due gruppi con una certa differenza genetica. Ogni coppia di specie da questi due gruppi condivide uno stesso antenato.
Quando si ha la speciazione si ottengono due gruppi con differenza genetica predominante, quindi molto "distanti".
Si hanno varie assunzioni:

- organismi vicini hanno genomi simili
- geni simili sono omologhi, ovvero hanno lo stesso antenato. A seguito di una duplicazione genica però si possono avere eventi di speciazione che porterà poi a specie che differiscono
- esiste un antenato universale comune a tutti, si ipotizza
- le differenze molecolari in geni omologhi sono correlate al tempo di evoluzione. Geni duplicato omologhi hanno differenze molecolari dipendenti dalla distanza evolutiva rispetto al tempo in cui è avvenuto l'evento di speciazione (???). Ad esempio le emoglobine di specie differenti, quanto più le specie sono vicine, tanto meno si hanno differenze a livello di mutazioni

Vediamo l'albero. Si hanno:

- le foglie sono le specie attuali
- la lunghezza/costo degli archi è correlata alla distanza con l'antenato comune
- i nodi interni sono antenati comuni

A seconda dell'evento si hanno geni omologhi per diversi motivi:

- **orthologs** dopo una speciazione
- **paralogs** dopo una duplicazione
- **xenolog** dopo un trasferimento orizzontale, come quello di un virus

Abbiamo anche un **albero di geni** dove nelle foglie si ha la sequenza genica, con varie copie dello stesso gene dopo un evento di speciazione (magari la versione 1A, la versione 2A e quella 3A). Posso avere sia eventi di duplicazione che di speciazione.
Un evento di duplicazione non dovrebbe creare nuove specie, cosa che avviene con la speciazione.
Posso avere alberi di diverso tipo:

- **con radice**, che comporta più versioni a parità di specie, quando si vuole cercare l'albero migliore che rappresenti l'evoluzione
- **senza radice**, dove comunque posso studiare percorsi tra specie

Una radice in un albero, qualora non sia presente, si sceglie un **outgroup**, che è molto distante dalle specie di interesse e quindi lo si attacca direttamente alla radice.
Si hanno tre metodi di ricostruzione per l'evoluzione:

1. basato su **distanza**, dove ricorsivamente si combinano due nodi a minima distanza, collegandole e creando antenati comuni. Circa quello che si fa nel clustering con il neighbor joining
2. basato su **parsimonia**, dove si minimizza il numero di cambiamenti. È il **rasoio di Occam**
3. basato su **maximum likelihood**, dove si usano reti Bayesiane a forma di albero. Sia associa una misura all'albero e si fa un calcolo statistico. È un metodo oneroso

5.1 Metodi basati su distanza

Si ha:

- **input**: matrice delle distanze tra le specie, calcolate tramite allineamento globale e si fa un calcolo basato sulle differenze. Si ha il numero delle mutazioni richieste
- **outline**: si clusterizzano le specie, avendo all'inizio dei singleton. Ad ogni iterazione si combinano due cluster vicini in uno nuovo, tramite un algoritmo come UPGMA, che si basa su neighbor joining. Quindi si clusterizza e si crea l'albero. I singleton sono le foglie e ogni clusterizzazione successiva un antenato ai due nodi dei cluster che unisco. Dopo la creazione di un cluster devo riaggiornare le distanze tra il nuovo cluster e i cluster restanti

- **output**: una filogenesi con radice

esempio su slide. Bisogna capire come calcolare la distanza tra cluster.

Definizione 26. *Si definisce* ***orologio molecolare*** *come l'assunzione per cui tutte le foglie sono allo stesso livello, dimenticando i tempi di evoluzione, avendo tempo identico su tutti i cammini. È una forzatura della realtà in quanto normalmente l'orologio molecolare dovrebbe tenere traccia del tempo reale.*
Un albero soddisfa l'assunzione dell'orologio molecolare quando le foglie sono tutte allo stesso livello.

UPGMA costruisce una filogenesi con radice. Si hanno punti nello spazio e in modo iterativo si raggruppano i cluster.
Si ha quindi:

- **input**: una matrice D $n \times n$ con l'insieme delle specie $S = \{s_1, \dots s_n\}$, con all'interno le distanze tra due specie, ovvero il numero di mutazioni necessarie per passare da una specie all'altra. Nella matrice D ho quindi le n specie che identificano righe e colonne. La matrice è ovviamente simmetrica. Per capire il numero di mutazioni si usa l'allineamento globale usando apposite matrici di punteggio. Tramite queste distanze si ricostruisce l'evoluzione.

- **output:** un albero filogenetico T le cui foglie sono le specie (che in questo caso sono rappresentate come sequenze) e l'albero è tale che $\forall s_i, s_j \in S$ si ha che $D(s_i, s_j) = d_T(s_i, s_j)$, ovvero è uguale alla distanza stimata nell'albero. Si dice che T è consistente con la matrice D

Vediamo quindi come fare un algoritmo per ottenere T da D. È un algoritmo base di **neighbor joining**.
Si ha un'iterazione di passi:

1. tra tutte le coppie di specie della matrice D scelgo la coppia x, y tale che $D(x, y)$ è la minima

2. aggiungo x e y all'albero T collegandole con un nodo padre che rappresenta $\{x, y\}$

3. si assegna agli archi che collegano le due specie al nodo $\{x, y\}$ dei pesi pari a $\frac{D(x,y)}{2}$

4. aggiorno D:
    - sostituisco x e y con $\{x, y\}$
    - preso un nodo z si ha che $D(z, \{x, y\}) = D(z, x) - D(x, \{x, y\})$
    - chiamo questa nuova D come D', si noti che sarà di dimensioni minori
5. ripeto dal passo 1

Non tutte le matrici ammettono un albero (questo a causa della semplicità della formula dell'algoritmo) perché devono valere certe condizioni. La matrice deve essere **ultrametrica**. Per rendere una matrice una valida in caso posso modificare i costi, normalizzando la matrice delle distanze.
Nei metodi basati su distanze la matrice in input deve avere distanze consistenti con la costruzione dell'albero. Qualora non lo fosse si devono normalizzare in qualche modo le distanze.
Per UMGMA si avrebbe invece una formula più complessa che non viene approfondita nel corso. Comunque, dati C_i e C_j due cluster si ha che la loro distanza è pari a:

$$d(C_i, C_j) = \frac{1}{|C_i||C_j|} \sum_{p \in C_i} \sum_{q \in C_j} d(p, q)$$

e quando si combinano per ottenere C_k si ha, dato un qualsiasi altro C_j:

$$d(C_k, C_l) = \frac{|C_i| d(C_i, C_l) + |C_j| d(C_j, C_l)}{|C_i||C_j|}$$

si definisce quindi il nodo k e si pongono i nodi a profondità:

$$\frac{d(C_i, C_j)}{2}$$

UPGMA soddisfa l'orologio molecolare. Lato score abbiamo ad esempio la **matrice Jukes-Cantor**, che assume eguale tasso di mutazione, che confronta singoli nucleotidi (usando pesi negativi per la diagonale principale dove non si hanno mutazioni). D'altro canto si ha che ogni altra mutazione è egualmente probabile.
Si ha anche il **modello Kimura** usato per le proteine, che da un peso diverso a transizioni e trasversioni (dove una purina diventa una piramidina), avendo quindi score diversi a seconda della coppia di basi. Questo modello considera la chimica del DNA.

5.1.1 Metodi Basati su Caratteri

Il modello discreto basato su caratteri è importante per diversi aspetti. La storia evolutiva è basata anche su caratteri discreti e non solo su distanze. La presenza di caratteristiche è un aspetto generale e ha importanza nella ricostruzione della filogenesi tumorale, tramite la **infinite site assumption**, per la quale si ritiene che una volta acquisita una mutazione resta per sempre, ottenendo la **filogenesi perfetta**.
Le mutazioni puntuali però spesso improvvisamente spariscono non potendo più usare l'assunzione. Si procede quindi a costruire l'albero evolutivo tramite caratteristiche comuni delle varie specie. Quando compare un carattere discreto in un nodo non viene più perso e tutte le specie sottostanti hanno quella caratteristica. Ogni carattere appare una volta sola nell'albero.
Bisogna tradurre l'albero in una matrice le colonne sono etichettate dalle caratteristiche e le righe dalle specie. Si ha che $a_{ij} = 1$ se la specie i ha la caratteristica j, 0 altrimenti.
Gusfield trovò un algoritmo lineare per costruire tale albero.
Tra gli scopi dell'algoritmo si ha lo studio di genotipi e aplotipi, essendo tra l'altro l'uomo un organismo diploide. Gli organismi diploidi hanno due copie di ogni cromosoma con piccole differenze. Il problema fondamentale è che sequenziando non ho modo di capire se un mismatch sia dovuto al fatto che ho le due sequenze materne e paterne.
L'uomo è anche biallelico, avendo che le opzioni per una certa posizione, sono due, che si indicano con 0 e 1. Si ha che 0 è l'allele dominante e 1 è detto allele di minoranza o allele recessivo. Se su due aplotipi ho valori diversi si parla di eterozigosi rispetto un certo allele.
La radice di un albero dove 0 rappresenta l'assenza di allele non si ha alcuna mutazione.
Si ha quindi il seguente problema:

- **input**: una matrice di aplotipi su $\{0, 1, *\}$
- **output**: una matrice binaria di aplotipi che risolve i genotipi in M e ammette una filogenesi perfetta

Ultima lezione da rivedere interamente.
Riassumendo il problema fondamentale è quello della ricostruzione dell'evoluzione tenuto conto che essa è legata a eventi, ovvero mutazioni, di tipo nucleotidico (che studiamo con l'allineamento globale) e di tipo genomico (che vedremo essere studiate tramite riarrangiamenti, con trasposizioni, inversioni, duplicazione genica etc...), anche a livello di cromosomi.
L'evoluzione è legata a questi eventi che comportano speciazione.

Si hanno quindi metodi basati su distanze, caratteri e parsimonia.
SI usa il DNA mitocondriale perché è quello che si "porta dietro" la storia delle specie, essendo quello meno modificato nel corso della storia. Si usa anche quello nucleare. Spesso si hanno problemi di **riconciliazione** dell'evoluzione, infatti prendendo geni diversi delle stesse specie ottengo alberi evolutivi diversi e quindi si vuole appunto "riconciliare" tali alberi, ottenendo l'**agreement** di essi. Anche metodi computazionali diversi producono alberi diversi e non si può sapere facilmente quale sia giusto. Ricostruire l'albero della vita è comunque un tempo di spazio e tempo computazionale.
In generale gli alberi evolutivi sono utilizzati nell'analisi delle sequenze. Quando si fa allineamento, specialmente multiplo, si avvale anche dell'albero filogenetico per comparare le sequenze. Nella matrice di allineamento multiplo il costo di una colonna è legato al costo degli archi in un albero evolutivo, che quindi è usato per stimare l'allineamento. Un esempio pratico è l'analisi del SARS-CoV2 dove si fanno le analisi in termini di mutazioni, associando alle sequenze dei cluster che seguono un'evoluzione nel tempo.
L'evoluzione non riguarda solo le specie ma anche l'inferenza di aplotipi da genotipi e anche l'evoluzione tumorale. Anche nel caso dei tumori ho alberi diversi a seconda del metodo ma in questo caso anche i nodi interni sono etichettati, a differenza dell'albero filogenetico per le specie.

5.1.2 Metodi Basati su Parsimonia

In questi metodi su ha che l'ipotesi più veritiera è quella che minimizza gli eventi, ovvero con il **rasoio di Occam**. Questo metodo si applica anche in altri contesti in quanto la natura è in se "parsimoniosa". Si assume in questo metodo che ciascuna specie è specificata dai caratteri o stati di attributi. Un albero di massima parsimonia ha come foglie gli tali stati di attributi delle specie e i nodi interni sono etichettati con i cambiamenti di stato, che, in tutto l'albero, sono minimizzati. Nel dettaglio un **carattere** è ad esempio un fenotipo, ovvero un certo attributo visibile, ma anche una certa informazione genomica (tramite ad esempio lo studio degli SNPs). Questi ultimi sono anche detti **caratteri genomici**. Con il termine **WGS (*whole genome scale*)** si indicano caratteri e attributi che sono ottenuti da lavori a larga scala. WGS fornisce vari caratteri o attributi, come indicazioni sui geni, combinazioni proteiche, SNP, studio di esoni e introni etc. . ..
La scelta dei caratteri influenza l'albero. I caratteri possono essere binari con 0 assenza e 1 presenza, usando quindi un vettore binario per gli stati.
Si ha la **parsimonia di Dollo**, dove si ha un solo passaggio da 0 a 1 ma tanti passaggi da 1 a 0 per un certo carattere c. Quindi dati c_i caratteri ho un albero che presenta l'etichetta c_i una e una sola volta. Una volta comparso un

carattere comunque può sparire. Le foglie sono quindi gli stati finali, ovvero le specie attuali, e i nodi interni degli stati che poi sono cambiati. Si hanno quindi:

- **input:** le foglie
- **output:** l'albero

Nel modello **Camin-Sokal** si ha invece che un carattere viene guadagnato ma non perso, ho quindi che c può passare da 0 a 1 quante volte si vuole. Posso quindi avere più archi etichettati con un certo c_i. Questo modello non si può usare per le assunzioni fatte con gli alberi tumorali. Questo modello ammette che due specie possano acquisire in modo indipendentemente una caratteristica.

Filogenesi Perfetta Binaria

Combinando, ovvero facendo l'intersezione, i modelli di Dollo e Camin-Sokal ottengo la **filogenesi perfetta binaria**, dove posso avere una sola volta il cambiamento da 0 a 1 e mai i cambiamenti da 1 a 0. Un carattere etichetta quindi un solo arco dell'albero e una volta guadagnato non è mai perso. Ho quindi un **gain** e **no loss**. Tale modello è risolvibile in tempo lineare mentre i due problemi di partenza sono **NP-complete**.
Per la filogenesi binaria perfetta si ha:

- **input:** una matrice binaria $n \times m$, con n specie, sulle righe s_i, e m, sulle colonne c_i caratteri
- si crea un albero dove ogni nodo x è etichettato da un vettore lungo m, detto l_x, dove $l_x[j]$ è lo stato del carattere c_j. La radice è etichettata con un vettore di m zeri. Se sono in $(0, 0, 0)$, ipotizzando tre caratteri, e passo con un arco etichettato con c_i ad un altro nodo esso sarà etichettato con $(1, 0, 0)$. Per ogni carattere c_j esiste esattamente un arco e etichettato c_j con c_j che rappresenta il cambiamento $0 \to 1$. Ciascuna riga della matrice etichetta esattamente una foglia dell'albero
- **output:** l'albero, se esiste

Se faccio, nell'albero di filogenesi perfetta, in un cammino da radice a foglia incontro solo i cambiamenti di stato che portano ad ottenere la foglia.

La matrice in input potrebbe non permettere la costruzione dell'albero, per le condizioni imposte. Ad esempio:

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}$$

è una **matrice proibita**.
La filogenesi perfetta è una **dollo 0**, ovvero non posso perdere caratteri.
Posso avere una **dollo k** dove il carattere può essere perso k volte.
Il calcolo della **dollo 0** è lineare per l'algortimo di Gusfield e ogni foglia sottostante un nodo dove si è acquisito un carattere contiene quel carattere.
La **dollo 1** si pensa sia comunque polinomiale e si usa nella filogenesi tumorale.
La **dollo k**, per $k \geq 2$ è NP-complete con però algoritmi parametrici.
La matrice proibita la posso rappresentare assumendo la loss del carattere.
Capiamo quando una matrice binaria ammette filogenesi perfetta.

Teorema 2. *Una matrice binaria ammette filogenesi perfetta se non contiene la matrice proibita.*

Ma possiamo dirlo diversamente.

Definizione 27. *Definiamo:*

$$O_j = \{i |\, M_{ij} = 1\}$$

come l'insieme di tutte le specie i con il carattere j.

Definizione 28. *Una collezione di insiemi $O_1, \dots O_n$ è detta **laminare** se per $\forall i, j$, con $1 \leq i$ e $j \leq n$, o O_i e O_j sono disgiunti o un insieme include l'altro.*

Teorema 3. *Una matrice binaria è una filogenesi perfetta sse la collezione delle colonne di M è laminare.*

Quindi studio le colonne delle matrici. Per ogni coppia di colonne o gli insiemi, rappresentati dai vettori colonna, sono disgiunti o uno contiene l'altro.
La laminarità corrisponde alla non esistenza della matrice proibita. Infatti, prendendo la matrice, $(0, 1, 1)$ e $(1, 0, 1)$ sono disgiunti ma nessuno dei due contiene l'altro.

Teorema 4. *La laminarità è una condizione necessaria e sufficiente per la costruzione dell'albero.*

Potrei quindi confrontare ogni colonna della matrice, in $O(n \cdot m^2)$ ma si può fare di meglio, avendo che la lettura di una matrice è in $O(n \cdot m)$.
Per vedere se una matrice è laminare, per Gusfield:

- si riordinano le colonne per numero di 1 non crescenti (la prima colonna è quella con più 1 e l'ultima quella con meno). Questo porta vantaggi anche per l'albero in quanto il primo carattere, quello della prima colonna nella matrice riordinata, è quello che probabilmente è in cima all'albero. Chiamo M tale matrice

- si costruisce una matrice ausiliaria quando si legge M. In questa nuova matrice L, quando si legge M, si memorizza per ogni posizione di M che è a 1 dove si trova a sinistra di quella posizione il primo 1 (che tra i vari 1 a sinistra è quindi quello più destra). Qualora tale 1 non esistesse (sto leggendo il primo 1 di una riga) si mette -1 in L. Se si legge 0 in M metto 0 in L.
  Qualora si faccia la matrice ausiliaria di una matrice proibita si ha che nelle colonne si ottengono colonne con valori non nulli diversi (ho quindi -1 e 1)

- eseguo il vero e proprio test, ovvero, per ogni colonna della matrice ausiliaria, verifico di avere solo valori uguali non nulli in quanto significherebbe

Capitolo 6

I dati in Bioinformatica

Dobbiamo introdurre i dati fondamentali della bioinformatica, ovvero **DNA** (figura 6.1) e **RNA**, intese come molecole biologiche che vengono trattate come stringhe.
Il **DNA (*acido deossiribonucleico*)** è una molecola composta da nucleotidi, che è formato da:

- il deossiribosio, uno zucchero, **D**
- un gruppo fosfato, **p**
- una base azotata (Adenina, Citosina, Guanina, Timina). Citosina e Timina sono dette **piramidine**, le altre due **purine**

Si ha la cosiddetta **direzione 5'3'** per leggere le sequenze del DNA (avendo la direzione posso leggere in sequenza le basi, solitamente è dal basso all'alto).
I nucleotidi si legano tramite **legame fosfodiesterico tra D e P**.
L'unità di lunghezza di un molecola di DNA è il **base pair (*bp*)** (base pair in quanto il DNA è in se un'accoppiamento di basi a doppia elica), ovvero il numero delle basi.
Il **genoma** è la lunga molecola di DNA contenuta in ogni nucleo cellulare. Nel 1953 Watson e Crick hanno scoperto essere a doppia elica. Il genoma è frammentato in **cromosomi** (22 autosomi e i due cromosomi X e Y). Il più lungo cromosoma è il primo e il più corto è il ventiduesimo.
Tra le due catene di un genoma si ha che la Timina si appaia all'Adenina (e viceversa) mentre la Citosina alla Guanina (e viceversa), questa è la **regola delle basi complementari**. Se una catena ha la direzione 5'3' dall'alto verso il basso l'altra lo ha dal basso all'alto (ma sempre da 5' a 3') e viceversa. Si legge sempre in direzione 5'3' e si hanno così le **due sequenze primarie appaiate**, una detta **forward strand (+,1,+1)** e una detta **reverse**

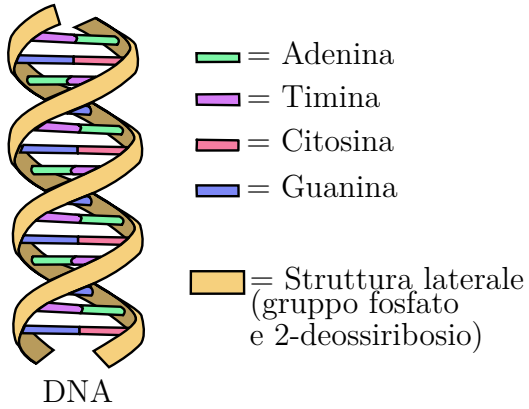


Figura 6.1: Rappresentazione grafica del DNA tratta da `https://it.wikipedia.org/wiki/DNA`

strand (-,-1).
Tra Adenina e Timina abbiamo due legami idrogeno mentre tra Citosina e Guanina se ne hanno tre (si richiede quindi più energia per eventualmente separare).
Si ha quindi a che fare con un alfabeto $\Sigma = \{a, c, g, t\}$ o $\Sigma = \{A, C, G, T\}$ per costruire le sequenze. Data la sequenza primaria di una delle due catene del DNA genomico, la sequenza primaria della catena appaiata è ottenuta per mezzo di un'**operazione di reverse&complement**, ottenuta sostituendo alla catena primaria ribaltata ogni base con la complementare (potrei anche prima sostituire e poi invertire). Se prendo entrambe le stringhe le chiamo **paired strands**.
Aggiungiamo altre definizioni:

- una regione di DNA genomico che ha una certa funzione prende il nome di **locus**
- la sequenza primaria di un locus è chiamata **sequenza genomica**, che di fatto è una sottostringa del genoma di un organismo

Per l'**RNA (*acido ribonucleico*)** si ha una composizione di nucleotidi del tipo:

- il ribosio, uno zucchero, **D**
- un gruppo fosfato, **p**
- una base azotata (Adenina, Citosina, Guanina, Uracile)

simbolo	sigla	nome	simbolo	sigla	nome
A	Ala	Alanina	M	Met	Metionina
C	Cys	Cisteina	N	Asn	Asparagina
D	Asp	Acido aspartico	P	Pro	Prolina
E	Glu	Acido glutammico	Q	GIn	Glutammina
F	Phe	Fenilalanina	R	Arg	Arginina
G	Gly	Glicina	S	Ser	Serina
H	His	Istidina	T	Thr	Treonina
I	Ile	Isoleucina	V	Val	Valina
K	Lys	Lisina	W	Trp	Triptofano
L	Leu	Leucina	Y	Tyr	Tirosina

Tabella 6.1: Tabella con l'elenco degli amminoacidi

Anche per l'RNA si ha la **direzione 5'3'** e si ha un alfabeto $\Sigma = \{a, c, g, u\}$ o $\Sigma = \{A, C, G, U\}$. La vera differenza è che l'RNA è sempre in **singola catena**.
Una **proteina** p una catena di amminoacidi e la sua sequenza primaria è una stringa costruita su un alfabeto di 20 simboli che rappresentano i 20 **amminoacidi** presenti in natura. I **geni** è il "responsabili" della produzione di proteine, che poi "regolano" la vita. Ogni proteina è prodotta da un gene e un gene genera più proteine. **I geni sono il 10% del genoma**, quindi solo una piccola parte. Il gene è quindi un locus del genoma che esprime una proteina. La sequenza primaria del locus di un gene è la sequenza genomica del gene. Ogni gene ha un identificativo detto **HUGO NAME**, con una precisa nomenclatura che è "circa" un acronimo. Un gene può appartenere al genoma di diversi organismi.
Sia il forward strand che il reverse strand contengono geni e gli uni non c'entrano con gli altri (figura 6.2). L'uomo ha circa 20000 geni codificanti (più i cosiddetti **geni non codificanti** che hanno altri scopi), ciascuno che codifica una o più proteine. Ogni cellula contiene l'intero set di geni che regolano la vita dell'organismo, ma nelle varie cellule si **esprime** solo un sotto-set di geni mentre gli altri rimangono **inattivi**, è il **profilo di espressione**. Può variare anche la quantità di proteina espressa. In due cellule posso esprimere lo stesso gene che però porta a diverse proteine.
Si ha quindi:

1. il locus viene trascritto (con anche lo splicing) nell'm-RNA detto trascritto

2. il trascritto porta alla proteina

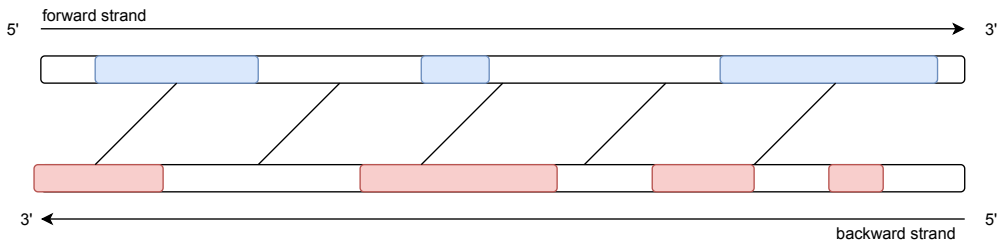


Figura 6.2: Rappresentazione stilizzata dei due strand con in azzurro e rosso, rispettivamente, i geni sul forward strand e sul backward strand

Per determinare la sequenza primaria di DNA e RNA usiamo il sequenziamento, producendo varie read e procedendo poi alla ricostruzione **in-silico**. Posso avere due tipi di output dal sequenziamento:

- **single-end reads**, che è un'unica sequenza
- **paired-end reads** e **mate-pair reads**, dove si parla di coppie di sequenze che sappiamo circa quanto distavano sulla sequenza originale

Per il sequenziamento ci sono stati vari step:

- **first generation**, per il **metodo Sanger** nel 1975. Si hanno:
  - read lunghe fino a 1000bp
  - elevata qualità
  - bassa coverage
  - costi elevati
- **second generation**, con le **Next-Generation-Sequencing (*NGS*) technologies**, dagli inizi degli anni 2000. Si hanno:
  - read corte, da 100bp a 400bp
  - alta coverage
  - bassi costi

  Tra le tecnologie *NGS* abbiamo:
  - Illumina (Solexa) con:
    - HiSeq System

        * Genome analyzer lix
        * **MySeq**, che produce read di 300bp, con il 99.90% di accuratezza, producendo 25 milioni di read per run
    - Ion Torrent-Life technologies con:
        * **Personal Genome Machine (*PGM*)**, che produce read di 200bp, con il 99.00% di accuratezza, producendo 11 milioni di read per run
        * Proton

    Per indicare i dati prodotti diciamo **NGS data**

- **third generation**, con le **Next Next-Generation-Sequencing technologies**, più avanti negli anni 2000. Si hanno:
    - read lunghe fino a 10000bp
    - qualità relativamente bassa

    Tra le tecnologie *Next-Next* abbiamo:

    - Pacific Biosciences con PacBio RS
    - Oxford Nanopore Technologies con:
        * GridION System
        * MinION

6.1 Formato FASTA

Vediamo il formato standard per sequenze primarie nucleotidiche:

- **FASTA** per le sequenze ottenute al metodo Sanger, quindi con pochi errori
- **FASTQ** per le sequenze ottenute con NGS. È un'evoluzione del FASTA in cui si associa alle sequenze dei reads un valore di qualità, uno per base

Il FASTA ha le seguenti caratteristiche:

- è un plain text

- ha la sequenza primaria (ma anche sequenze di amminoacidi) più eventuali informazioni extra a scelta. Si ha:
  - un header introdotto da > con tutte le informazioni extra in una sola riga (tra cui, volendo, cromosoma di partenza, indici di inizio e fine per specificare la porzione, ricordando che le posizioni sono 1-based, la "release", ad esempio tramite la sigla *GRC*, lo strand tramite +1 e -1 etc...)
  - la sequenza genomica separata in righe di 60 o 80 bp (quindi dopo 60 o 80 simboli vado a capo)
- è stato pensato come formato di input del software FASTA per l'allineamento
- ha come estensione `.fa` o `.fasta`

6.1.1 ENSEMBL

ENSEMBL è il **genome browser** attualmente più usato. Un genome browser è un database con associata un'interfaccia di esplorazione. È un progetto partito nel 1999 da **EMBL (*European Molecular Biology Laboratory*)** e dal **Wellcome Trust Sanger Center**. Dal 2017 fa capo a **EMBL-EBI (*European Bioinformatics Institute*)**, sempre di **EMBL**. Si hanno i dati di 250 specie di cordati, tra cui uomo, topo, ratto e zebrafish. Si hanno vari livelli di informazione (ovvero di annotazioni), tra cui; genoma, gene, proteina etc...
I genome browser sono consistenti rispetto a progetti di ricerca e analisi genomiche. SI può accedere, tra le altre cose a:

- la sequenza del genoma di una data specie o dei singoli cromosomi
- le sequenze dei geni annotati su di un genoma e i relativi trascritti espressi
- le variazioni annotate rispetto al genoma di riferimento che caratterizzano i singoli individui

Le coordinate sono sempre rispetto alla catena forward.
ENSEMBL si può raggiungere tramite `www.ensembl.org`.
Si ha anche il formato **EMBL** stesso, pensato originariamente per memorizzare sequenze nucleotidiche in banche dati che non erano db relazionali ed erano senza accesso. Il file poteva essere scaricato tramite un ID e basta (era

poi scopo del biologo e dell'informatico riutilizzarlo). Il file EMBL è pieno di informazioni, con in fondo la sequenza nucleotidica. Banche dati così ci sono ancora, come l'**ENA (*European Necleotide Archive*)**.

6.2 Qualità dei Dati e FASTQ

Valutiamo quindi la **qualità del dato di sequenziamento** e il formato standard **FASTQ**, il formato di output dei sequenziatori di nuova generazione.
Le NGS producono una gran quantità di frammenti, piuttosto corti, per i quali è essenziale conoscere la qualità di ogni base in fase di processamento della read, per eliminare/modificare le parti problematiche. Per ogni base di una read NGS si ha quindi un *indice di qualità*, detto **Phred Quality Score**.
Phred, abbreviato, è stato creato per il *base caller Phred*, che sfrutta il cromatogramma che valutare la qualità.

Definizione 29. *Definiamo **Phred Quality Score** q tramite la formula:*

$$q = -10 \log_{10} p$$

con:

- *base b*
- *p probabilità che b sia errata*

il valore q viene arrotondato all'intero più vicino.
Chiedere una base corretta al 100% comporterebbe $q = \infty$ (avendo $p = 0$) e quindi si considera corretta una base con $q \geq 50$ (che comporterebbe probabilità che sia errata pari a $p = 0.00001$, quindi 0.001%).
Una base con $30 \leq q \leq 50$ è solitamente considerata buona ma dipende dal contesto. Avere $q = 30$ hanno probabilità delle 0.1% di essere errate.

Esempi su slide.
Il formato **FASTQ** è standard per i sequenziatori NGS, che quindi non producono un **FASTA**.
SI ha che è:

- plain text
- è stato sviluppato per associare ad una sequenza il Phred Quality Score

- ha estensione `.fq` o `,fastq`

Si hanno 4 righe, dette record:

- un header con l'identificatore. Tale riga ha "@" come simbolo iniziale. Non si hanno informazioni extra come per il FASTA
- sequenza delle basi della read in una riga
- un header dei phred values. Tale riga inizia con il simbolo "+"
- la sequenza dei valori di qualità, una per ogni base della read

Esempio 15. *Vediamo un esempio:*

```
@HWUSI-EAS522:8:5:662:692#0/1
TATGGAGGCCCAACTTCTTGTATTCACAGGTTCTGC
+HWUSI-EAS522:8:5:662:692#0/1
aaaa`aa`aa`]__`aa`_U[_a`^\\UTWZ`X^QX
```

con:

- *l'header della sequenza un identificatore tipico di illumina*
- *le basi su una riga*
- *l'header, con lo stesso identificatore della sequenza (è opzionale, potrei non averlo), dei phred*
- *la sequenza dei phred values, con una codifica che permette di associare facilmente il valore di qualità. Ogni valore è in corrispondenza di indice con la base di cui rappresenta la qualità (se usassi gli interi avrei magari un intero di due simboli che renderebbe impossibile capire a che base si riferisce)*

I valori interi di qualità vengono quindi convertiti in un certo carattere e ogni sequenziatore ha la sua funzione per convertire l'intero rappresentante la qualità q in char:

$$c = f(q)$$

Ad esempio per Illumina è (con $ASCII(X)$ che converte in char l'intero):

$$c = ASCII(\min\{q, 93\} + 33)$$

Esempi su slide.
Conoscere la qualità delle basi permette di fare **trimming** che, fissata una soglia minima di qualità (decisa di volta in volta), consiste in:

- trovare la più lunga sottostringa della read composta da basi con qualità superiore a questa soglia minima
- sostituire tale sottostringa all'intero read sse questa sottostringa è lunga a sufficienza (lunghezza anch'essa decisa di volta in volta). Qualora la sottostringa sia troppo corta si elimina la read. Su milioni di read sacrificarne qualcuna non è un problema particolare e non solo, si migliorano le analisi successive

6.3 Espressione di un Gene e GTF

Ricordiamo che un gene è un locus che codifica una proteina. La sequenza primaria del locus di DNA del gene è detta **sequenza genomica del gene** (e si ricorda che entrambi gli strand contengono geni, negli esempi si studia comunque il forward essendo già da "sinistra a destra").
I geni si identificano tramite il cosiddetto **HUGO NAME**.
I geni sono formati da **esoni**, regioni codificanti, e **introni**, regioni non codificanti. Il confine tra un esone a sinistra e un introne a destra è detto **sito di slicing al 5'**, detto anche **donor splice site**. Se ho un introne a sinistra e un esone a destra ho un **sito di slicing al 3'**, detto anche **acceptor splice site**. Nell'uomo il 99.24% degli introni iniziano con GT e finiscono con AG e si parla di **introne canonico**. Si hanno anche introni non canonici.
Vediamo gli step di sintesi proteica in modo "operativo":

- si ha la **trascrizione** dove l'intero locus del gene viene copiato in una molecola di RNA (convertendo T in U), detta **pre-mRNA**, lunga quanto il locus del gene
- avviene quindi lo **splicing**, dove si eliminano gli intorni dal pre-mRNA producendo l'**mRNA**, che è una sequenza continua di esoni, di regioni codificanti, detta anche **trascritto**
- il trascritto viene tradotto in proteina. In realtà all'interno del trascritto si ha una parte centrale detta **coding sequence (*CDS*)** che è la parte che da origine alla traduzione in proteina. La CDS inizia sempre con la **tripletta di inizio** AUG e finisce con una tra le seguenti **triplette di fine** UAG, UAA o UGA. La lunghezza della CDS è sempre un multiplo di tre per cui la si può vedere come una sequenza di triplette dette **codoni** (quindi si hanno il **codone di inizio** e **codone di fine** anziché chiamarle triplette). Le due parti a monte e valle della CDS sono dette **5'UTR** (la parte prima) e **3'UTR** (la parte dopo)

- dalla sequenza di codoni si passa alla sequenza di amminoacidi e quindi alla proteina. Questo passaggio è facilmente studiabile tramite il **codice genetico** (dove per comodità l'uracile è spesso segnato segnato con la timina, questa è una cosa standard nelle banche dati in quanto permette confronti diretti con il genoma). Ogni codone corrisponde ad un amminoacido. Il codice genetico è **degenere** quindi più codoni mappano lo stesso amminoacido. Si nota che AUG è la metionina (e solo AUG lo codifica) che è l'amminoacido che inizia ogni proteina. I codoni di fine hanno associato solo un "segnale di stop", non producendo alcun amminoacido, che ferma la traduzione

Il sequenziamento che produce read di RNA, se di tipo NGS, produce read dette **RNA-seq read**, se invece si usa Sanger si chiamano **Expressed Sequence Tag (*EST*)**, che sono ormai in disuso.
Come si era già detto 20000 geni producono centinaia di migliaia di proteine e quindi non si ha una corrispondenza "1:1". Un gene può esprimere una molteplicità di proteine. Questo accade in quanto si ha lo **splicing alternativo**, dove un gene è in grado di combinare i suoi esoni in modo diversi. Si definisce **isoforma** un particolare trascritto che un gene esprime in virtù dello splicing alternativo. Lo splicing alternativo è importante in quanto:

- è **tessuto-specifico** quindi un gene esprime trascritti diversi in tessuti diversi
- dipende dalle condizioni della cellula
- è fortemente legato alle malattie (studiare le differenze tra la sintesi di cellule sane e malate è fondamentale per trovare terapie mirate)

Si descrivono i vari eventi di splicing alternativo:

- l'**exon skipping**, ovvero *salto dell'esone*, dove un esone (o anche più esoni) può essere escluso dal trascritto primario
- l'**alternative acceptor site**, ovvero *sito di taglio alternativo* $3'$, detto anche **3' competing site**, dove una parte del secondo esone può essere considerata non codificante o, alternativamente, una porzione dell'introne adiacente può essere considerata codificante
- l'**alternative donor site**, ovvero *sito di taglio alternativo* $5'$, detto anche **5' competing site**, dove una parte del primo esone viene considerata non codificante o, alternativamente, una porzione di introne adiacente può essere considerata codificante

- i **mutually exclusive exons**, ovvero *esoni mutuamente esclusivi*, dove solo uno di due esoni viene conservato nel trascritto. In due isoforme diverse non ho mai entrambi gli esoni
- l'**intron retention**, ovvero *introne trattenuto*, dove un certo introne viene incluso nel trascritto primario o dove un esone viene considerato in più parti con un introne in mezzo
- **multiple promoters** dove non si considera un prefisso del primissimo esone
- **multiple polyA** dove non si considera un suffisso del'ultimo esone

6.3.1 Il formato GTF

Il **formato GTF (*Gene Transfer Format*)**, `.gtf` è il formato standard che permette di annotare un dato gene su una sequenza genomica di riferimento che contiene il locus del gene (la sequenza può anche essere nello strand opposto a quello dato). Un GTF fornisce per un gene:

- la composizione in esoni delle sue isoforme di splicing (quindi di tutti i trascritti)
- la composizione delle CDS in relazione ai trascritti
- la composizione delle 5/3'UTR in relazione ai trascritti
- la presenza dello start e dello stop codon delle CDS

Da un file GTF si possono ricostruire tutti i trascritti e le coding sequence del gene annotato.
GTF deriva dal formato GFF, spesso si hanno ancora GTF in `.ggf` ed è formato da 9 campi separati da tabulazioni.
Il trascritto viene annotato "all'indietro" sul locus del gene.
Ogni record del GTF fornisce una **feature**, ovvero una regione continua della sequenza genomica, che rappresenta uno tra:

- un esone, tramite `exon`. Ad un trascritto corrispondono tante feature di tipo `exon` sulla genomica di riferimento quanti sono gli esoni che lo compongono (e per ogni feature di tipo `exon` ho un record)

- una porzione di CDS, tramite `CDS`. Una porzione in quanto la CDS completa sul locus è "spezzata" dagli introni e quindi mi servono più feature per rappresentarla. Ipotizzando che una CDS sia un pezzo di un esone e un intero secondo esone avrò due feature `CDS` con gli indici di queste due parti (se corrisponde ad un esone avrò anche una feature con gli stessi indici di tipo `exon`). Ad una CDS corrispondono tante feature quanti gli esoni che copre sul trascritto
- uno start codon, con `start_codon`. Solitamente è un unico record
- uno stop codon, con `stop_codon`. Solitamente è un unico record
- una porzione 5'UTR, con `5UTR`. A un 5'UTR corrispondono tante feature sulla sequenza di riferimento quanti sono gli esoni che copre sul trascritto
- una porzione 3'UTR, con `3UTR`. A un 3'UTR corrispondono tante feature sulla sequenza di riferimento quanti sono gli esoni che copre sul trascritto

Il file GTF ha solo gli indici di partenza e fine dei vari elementi e quindi mi serve associato un file FASTA con la genomica di riferimento.
Un GTF può avere più geni annotati.
Ogni record ha 9 campi:

1. l'**identificatore della genomica di riferimento** (presa su uno dei due strand) che copre il locus del gene
2. la **sorgente** che ha prodotto l'annotazione (ad esempio un software se prodotta "in silico")
3. il **nome della feature** (*exon, CDS, 5UTR, 3UTR, start_codon, stop_codon*)
4. la **posizione di inizio** della feature sulla genomica di riferimento (1-based, non si parte da 0 ma 1 con gli indici)
5. la **posizione di fine** della feature sulla genomica di riferimento (1-based)
6. lo **score** della feature

7. lo **strand** ("+", "-"). In un file GTF possono coesistere geni che si esprimo su strand opposti e che vengono annotati su un'unica genomica di riferimento. Il GTF permette di annotare un gene che contiene il sup locus sulla catena opposto e questo campo mi permette di segnalare la cosa. Se ho "-" so che il gene è sullo strand opposto rispetto a quello della genomica di riferimento (con "+" è sullo stesso). Con "-" e features del gene avranno coordinate decrescenti passando da sinistra a destra sul trascritto mentre con "+" avranno coordinate crescenti passando da sinistra a destra sul trascritto. Ne segue che:

    - per ottenere la sequenza di una feature con strand "+" basta estrarre la sottostringa della genomica di riferimento che corrisponde alla feature
    - per ottenere la sequenza di una feature con strand "-" bisogna estrarre la sottostringa della genomica di riferimento che corrisponde alla feature e fare *reverse&complement* della sottostringa estratta

    Ovviamente tutte le features annotate per un dato gene avranno sicuramente lo stesso campo strand

8. il **frame** (0, 1, 2) solo per feature `CDS`, `start_codon` e `stop_codon`. Nel dettaglio si ha che, per la feature `CDS`:

    - 0, se la prima base della feature è la prima base di un codone
    - 1, se la prima base della feature è la terza base di un codone
    - 2, se la prima base della feature è la seconda base di un codone

    Possiamo dire che, dato L lunghezza totale delle feature `CDS` che vengono prima di quella in analisi, si ha:

    $$(3 - L \bmod 3) \bmod 3$$

    Per le altre feature trovo un "."

9. il campo degli **attributi della feature** come coppie chiave-valore: *<attribute_name1> <value1>; <attribute_name2> <value2>; ...*
   Gli attributi *gene_id* e *transcript_id* sono attributi obbligatori

Capitolo 7

Strutture di Indicizzazione

Le strutture di indicizzazione permettono varie cose nello studio del genoma, dall'assembly, all'individuazione di match, MEMs etc..., anche in ambito pan-genomica. Tra le strutture principali abbiamo:

- suffix tree
- suffix array
- LCP
- BWT

e le studieremo sia in ottica genomica che pan-genomica.
Con $T[i; j]$ abbiamo la sottostringa da i a j (incluso) e con $T[i :]$ il suffisso che inizia in i.

7.1 Caso Genomico

7.1.1 Suffix Tree

È una delle prime strutture create, è molto efficiente e permette di risolvere tanti problemi in tempo lineare ma ha lo svantaggio dell'occupazione di memoria, memorizzando in modo ridondante. Non va quindi bene applicato direttamente a genomi e dati NGS. Volendo può essere simulato tramite SA, LCP e altre strutture.

Definizione 30. *Dato un testo T di n caratteri un suffix tree è un albero radicato con:*

- *n foglie etichettate da 1 a n*

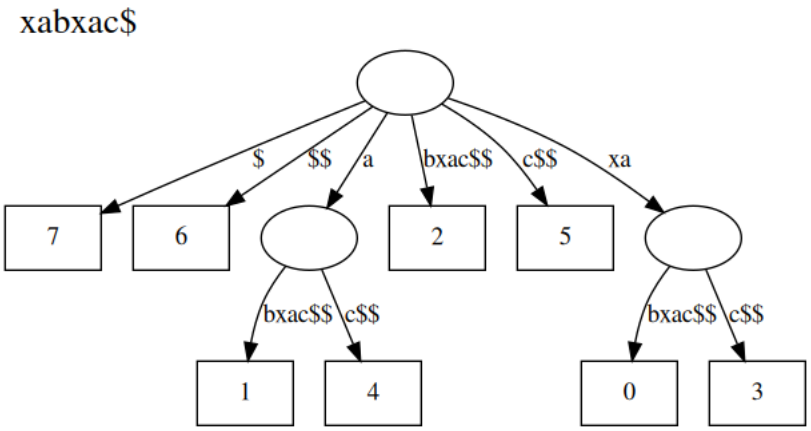


Figura 7.1: Esempio di suffix tree per la stringa "xabxac"

- *ogni nodo interno ha almeno 2 figli*
- *ogni arco è etichettato da una sottostringa di T*
- *non esistono due archi uscenti da un nodo con lo stesso carattere iniziale per l'etichetta*
- *la concatenazione delle etichette dalla radice alla foglia i è* $T[i:]$

Anche ogni etichetta di arco che porta ad una foglia è un suffisso*, da qui il discorso delle ripetizioni e della ridondanza di informazione.*

Teorema 5. *Il ST di T esiste sse nessun suffisso occorre come prefisso di qualche altro suffisso, in quanto avrei un nodo interno con figli che iniziano con lo stesso carattere come primo carattere delle sottostringhe e per evitare la cosa otterrei nodi con singoli figli.*

Per risolvere il problema si usano testi $ terminati, con $ che è il carattere minore di tutti lessicograficamente.
Si hanno tre algoritmi lineari per la costruzione, in $O(n)$:

- **algoritmo di Weiner**
- **algoritmo di McCreight's**, efficiente in memoria
- **algoritmo di Hukkonen**, online

Definizione 31. *Dato un nodo w definisco la* ***path label*** L_w *come la concatenazione delle etichette dalla root a w. Se w è una foglia i allora la path label è il suffisso* $T[i:]$

Definizione 32. *Dato un nodo w definisco la* ***string depth*** *come la lunghezza della path label L_w*

Definizione 33. *Definisco* ***dummy node*** *come un falso nodo che separa in due l'etichetta di un arco (w, w') in due stringhe l_1 e l_2. La sua path label è $L_w L_1$. È un nodo a metà di un arco.*

Teorema 6. *Due nodi diversi non possono avere la stessa path label.*

Teorema 7. *Dato un nodo interno w, le k foglie del sottoalbero radicato in w danno le posizioni di inizio dei k suffissi che condividono L_w come prefisso.*

Teorema 8. *Dato un dummy node d che separa (w, w') le k foglie del sottoalbero radicato in w danno le posizioni di inizio dei k suffissi che condividono $L_w L_1$, ovvero la path label di d, come prefisso.*

Per il pattern matching, con pattern P lungo m e testo T lungo n, con ST si ha una soluzione in tempo:

$$O(n + m + k)$$

avendo:

- costruzione di ST in $O(n)$
- ricerca delle k occorrenze in $O(m + k)$, cercando in $O(m)$ ò'unico nodo w che ha path label pari a P e visito in $O(k)$ il sottoalbero radicato in w elencando le k foglie

In quanto si sa ha che:

- se P occorre k volte allora P è prefisso di k suffissi di T
- se P è prefisso di k suffissi di T allora P è la path label di un nodo w, anche dummy, che è unico
- le k foglie del sottoalbero radicato in w, che può essere dummy, danno le posizioni di inizio delle k occorrenze di P su T

Se P non occorre non trovo il w e mi fermo.
Avendo alfabeto Σ finito la ricerca dell'arco uscente da un certo nodo la cui etichetta inizia con un certo carattere è in tempo costante con la giusta implementazione.
Su slide esempio di pattern matching.

7.2 Suffix Array

Riprendere teoria da appunti di Teoria della Computazione.
Il SA è nato come struttura ausiliaria di un ST. Si ricorda che la costruzione è in $O(n)$ e il pattern matching è in $O(m \log n)$.
Si usa spesso il suffix array inverso SA^{-1}, è definito in modo tale che in posizione j ho l'indice i sse $T[j:]$ è l'i-esimo suffisso nell'ordinamento lessicografico dei suffissi di T:

$$SA^{-1}[j] = i \iff SA[i] = j$$

In tempo lineare posso costruire il SA dal ST, visitando in profondità, in $O(n)$, il ST seguendo l'ordine lessicografico delle etichette degli archi uscenti da un dato nodo, scelgo sempre l'arco con etichetta "minore". Ogni volta che arrivo ad una voglia aggiungo l'etichetta della foglia al SA, in ordine. Ne segue che il primissimo è ovviamente sempre \$ e quindi il primo indice del SA è ovviamente il suffisso nullo.
Esempio su slide.

7.2.1 Longest Common Prefix

Passiamo ora all'**LCP** è stato introdotto come struttura ausiliaria per il SA.

Definizione 34. *Dato un testo T dollaro terminato, l'LCP è un array lungo n tale che $LCP[i] = l$ sse l è la lunghezza del più lungo prefisso comune tra $T[S[i]:]$ e $T[S[i-1]:]$. In altri termini è la lunghezza del più lungo prefisso comune dei suffissi corrispondenti a $S[i]$ e $S[i-1]$.*
SI assume $LCP[1] = -1$

L'algoritmo di Kasai lo costruisce a partire da SA in tempo lineare.
SA e LCP permettono il pattern matching in $O(m + \log n)$, migliorando il $O(m \log n)$ del solo SA. Questo è permesso dal fatto che con LCP il confronto del pattern non torna mai indietro.
Quindi $lcp(i, j)$ è la **LCP-funcion** è computa lunghezza del più lungo prefisso comune tra $T[S[i]:]$ e $T[S[j]:]$. Assumendo $i < j$ si ha che $lcp(i, j)$ è il minimo tra i valori di LCP compresi tra $LCP[i+1]$ e $LCP[j]$ quindi, usando la **range minimum query** (cerco il valore minimo tra $LCP[i+1]$ e $LCP[j]$):

$$lcp(i, j) = RMQ(LCP, i + 1, j)$$

Esempio su slide.

Definizione 35. *Definiamo* ***lcp-interval*** *di valore l come l'intervallo di posizioni $[i, j]$ tale che:*

- $i < j$
- $LCP[i] < l$
- $LCP[j + 1] < l$
- $LCP[k] \geq l, i + 1 \leq j \leq j$
- $LCP[j] = l$ *per almeno un* $k \in [i + 1, j]$*, tale* k *è detto* l*-index*

L'lcp-interval $[i, j]$ è anche detto l-interval, detto anche l-$[i, j]$.
Il valore l di un lcp-interval $[i, j]$ è uguale a:

$$RMQ(LCP, i + 1, j)$$

Il valore l è la lunghezza del più lungo prefisso comune tra i suffissi di T che iniziano in posizione $S[i], S[i + 1], \ldots, S[j]$, che viene denotato con ω.
Dato $l = 0$ si ha che:

$$0\text{-}[1, n]$$

è lo 0-interval che ha come ω il prefisso nullo.

Esempio su slide.

Definizione 36. *Definisco l'-interval $[g, d]$ con prefisso ω' come* ***embdedded*** *in l-interval $[i, j]$, con prefisso ω sse;*

$$i \leq g < d \leq j$$

Quindi $[i, j]$ racchiude $[g, d]$.
In tal caso si ha:

$$l' > l$$

e ω è prefisso di ω'.

Esempio su slide.

Definizione 37. *Si dice che l'-$[g, d]$ è figlio di l-$[i, j]$ sse non esiste un altro lcp-interval che racchiude $[g, d]$ e che a sua volta è racchiuso in $[i, j]$.*
Per trovare i figli di l-$[i, j]$:

- *identifico gli l-indici $i_1, \ldots i_k$, dal più piccolo al più grande*

- *il primo figlio è* $[i, i_1 - 1]$*, il secondo* $[i, i_2 - 1]$ *e etc... fino a* $[i_k, j]$*, scartando però quelli di ampiezza 1*
- *il valore* l *di ogni figlio* $[g, d]$ *è dato da:*

$$RMQ(LCP, g + 1, d)$$

Definizione 38. *Definisco* ***lcp-interval tree*** *come un albero radicato* (V, A) *tale che:*

- V *è l'insieme degli lcp-interval*
- A *è l'insieme di tutte le relazioni parent-child*
- *la radice corrisponde a* 0-$[1, n]$

Esempio su slide.
Grazie a tutte queste nozioni possiamo legare SA e ST. Se prendo un ST e nascondo foglie e archi in esse entranti ottengo una struttura uguale a quella dell'lcp-interval tree, stessa topologia. Esiste un nodo l-$[i, j]$ con prefisso ω nell'lcp-interval tree sse si ha un nodo nel ST con path label ω.
Con SA, SA inverso, LCP posso simulare un attraversamento del ST, anche senza lcp-interval tree esplicito tramite *enhanced SA*, senza mi serve anche il lcp-interval tree.

7.2.2 BWT e FM-Index

Riprendere teoria da appunti di Teoria della Computazione.
Posso avere Q-intervalli che sono lcp-interval ma non tutti, visto che scarto gli lcp-interval di ampiezza 1. Quando si ha la corrispondenza ho $Q = \omega$.
Dato il Q-intervallo $[b, e]$ (quindi con intervallo chiuso, non come a teoria della computazione) ho la backward extension per σ: ho che:

$$b' = C(\sigma) + Occ(\sigma, b) + 1$$

$$e' = C(\sigma) + Occ(\sigma, e + 1)$$

Se $e' < b'$ allora il σQ-interval non esiste.
Se ho un Q-intervallo $[b, e]$ che sulla BWT contiene \$ allora esso caratterizza una stringa Q che è prefisso di T.

7.3 Caso Pan-Genomico