Riassunto di biologia computazionale

# Allineamento delle sequenze di DNA

Per allineare le sequenze di DNA, si effettua una ricerca all’interno della banca dati, qui vengono effettuati dei confronti in base ai concetti di funzione, evoluzione e annotazione.

Ma come viene effettuata questa ricerca? ebbene, essa si basa sul concetto di edit distance, un tipo di distanza in cui si contano tutte le modifiche da effettuare per convertire una stringa A in una stringa B.

Nel caso delle sequenze di DNA, l’edit distance permette di allinearle.

Quella che interessa tra tutte è la sequenza minima, cioè quella col minor numero di modifiche da effettuare.

Precisamente, l’edit distance si basa su tre operazioni:

* la sostituzione di un carattere con un altro;
* la cancellazione di un carattere;
* l’inserimento di un carattere.

Due sequenze di DNA possono avere più edit distance associate, ciò succede perché è possibile ottenere una sequenza a partire da un altra in più modi. Oltre a ciò, bisogna inoltre tenere conto di eventuale conoscenza pregressa.

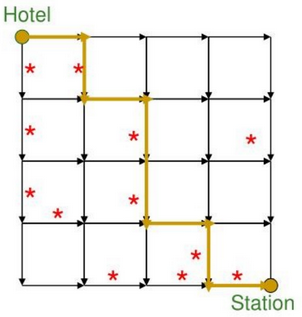
Un modo per calcolare l’edit distance è la dot matrix, una matrice in cui si indicano per ogni carattere i punti che non necessitano di modifiche. Di questa matrice interessa la diagonale, essa infatti indica tutti i punti che non cambiano tra le due sequenze.

## Il problema del resto

Il problema del resto indica che, data una certa quantità di denaro e un certo numero di monete dal valore prestabilito, si vuole cambiare la somma minimizzando il numero di monete.

Questo problema si risolve con la programmazione dinamica utilizzando il seguente algoritmo:

| RestProblem(amount,moneys[])  best[0]=0;  for(int a to amount)  best[a]=infinite  foreach(money m in moneys)  if(a>=m.value)  if(best[a -m.value] < best[m])  best[a]=best[a - m.value] + 1  return best[a] |
| --- |

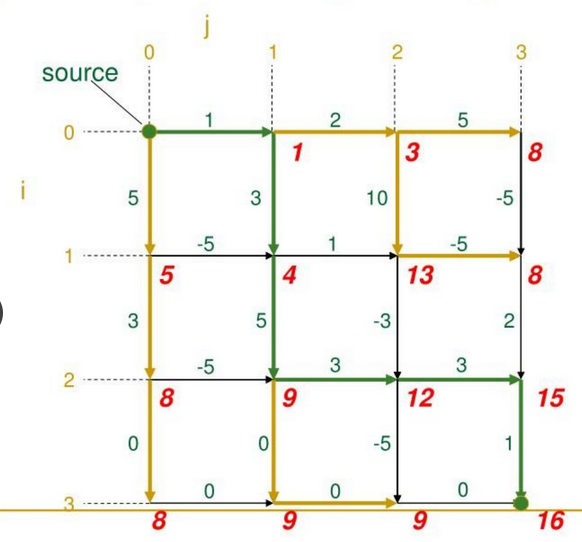


## Il problema del turista di Manhattan

Il problema del turista di Manhattan è un problema in cui, data una matrice, bisogna andare dall’hotel alla stazione (rispettivamente i vertici in alto a sinistra e in basso a destra) visitando più attrazioni possibili, queste sono contrassegnate da un \* nella matrice.

Il turista non ha molto tempo, di conseguenza può viaggiare solamente verso est e verso sud.

Qual è il percorso migliore?

Per risolvere questo problema, si formula il problema come segue:

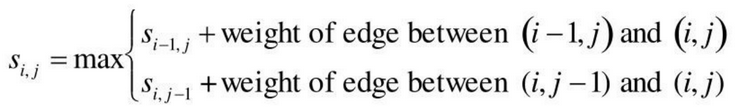
* l’input del problema è una matrice pesata G con due vertici distinti, cioè il punto di partenza e quello di arrivo;
* il goal è il percorso più lungo nella matrice;
* L’output è il percorso più lungo in G tra partenza e arrivo.

Come si risolve il problema? Attraverso la programmazione dinamica! Come funziona? In pratica calcola il percorso ottimale calcolando il minimo/massimo tra tutti i possibili percorsi, considerando l’esempio:

* calcolare il percorso ottimale tenendo conto dell’arco 0-1 e delle precedenti scelte;
* calcolare il percorso ottimale tenendo conto dell’arco 1-0 e delle precedenti scelte;
* prendere tra i due precedenti quello di peso minimo/massimo in base al caso;
* se nel frattempo si riesce a trovare un cammino migliore, si sovrascrive il risultato.

## Algoritmo

Tutto ciò si può riassumere col seguente algoritmo:



Con una matrice di n righe e m colonne, la complessità è O(n\*m).

Come si calcolano le distanze? Si attraversa tutta la matrice, per farlo si utilizzano apposite strategie, le più comuni sono:

* colonna per colonna;
* riga per riga;
* andando in diagonale, il cosiddetto dove tailing.

All’inizio c’è un passo di inizializzazione in cui si settano la cella i-0 e la cella 0-j rispettivamente a w\*i e w\*j, dove w è la gap penalty.

Dopodichè si passa all’algoritmo verso e proprio: per ogni cella i-j si calcolano i seguenti valori:

* valore precedente sulla diagonale più la score di match o mismatch;
* valore precedente sulla riga più w;
* valore precedente sulla colonna più w.

Di tutti questi, si prende il valore massimo.

Come si ricostruisce la soluzione ottimale? Ebbene, data la matrice compilata, si parte dal punto in basso a destra e si procede come segue:

* se i caratteri sulla riga e sulla colonna sono uguali, si procede in diagonale andando nella cella in alto a sinistra immediatamente più vicina;
* se invece i caratteri sono diversi, si va nella cella immediatamente a sinistra.

# Cos’è il DNA? e l’RNA invece?

Il DNA, o acido desossiribonucleico , si può rappresentare informaticamente attraverso una stringa, su di essa si utilizza un alfabeto di quattro caratteri, le cosiddette basi azotate (adenina, timina, citosina e guanina, rispettivamente A, T, C e infine G).

L’RNA è l’acido ribonucleico, esso si differenzia dal DNA per via della presenza di due molecole OH in quest’ultima.

Il DNA è a doppia elica, ogni filamento è sempre accoppiato con un altro secondo il seguente ragionamento:

* la base azotata A (adenina), si accoppia con T (timina);
* la base azotata C (citosina), si accoppia con G (guanina).

Il DNA umano è formato da circa tre miliardi di paia di basi, tutte queste si possono scrivere in un file, occupando 3 GB di memoria.

La percentuale di DNA che codifica i geni è solo il 5%, il 3% invece è la regione codificante, una sottoregione dei geni.

La regione di DNA che si trova tra otto geni è detta regione intergenica.

Il processo di trascrizione del DNA permette la copia dei filamenti a partire da un Transcription Start Site (TSS) e finisce in un altro punto ben definito.

Il gene in questo processo viene quindi trascritto in una molecola di RNA, questo viene modificato attraverso un processo detto maturazione e diventa mRNA.

Dopodichè, vengono tradotti i nucleotidi in amminoacidi, questo processo avviene convertendone tre alla volta.

### Esempio

Si vuole allineare la sequenza ATCCTAT con tutto il genoma.

Prima di tutto bisogna confrontare le due sequenza con differenze di ordine di grandezza. Oltre a ciò occorre tenere presente di eventuali interruzioni e anche del fatto che ogni genoma è differente.

La sequenza si può allineare non solo con la sequenza genomica ma anche con l’mRNA presente nelle banche dati.

## La genomica comparata

La genomica comparata parte da un presupposto: se qualcosa serve all’evoluzione, essa viene mantenuta. Prendendo come esempio il DNA, solo i geni che servono vengono mantenuti e, di conseguenza, le relative sequenze al loro interno.

Se la stringa non è presente nella banca dati o le informazioni trovate non sono sufficienti, è possibile utilizzare anche banche dati di altri organismi. La problematica da gestire riguarda le differenze tra sequenze, infatti queste risultano sempre maggiori, serviranno quindi altrettanti algoritmi per gestirli.

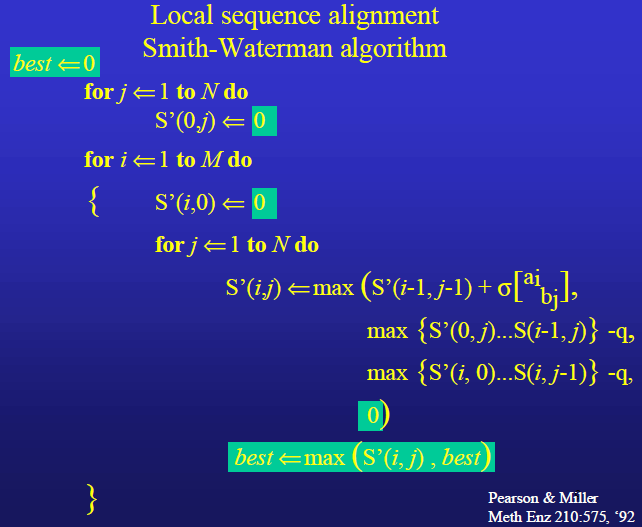
Nel caso delle proteine, queste sono formate da 20 amminoacidi, queste ultime sono inoltre di diversi tipi e quindi il confronto risulta complicato e può causare dei mismatch.

Ogni mismatch ha un suo punteggio e un suo valore, questi possono essere considerati più o meno sulla basi di matrici di sostituzione che valutano gli amminoacidi.

## Allineamento locale

L’allineamento locale serve a indicare quanto sono simili due sequenze di DNA, precisamente è necessario per:

* l’organizzazione modulare di geni e proteine;
* la ripetizione;
* le sequenze diverse in modo da trattenere la similarità.

L’obiettivo è quello di trovare la regione o sottosequenza comune più grande tra due sequenze, da ciò produciamo un gap indicante quanto bisogna shiftare una sequenza rispetta all’altra.

Come procediamo? Un metodo per allineare due sequenze è l’algoritmo di Smith-Waterman in cui si prendono in considerazione le seguenti misure:

* indica lo score di allineamento tra i residui a e b;
* q è la penalità di gap;
* S’(i,j) è lo score ottimale per allineare i residui i e j;
* best è lo score più alto della matrice S.

L’algoritmo è una variazione della massima sottosequenza comune, quindi viene utilizzata una matrice in cui segniamo i match, precisamente il valore (i,j) è il massimo tra:

* il precedente sulla riga meno il gap;
* il precedente sulla colonna meno il gap
* il precedente in diagonale più lo score , quest’ultimo è 1 quando a e b sono uguali e -1 e altrimenti;
* il valore zero per rendere i valori non negativi.

Lo score per l’allineamento dovrebbe essere negativo in modo da identificare l'allineamento. Esistono comunque allineamenti locali ottimi che permettono di ottenere allineamenti più lunghi su regioni non correlate. In generale lo score di allineamento dipende dal gap, dalla matrice e dal modello di gap score.

Il gap score può infatti essere una relazione costante sulla lunghezza del gap:

dove g è il numero di residui gappati;

Oppure una funzione affine:

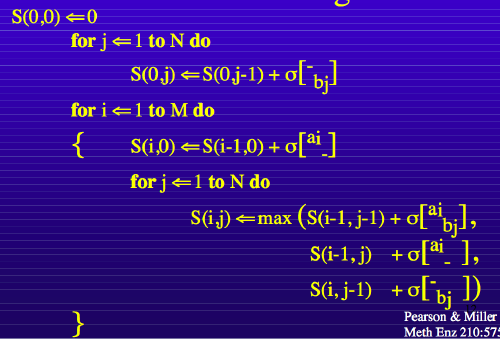
Se l’allineamento avviene su sequenze non correlate, la somma dei singoli score dovrebbe risultare negativa. Possiamo identificare facilmente nella matrice questi casi, tuttavia non abbiamo un modo analitico per trovarle con un gap score che soddisfa gli allineamenti.

### Matrice di sostituzione unaria

La matrice di sostituzione unaria utilizza due score, uno per i match e un’altro per i mismatch.

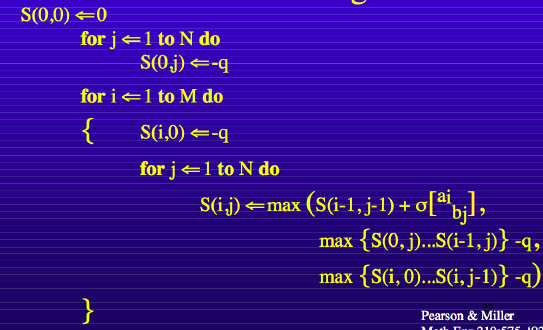
In una sequenza di proteine ci sono venti tipi di residui con una complessa relazione con la dimensione, la carica, il codice genetico e la chimica. La matrice unitaria è overperformata da matrici, ognuna di queste può avere score differenti. Queste matrici vengono calcolate calcolando lo score tra le differenti relazioni tra sequenze in base alle feature o alle occorrenze sostituite nell’ordinamento corretto.

# Allineamento globale

Per l’allineamento globale abbiamo l’algoritmo di Needleman-Wunsch, un metodo derivato dalla massima sottosequenza comune in cui:

* in caso di match tra geni, si incrementa di 2 il valore della soluzione;
* in caso di mismatch, non si aggiunge niente.

Come funziona? Date due stringhe r e s:

* la prima fase inizializza una matrice ponendo tutte le celle a zero, le righe indicano i caratteri della stringa r, le colonne invece quelli di s;
* Si scorre la matrice per individuare i match, in tal caso si scrive 2 nella rispettiva cella;
* A partire dalla cella in alto a sinistra, si calcola lo score della cella come il massimo delle celle precedenti più il match/mismatch.

Una variante di questo algoritmo calcola utilizza la gap penalty nel calcolo della cella, rendendolo più simile all’allineamento locale.

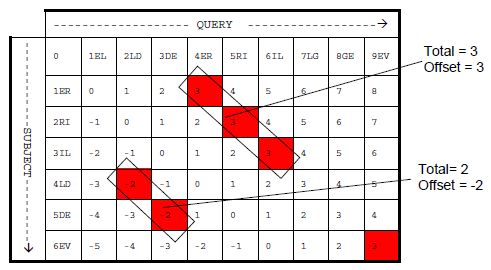
## Metodi euristici

L’applicazione dei metodi di programmazione dinamica ha un costo che dipende da amminoacido ad amminoacido, ciò accade perchè i match non sono tutti uguali a causa delle proprietà dell’amminoacido stesso.

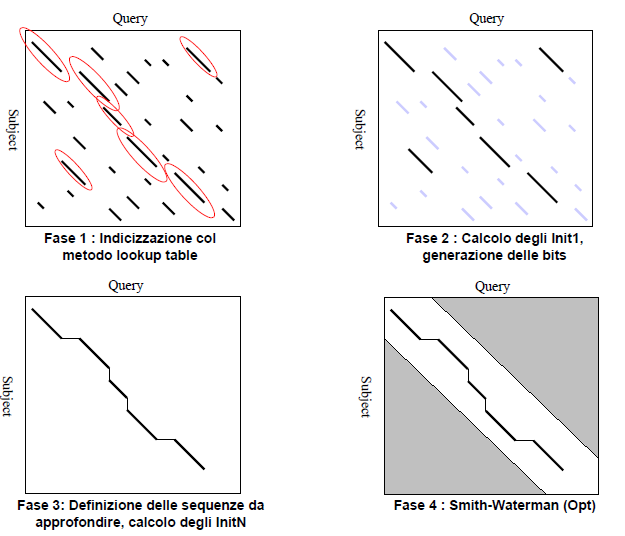
Considerando la foto a destra, la sostituzione di una proteina basica con una acida potrebbe costare di più rispetto a una dello stesso tipo.

Proprio per questi motivi esistono i metodi euristici, dei metodi per cui si riduce lo spazio di ricerca attraverso veloci metodi approssimativi. Questi metodi selezionano le sequenze che sono probabilmente più simili a una data query. Ciò permette quindi di allineare solamente le sequenze selezionate e solo su una determinata porzione.

## FASTA

L’algoritmo FASTA utilizza una lookup table e una strategia di indicizzazione delle parole, precisamente la proteina in input viene divisa in parole lunghe k. Maggiore è k, più la ricerca sarà rapida e meno accurata.

Si procede ora alla creazione della tabella degli offset: qui utilizziamo una tabella in cui le colonne sono le parole della query mentre le righe sono le parole di un subject della banca dati, in ogni cella andiamo a scrivere l’offset, calcolato come segue:

Dopodichè memorizziamo le diagonali che presentano il miglior punteggio, come le troviamo? Lo troviamo attraverso somme di coppie.

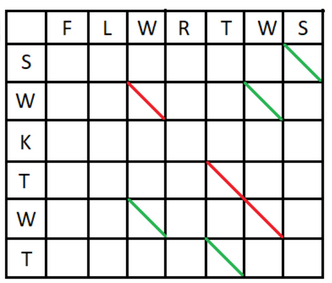
Le migliori diagonali identificano le best inital regions (BIR), i cui punteggi vengono definiti init1.

Per ogni subject, abbiamo uno score che definisce la similarità media rispetto alla query. Quindi dato un’init1 minimo, scegliamo in modo euristico delle sequenze per effettuare analisi più approfondite.

Preso l’init1, la fase successiva è quella di ricongiungere le BIR, come si fa? Si possono utilizzare diversi criteri:

* si escludono eventuali overlap tra regioni;
* si calcola un punteggio e si prendono solamente quelli superiori a una data soglia;
* si calcolano dei punteggi di penalizzazione per i gap.

Da ciò si ottiene un nuovo score detto InitN e consiste in un segmento spezzato composto dalla maggior parte delle BIR.

Infine si allinea tutta la regione del subject con l’algoritmo di SmithWaterman, inoltre si tiene una finestra intorno alla diagonale principale abbastanza stretta. Da qui si ottiene lo score di allineamento definitivo detto opt.

Per fare un esempio, supponendo di avere le sequenze FLWRTWS e SWKTWT, il calcolo della diagonale propone un +1 per T e le due W, quindi si shifta di 1 la seconda sequenza:

FL**W**R**TW**S

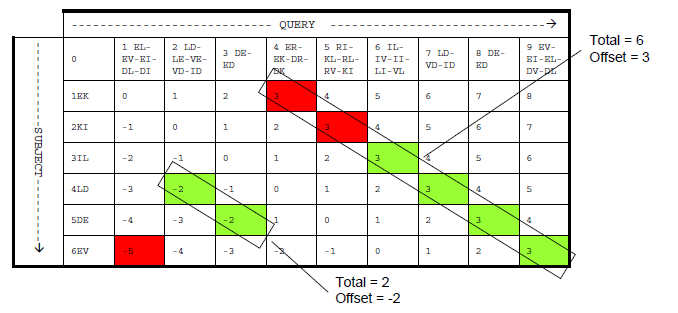
S**W**K**TW**T

Si può riassumere tutto nella tabella a destra, qui infatti sono presenti tutte le possibili diagonali tra le due sequenze, in particolare quelle rosse indicano le diagonali migliori.

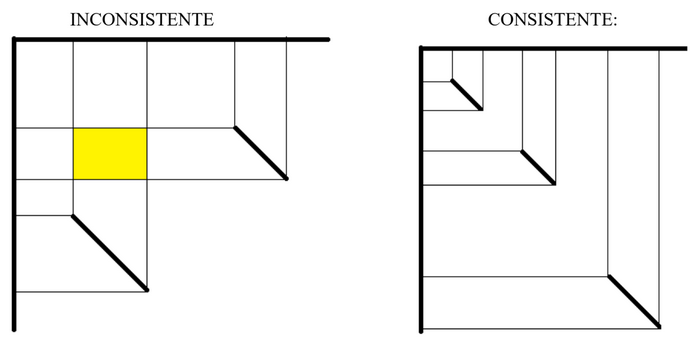
Il problema di questo algoritmo riguarda l’aspetto euristico, infatti si rischia di perdere anche buoni match in questo modo.

## BLAST

BLAST è un algoritmo euristico che, come FASTA, è basato sull’indicizzazione, qui però c’è della complessità.

L’algoritmo inizio creando un elenco di parole di lunghezza W a partire dalla query. Per ogni parola vengono create tutte le possibili w-mers, cioè parole di lunghezza W che danno uno score maggiore di un dato threshold quando allineate. Lo score è dato da una matrice di sostituzione decisa in precedenza.

Anche in questo caso si usa una tabella query-subject, in essa si segnano il numero di hit (match positivi) anche se le due parole non sono identiche. Gli indici fanno riferimento alle parole vere della query, non a dei sinonimi. Quello che si ottiene è una lista di proteine con cui trovare la corrispondenza con i frammenti della query.

Una volta trovate le hit, BLAST le congiunge, non sono tuttavia ammessi dei gap. L’allungamento infatti si ferma quando lo score va sotto una data soglia S. Ciò che abbiamo individuato prende il nome di High-scoring segment pair (HSP). L’HSP può ancora allungarsi quando la soglia scende solo su pochi residui per poi risalire, quanti di questi residui può sottortare? Il numero è dato dal parametro X, esso indica la perdita di score massima che si può tollerare.

Le regioni trovate sono divisibili in:

* Consistenti quando è possibile allinearne in maniera consecutiva senza intersezioni;
* Inconsistenti se invece ci sono sovrapposizioni, quindi due regioni uguali non possono coesistere nello stesso allineamento.

# Hidden Markov Model

BLAST e FASTA sono algoritmi che permettono di allineare sequenze senza troppa complessità, ma in caso di coppie poco rappresentative nel genoma umano? Nel caso della coppia citosina-guanina, questa va contro la metilazione, quindi la base C ha la tendenza a diventare methyl-C proprio per questo motivo. Methyl-C a sua volta ha una buona probabilità di mutare in T.

Le regioni citosina-guanina, dette anche CpG, indicano l’inizio di un gene, come si possono scoprire? Utilizzando gli Hidden Markov Models!

## Problema del casinò a scommessa equa

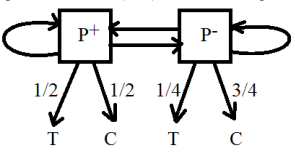
Il problema del casinò a scommessa equa consiste nel lancio di una moneta, tuttavia non si sa se questa è stata truccata o meno. Dato che la moneta vera viene scambiata con quella falsa ogni tanto, bisogna capire quando avviene questo scambio.

Si definiscono le seguenti variabili:

* P+, la probabilità di avere la moneta vera;
* P-, la probabilità di avere la moneta truccata;
* La probabilità che esca testa (T), rispettivamente 0.5 per la moneta vera e 0.25 per quella truccata;
* La probabilità che esca croce (C), cioè 0.5 per la moneta vera e 0.75 per quella truccata;
* un numero n di lanci;
* un numero k indicante quante volte è uscito testa.

Data la sequenza TCCTTTCCCTT, con quale probabilità si ottiene?

* Con la moneta vera, la probabilità è P+ \* (½)^n;
* Con la moneta truccata invece, la probabilità risulta essere P- \* (¼)^k \* (¾)^n-k.

Qui però non si è sicuri del momento in cui avviene il cambio, quindi come si procede? Si cambia prospettiva!

Data una sequenza di monete (corrispondenti a degli stati), si massimizza la probabilità che questa sequenza derivi da un dato stato.

Quindi, se nello stato attuale al moneta è vera (P+), con quale probabilità la moneta rimane tale o cambia nello stato successivo? Per fare ciò, dato che ogni stato emette un simbolo con una certa probabilità, si può ricavare la sequenza di stati più probabile.

Quali sono i problemi?

Bisogna innanzitutto disegnare il modello, da qui ricavare le probabilità corrette e, attraverso un buon algoritmo di deconvoluzione, dare un significato al risultato.

## JMB

JMB è un hidden markov model particolare che permette di generare sequenza di amminoacidi appartenenti a una data famiglia proteica.

Un esempio di JMB sta nella foto a destra, qui i quadrati indicano gli stati residuo, cioè quelli in cui viene prodotto un simbolo in base alla propria tabella di probabilità.

I rombi invece sono stati inserzione mentre i cerchi gli stati delezione.

Le varie frecce indicano le possibili transizioni, ognuna di esse ha una propria probabilità.

Le transizioni verso un quadrato o un cerchio fanno muove in avanti il modello di una posizione, quelle verso un rombo invece lo fanno rimanere nella stessa posizione.

In questo modello, il primo e l’ultimo stato servono per iniziare/terminare il modella senza emettere nessun simbolo.

In generale, gli hidden markov model permettono di ottenere la sequenza più probabile, l’allineamento di solito delle proteine e la ricerca di una sequenza somigliante in una banca dati.

## Come si trova un pattern di sequenza?

Supponendo di aver già trovato un’ancora, si possono usare tre approcci:

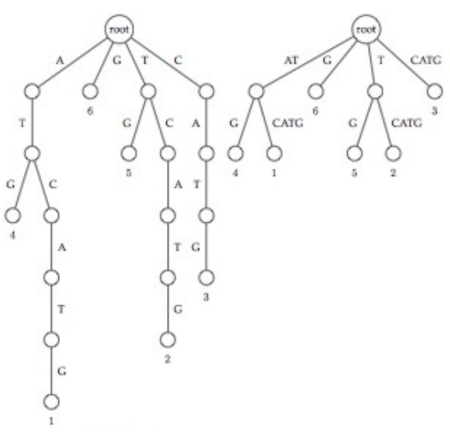
* scandire l’intera sequenza, ciò però ha una complessità di O(N\*M), la quale diventa O(K\*N\*M) se il numero di pattern è K.
* si usano i keywork tree, degli alberi che indicizzano i pattern e ci fanno scorrere sopra il testo. Quello che si ottiene è un albero in cui, dalla radice a ogni nodi, non possono esserci le stesse lettere negli archi uscenti.

Ogni percorso dalla radice alle foglie rappresenta un pattern, come si trovano? A partire da una lettera, se questa viene trovata, si segue quel ramo finché si può, altrimenti si ferma prima e riparte usando un’altra lettera;

* si usano gli alberi di suffissi, un’evoluzione dei keyword tree in cui vengono collassati tutti i nodi che presentano un solo arco uscente, di conseguenza un arco può contenere più caratteri. Tutto il resto rimane uguale ai keyword tree.

# Alberi di keyword e di suffissi

Data una stringa di geni lunga L, ciò che bisogna fare è identificare i geni e le regioni regolatorie (cioè le funzioni dei geni). Geni e regioni regolatorie vengono conservati se questi effettivamente servono.

La comparazione tra geni si basa su cose già scoperte confrontando proteine, strutture di geni e regioni regolatorie, quella tra genomi invece permette di confrontare sottoinsiemi di genomi per cercare di estrapolare più informazione!

L’idea è quindi calcolare l’edit distance tra due sequenze e da qui trovare il miglior allineamento. E se provassiamo ad allineare più di due sequenze? In questo caso bisogna definire una overall distance, il problema però è la complessità che cresce esponenzialmente col numero di sequenze.

C’è un modo che permetta questo allineamento in tempo più o meno lineare? L’idea è quella di trovare le ancore, cioè gli allineamenti locali, e concatenare quelli ottimali. Inoltre, se è possibile, si allineano le regioni tra le ancore.

Gli alberi di keyword sono strutture che permettono l’allineamento di interi genomi, il loro utilizzo è dato dal fatto che algoritmi come BLAST e FASTA sono onerosi in spazio e non sono ottimali.

Come costruiamo un albero di keyword? Data una stringa:

* si parte da un nodo radice e si considera una sottostringa alla volta;
* per ogni carattere della sottostringa, se esiste un ramo corrispondente a esso, allora si percorre, altrimenti si crea un nuovo ramo.

Il problema degli alberi di keyword è che ci sono molti nodi consecutivi che presentano un solo ramo, come possiamo diminuire la dimensionalità del problema? Utilizzando gli alberi di suffissi, una variante degli alberi di keyword in cui tutti i nodi con un solo arco uscente vengono collassati in uno solo, l’immagine a destra ne è un esempio.

Ogni arco risulta quindi essere una sottostringa della stringhe iniziali!

Per ridurre la dimensionalità si utilizza l’edge encoding:

* si assegna a ogni carattere della stringa un numero;
* si assegna a ogni arco il range corrispondente alla sottostringa.

Ora che abbiamo l’albero, come lo utilizziamo? Data una stringa diversa dalla precedente, percorriamo l’albero finché non si ferma:

* se l’algoritmo si ferma su una foglia, otteniamo un suffisso;
* se invece si ferma su un nodo intermedio, abbiamo una sottostringa comune, da qui possiamo ricavare tutti i possibili allineamenti;

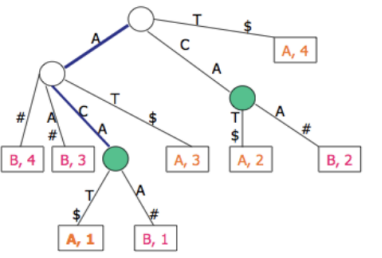
## MUM e MEM

Ora che ci sono le ancore, come si utilizzano?

Esistono due modi per farlo:

* MUM (Minimal Unique Matches), cioè quando una sottosequenza S compare esattamente una volta in entrambe le sequenze, quindi non deve esistere un’altra sottosequenza S’ che contiene S. In poche parole, S è la sottosequenza massima tra le due sequenze;
* MEM (Minimal Exact Matches), è il ragionamento duale al precedente, infatti una sottosequenza S può occorrere più volte. Anche in questo caso si tiene conto delle sole sottosequenze massime.

E’ possibile utilizzare gli alberi di suffissi per estrarre dei multiMEM, delle sequenze MEM in cui occorrono tutte le sequenze, in esse non appaiono tutte i geni ACGT. Precisamente i multiMEM corrispondo ai nodi dell’albero di suffissi, quindi possiamo usarli come ancore per l’allineamento.

E per localizzare le MUM invece? Esiste un algoritmo chiamato MUMmer che permette di farlo, come? Partiamo con le seguenti premesse:

* utilizziamo due caratteri per indicare il fine stringa, uno per genoma A e l’albero per il genoma B;
* utilizziamo un albero di sintassi, nella foglie segnamo a quale genoma appartiene una data sottostringa.

I MUM sono le sequenze più lunghe che possiamo trovare visitando l’albero partendo dalla radice.

Possiamo ordinare le MUM in base alle posizioni in un genoma,si può anche utilizzare una variante della Longest Increasing Subsequence (LIS).

Esistono quattro tipi di gap nell’allineamento MUM:

* SNP: nel caso semplice è un gap base tra due MUM adiacenti, è invece un tandem se tangente a sequenze ripetute;
* Regione polimorfica: se la regione è piccola, l’allineamento è stato fatto con l programmazione dinamica, altrimenti è un’applicazione ricorsiva di MUMmer;
* Inserzione/delezione: un’elemento è fuori dall’ordine di allineamento;
* Repeats: rilevamento di tandem tra MUM in overlap e altri tipi di gap.