# Discussione Slide Tesi

Trascrizione della discussione orale

## Slide 1: «Concetti base di biologia»

Prima di poter definire che cos’è un albero evolutivo, è necessario fornire dei concetti base di biologia, altrimenti non sarebbe possibile capire il senso di questi studi:

Il DNA o acido desossiribonucleico è una macromolecola contenente il patrimonio genetico degli esseri viventi, quindi ne detiene tutta l’informazione genetica ed ereditaria. Possiamo considerarla una sorta di archivio delle informazioni degli esseri viventi!

La struttura è caratterizzata da una doppia elica di lunghezza variabile, composta da quattro tipi di basi azotate:

* Timina (T)
* Adenina (A)
* Guanina (G)
* Citosina (C)

Una successione di basi azotate prende il nome di sequenza.  
Un esempio di una sequenza di DNA può essere la seguente: ATGTAAGACT

## Slide 2: «Che cos’è la bioinformatica?»

Per molti anni l’informatica è stata una scienza a sé stante, tuttavia negli ultimi decenni, grazie al progresso scientifico e tecnologico, sono nate nuove discipline chiamate genericamente X-Informatics. Queste sono il risultato dell’incontro tra l’informatica ed altre scienze di base (quali la biologia, la chimica, l’astronomia, la geologia ecc).

Tra tutte queste discipline, risulta di particolare importanza la bioinformatica. Ma che cosa è?  
La bioinformatica è un campo multidisciplinare della scienza che coinvolge la genetica, la biologia molecolare, l’informatica, la matematica e la statistica, rivolta a studiare sistemi biologici utilizzando metodi e modelli informatici e computazionali. Tale scienza occupa numerose aree di ricerca, quella di cui ci occupiamo in questa discussione è la filogenetica, che studia le relazioni evolutive tra le entità biologiche (dagli esseri viventi fino ai virus) attraverso la costruzione di alberi evolutivi (chiamati anche alberi filogenetici).

## Slide 3: «Albero evolutivo»

Ricordando la definizione di albero, ovvero un grafo non orientato connesso e aciclico, l’albero evolutivo (o albero filogenetico) è un diagramma che rappresenta le relazioni evolutive tra le varie entità biologiche (animali, piante, virus e così via), dove i nodi (o vertici) rappresentano tali entità, mentre gli archi mostrano le relazioni tra di loro.

Gli alberi evolutivi possono essere suddivisi in due tipi: alberi radicati e non radicati.

L’albero radicato o albero con radice si sviluppa a partire da un nodo speciale, chiamato radice e si estende fino alle foglie. I vertici che hanno grado (num di archi incidenti al vertice) maggiore di uno, definiti nodi interni, sono gli antenati, mentre quelli con grado esattamente uguale ad uno, definite foglie, sono le entità attualmente esistenti. La radice, quindi, è l’antenato comune a tutti i vertici dell’albero.

Gli alberi non radicati sono alberi senza la radice. Anche in questo caso, i vertici che hanno grado maggiore di uno sono definiti nodi interni, mentre quelli con grado esattamente uguale ad uno sono dette foglie. A differenza di quelli con radice, si concentrano perlopiù a mostrare le relazioni tra le entità piuttosto che a mostrare l’antenato comune a tutti.

Ma come si costruiscono gli alberi evolutivi?

## Slide 4: «Matrice delle distanze»

Gli algoritmi utilizzati per la costruzione degli alberi evolutivi mostrati in questa tesi prendono il nome di algoritmi basati sulla distanza, in quanto prendono in input una matrice delle distanze.

Dati due punti x ed y, la distanza è una funzione d(x, y) che possiede le seguenti proprietà:

1. Non negatività [mostrare formula]
2. Identità [mostrare formula]
3. Simmetria [mostrare formula]
4. Disuguaglianza triangolare [mostrare formula]

Ma allora date unità, calcolando la distanza per ogni coppia di elementi si ottiene una *matrice D delle distanze*

Ci sono vari modi per calcolare la distanza (ad esempi distanza euclidea, di Manhattan, ecc). In questo esempio [mostrare la matrice di esempio] abbiamo delle sequenze ipotetiche di DNA di quattro specie. In questo caso la distanza è ottenuta calcolando il numero di simboli differenti per ogni coppia di elementi (ad esempio la distanza tra umano e lo scimpanzé è di 3, in quanto la loro sequenza differisce di 3 simboli). Il rispettivo albero evolutivo che si ottiene viene mostrato nella slide successiva.

## Slide 5: «Problema degli alberi basati sulla distanza»

[mostrare albero]

Questo è l’albero evolutivo ottenuto dalla matrice D. Si possono notare delle proprietà importanti su tale albero:

* su ogni arco è presente un numero non negativo. Esso prende il nome di “peso dell’arco”, e rappresenta la distanza tra le foglie.
* Si definisce distanza evolutiva tra due entità biologiche i e j corrispondenti a due foglie dell’albero come la somma del peso degli archi che collegano i e j. Ad esempio la distanza evolutiva tra “Scimpanzé” ed “Umano” è 3, ottenuto dalla somma 1+2 (vedi albero).
* Tutti i vertici hanno grado diverso da 2, quindi si parla di albero semplice.
* *L’albero si adatta alla matrice D.*

Si dice che un albero T si *adatta* ad una matrice delle distanze D se per ogni coppia di foglie i e j si ha che , ovvero l’elemento nella riga i e colonna j nella matrice è uguale alla distanza tra le due foglie i e j in T, in tal caso sia la matrice che l’albero vengono definiti *additivi*. In caso contrario, si parla di *non additività*.

Dopo tutte queste nozioni, possiamo introdurre il problema degli alberi basati sulla distanza:

*Data in* ***input*** *una matrice delle distanze additiva restituire in* ***output*** *un albero evolutivo semplice.*

Quindi l’obiettivo degli algoritmi basati sulla distanza è quello di trovare una soluzione al problema degli alberi basati sulla distanza.

## Slide 6: «Algoritmo per il problema degli alberi basati sulla distanza» - Parte 1

L’obiettivo è quello di costruire un albero semplice T che si adatti alla matrice delle distanze *additiva* D.

Si prenda in considerazione la matrice delle distanze D [mostrare matrice], che è la stessa di un paio di slide fa, quindi u=umano, b=balena, ecc. L’idea di base dell’algoritmo è che all’elemento più piccolo della matrice corrispondano due foglie vicine nel rispettivo albero. Quindi poiché è l’elemento più piccolo della matrice, possiamo supporre che siano vicini e che il loro genitore sia uno generico vertice p.

Come trovare la distanza tra p e f e tra p e b? Usiamo le uniche informazioni che abbiamo, ovvero la distanza tra le foglie in D. Si aggiunge quindi le foglie u ed s all’albero T, tratteggiando il loro arco in quanto ancora non è possibile sapere come sono realmente collocate [mostrare albero] e si riscrivono le distanze in funzione di p [mostrare formule:  e ]. Dalle formule si può notare che l’unica incognita è . Possiamo riscriverlo in funzione delle foglie, ovvero in nel seguente modo [mostrare formula ]. Se sostituiamo questa uguaglianza alle due formule precedenti, riusciamo a trovare e .

## Slide 7: «Algoritmo per il problema degli alberi basati sulla distanza» - Parte 2

Poiché conosciamo la distanza tra f e p e dalla matrice sappiamo che la distanza tra f ed u, basta fare la differenza tra i due valori e troviamo pure (mostrare formula ]. Analogamente si trova (mostrare formula ]. L’albero risultante sarà: [mostrare albero].

Adesso è necessario aggiornare la matrice D: poiché abbiamo già calcolato la distanza di f e b rispetto a p, li possiamo togliere da D ed al suo posto aggiungiamo p stesso, quindi la matrice sarà la seguente [mostrare matrice].

Rimane comunque da capire se u e s abbiano altri genitori. Si ricorda, infatti, che i loro archi sono stati tratteggiati in quanto ancora non si conosce la loro collocazione definitiva nell’albero. Si sceglie un generico nodo interno 𝑘 come genitore di u ed s e si applicano ricorsivamente gli step precedenti, ottenendo quindi [mostrare albero].

## Slide 8: «Algoritmo per il problema degli alberi basati sulla distanza» - Parte 3

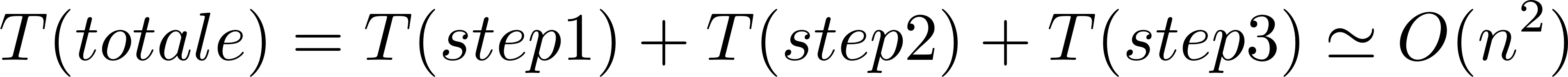
Rimane un ultimo passo da completare, ovvero calcolare la distanza tra k e p, che è ottenuta dalla seguente differenza [mostrare ]. L’albero finale è il seguente [mostrare albero finale!]. L’algoritmo è terminato!!

### Complessità temporale

Per calcolare la complessità nel tempo dell’algoritmo possiamo suddividerlo in tre step:

1. Trovare il minimo in una matrice di dimensione n × n. Si suppone l’utilizzo di un algoritmo di ricerca lineare, la cui complessità nel caso pessimo, dato in input un vettore di lunghezza n, è pari a O(n). Poiché la matrice è nxn, allora T(step1)=O(n^2)
2. Trovare il genitore per ogni coppia di foglie e calcolare la distanza di tutte le n foglie rispetto al genitore stesso. T(step2)=O(n)
3. Calcolare la distanza tra le foglie interne (genitori). Poiché per ogni coppia di foglie c’è un genitore solo, si ha che T(step3)=O(n/2)

Adesso è sufficiente sommare le tre complessità e si trova quella totale dell’algoritmo:

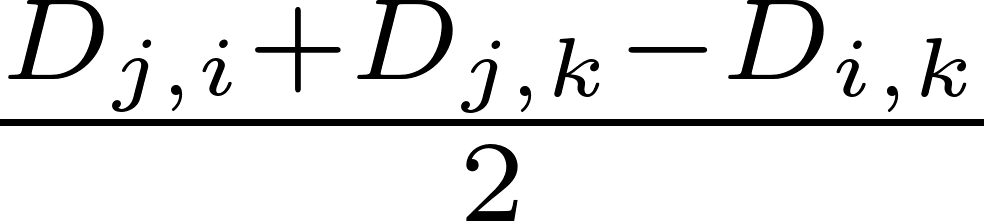
[mostrare formula ].

## Slide 9: «Albero additivo» - Parte 1

L’algoritmo mostrato nella sezione precedente presenta una criticità, infatti riesce a risolvere il problema degli alberi basati sulla distanza solamente se l’elemento più piccolo della matrice D corrisponde a due foglie vicine nell’albero T, ma questo non è necessariamente vero!

Pertanto, si deve approcciare il problema in modo diverso: invece di cercare le foglie vicine in un albero (come visto nelle slide precedenti), l’idea di base dell’algoritmo è quella di costruirlo aggiungendole una alla volta. Per far ciò è necessario conoscere il peso degli archi che collegano le foglie ai rispettivi genitori. Questi prendono il nome di *arti.* In questo ci viene in aiuto il teorema del peso degli arti, che ci permette di calcolarli:

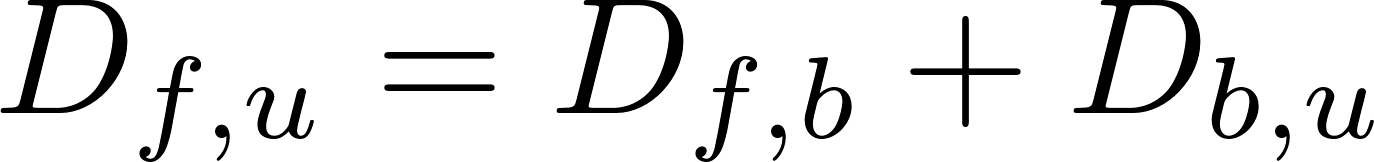
## Teorema del peso degli arti

Sia 𝑙𝑖𝑚𝑏𝑤𝑒𝑖𝑔ℎ𝑡(𝑗) il peso dell’arto di 𝑗. Data una matrice delle distanze additiva 𝐷 ed una foglia 𝑗, 𝑙𝑖𝑚𝑏𝑤𝑒𝑖𝑔ℎ𝑡(𝑗) è uguale al valore minimo di tra tutte le foglie 𝑖 e 𝑘.

A questo punto possiamo spiegare l’algoritmo “albero additivo”. Sia D [mostrare la matrice] una matrice 4x4 dove l’elemento più piccolo della matrice non sono due vicini nel rispettivo albero… (continua slide 10)

## Slide 10: «Albero additivo» - Parte 2

… Gli step da eseguire sono i seguenti:

1. Se la matrice è composta da 2 elementi, allora restituisci l’albero composto dai due elementi, altrimenti;
2. Scegli una qualunque foglia da D (in questo caso scegliamo b) e calcola “limbweight(b)” usando il th del peso degli arti. Omettendo i calcoli (il tempo è quello che è!!) il risultato è 2;
3. Adesso aggiorna D, sottraendo limbweight(b) sia nella riga che nella colonna b (esclusa la diagonale. La matrice è la seguente [mostrare];
4. Applica nuovamente il th del peso degli arti, stavolta impostando la limbweight(b) = a 0. Questo ci permette di capire dove è posizionata la foglia b nell’albero T [mostrare formule ]. il punto di attacco di b è lungo l'arco che collega f con u.

## Slide 11: «Albero additivo» - Parte 3

1. Rimuovi la riga e la colonna b da D e esegui questi step fino a che non ottieni una matrice 2x2
2. Costruisci l’albero T dalla matrice 2x2, pertanto all’inizio avrà solo 2 foglie! Nel nostro esempio l’albero è il seguente [mostrare albero]
3. A questo punto abbiamo tutte le informazioni per costruire l’albero, quindi: Inserisci di volta in volta le foglie in T, il cui peso del loro arto è limbweight. Il risultato di questa operazione è l’albero completo T!

L’algoritmo è terminato!

## Slide 12: «Albero additivo» - Parte 4

Per calcolare la complessità di questo algoritmo, possiamo individuare 3 step:

1. calcola il peso dell’arto dell’n - esima foglia;
2. Aggiorna la riga e l’n-esima colonna in D esclusa la diagonale;
3. Individua il punto in cui va inserita n in T.

Poiché la matrice è di dimensione nxn, allora T(alberoadditivo)=O(n^2). Tuttavia, “AlberoAdditivo” viene eseguito n volte, ovvero fino a che non si costruisce un albero T con n foglie, pertanto la complessità totale dell’algoritmo è [mostrare la formula: = ].

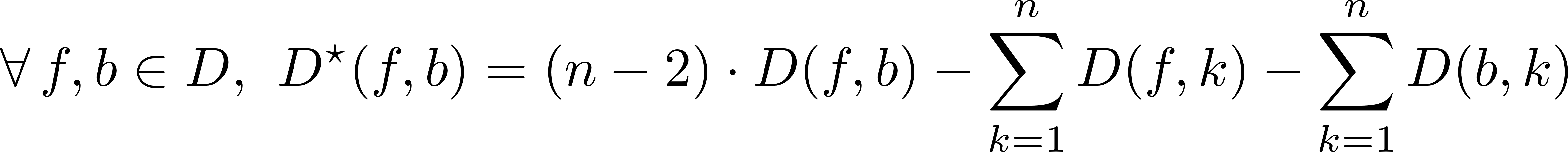
Sebbene “AlberoAdditivo” risolva la criticità dell’algoritmo precedente, comunque non riesce a costruire T se la matrice delle distanze D non è additiva.

È necessario però fare una precisazione: non c’è modo che un albero si adatti ad una matrice non additiva, proprio per definizione di non additività, tuttavia è possibile costruire un albero che approssimi al meglio la distanza tra le foglie della matrice attraverso uno degli algoritmi più importanti della bioinformatica, il Neighbor-Joining. Nel caso in cui la matrice sia additiva, allora il NJ costruisce un albero che si adatta ad essa

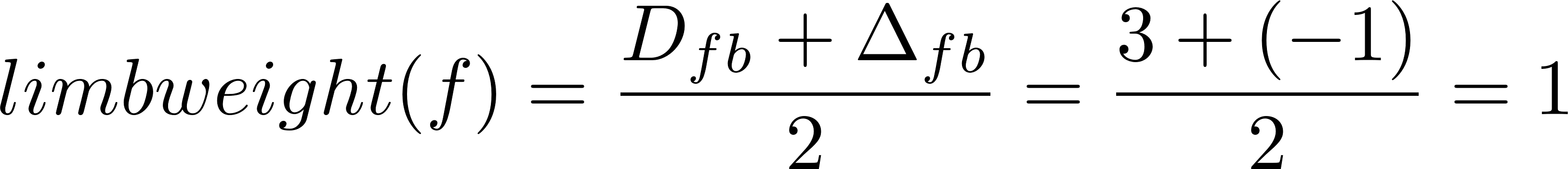
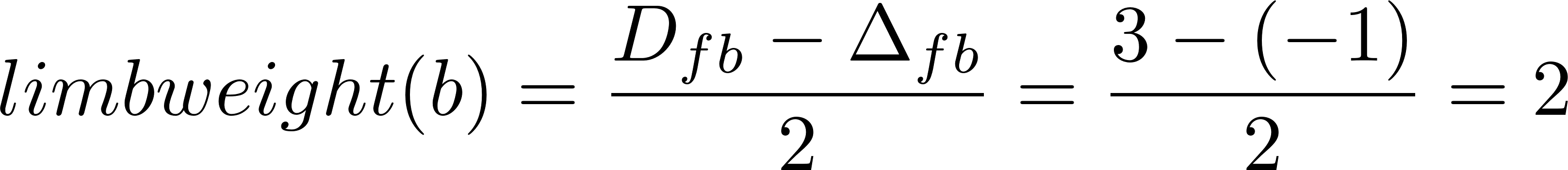
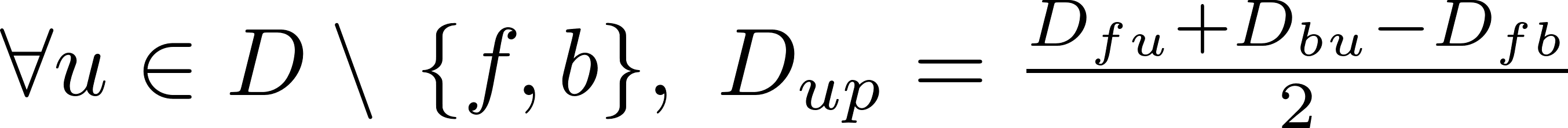
## Slide 13: «neighbor-joining» - Parte 1

Consideriamo la seguente matrice non additiva: [mostrare matrice]. L’obiettivo è quello di costruire un albero T che approssimi al meglio la matrice D!

L’algoritmo è suddiviso in più step:

1. Costruiamo la matrice D★ 🡪 Dato in input D si definisce D★ la seguente matrice [mostrare formula ]. Per semplicità la prima sommatoria la possiamo chiamare “totalDistance(Df)”, mentre la seconda “totalDistance(Db)”. La matrice risultante sarà [mostrare la matrice].  
   La sua caratteristica principale è che qualunque sia la matrice delle distanze in input, l’elemento più piccolo della relativa matrice D★ *corrisponderà sempre* ad una coppia di foglie vicine nell’albero T.

## Slide 14: «neighbor-joining» - Parte 2

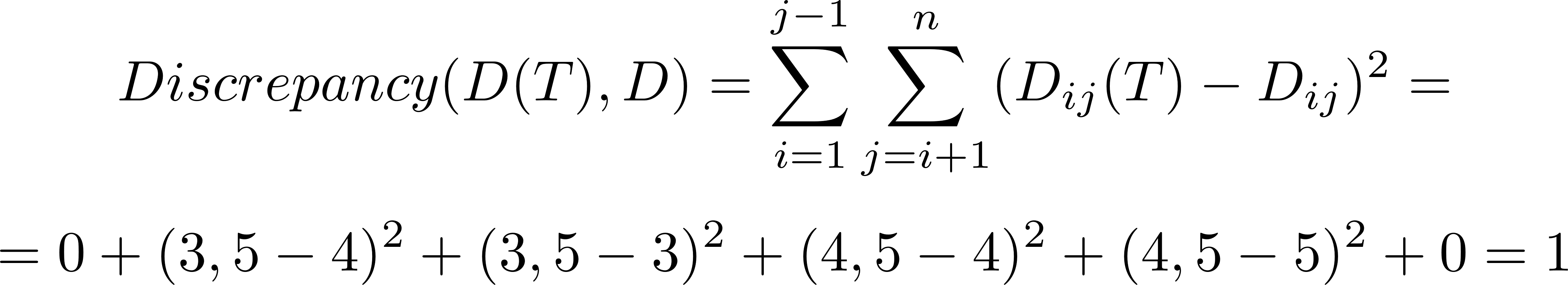
1. Si cerca l’elemento minimo in D★, ovvero D★(f,b). Adesso sappiamo che f e b sono vicine nel rispettivo albero T.
2. si calcola il delta tra totalDistance(Df) e totalDistance(Db), quindi: [mostrare formula ].
3. Si calcola il peso dell’arto di f e di b, quindi: [mostrare le formule , ]
4. Poiché sappiamo sia che f e b sono vicini che il loro peso, possiamo aggiornare la matrice, aggiungendo il loro genitore non noto in D, ovvero una riga ed una colonna p tale che [mostrare la formula ]. Infine, eliminiamo f e b da D.

## Slide 15: «neighbor-joining» - Parte 3

La matrice risultante è [mostrare la matrice].

Esegui i 5 step fino a che non ottieni una matrice 2 × 2: [mostrare la matrice]. Dalla matrice si ricava che p e k sono dei nodi interni legati tramite un arco di peso 1, 5, inoltre in precedenza abbiamo trovato tutte le altre informazioni necessarie per costruire l’albero finale, che sarà: [mostrare albero].  
L’algoritmo è terminato!!

## Slide 16: «neighbor-joining» - Parte 4

Per capire quanto l’albero T approssimi al meglio la matrice D, possiamo costruire la matrice D(T) a partire da T e calcolare la discrepanza tra D e D(T), quindi: [mostrare D(T) e la formula ]. Il risultato mostra che non c’è una grande discrepanza tra D(T) e D.

Complessità temporale:

Per calcolare la complessità possiamo individuare 2 step:

1. Crea D★ e cerca il suo elemento minimo. Poiché la matrice è nxn, allora la complessità è O(n^2)
2. Calcola il *delta*, il peso degli arti ed infine aggiorna la matrice . In particolar modo deve aggiornare n valori della matrice, quindi O(n).

Quindi la complessità del neighbor joining è: T(neighbor joining)=O(n^2). Poiché queste operazioni vengono eseguite tante volte quante sono le foglie nella matrice (ovvero n), allora [mostrare formula].

L’algoritmo neighbor-joining risulta uno degli algoritmi più usati per la costruzione degli alberi evolutivi, insieme al UPGMA.

## Slide 17: « Unweighted Pair Group Method with Arithmetic Mean» - Parte 1

Fino ad ora abbiamo visto algoritmi che costruiscono alberi senza radice. L’algoritmo UPGMA invece, prende in input una matrice delle distanze D additiva o non additiva e restituisce un albero radicato T in cui tutte le foglie sono alla stessa distanza dalla radice. Questo tipo di albero viene definito anche albero ultrametrico.

Tale albero possiede delle caratteristiche, ovvero:

* Le foglie sono le entità biologiche attualmente esistenti;
* I nodi interni rappresentano le speciazioni, quei processi attraverso i quali si formano nuove specie.
* Ogni vertice ha associato un numero non negativo che rappresenta l’età del vertice;
* il peso degli archi è dato dalla differenza tra le età dei nodi;

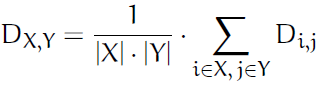
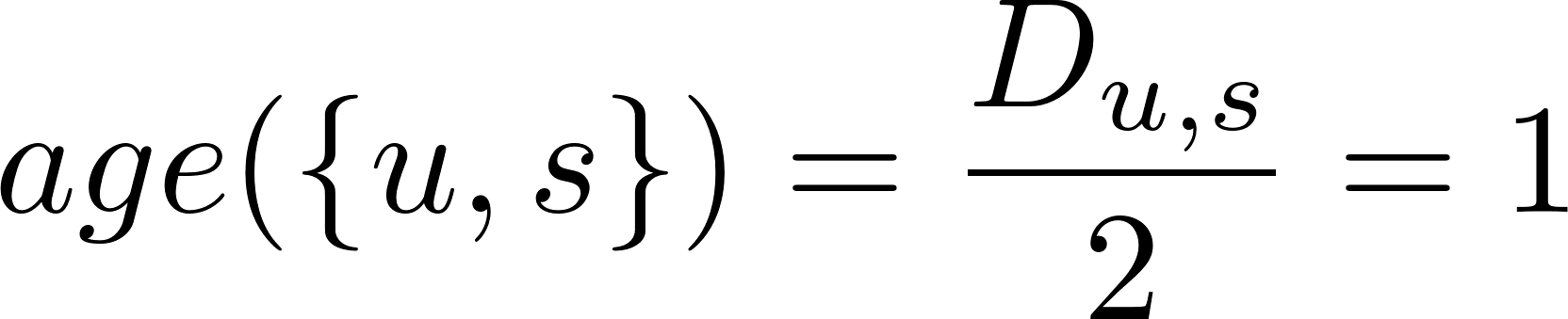
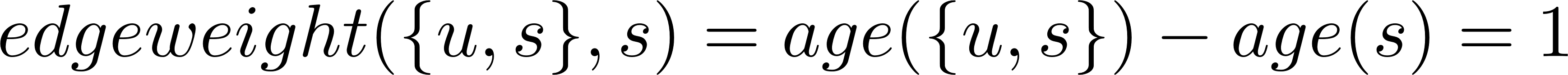
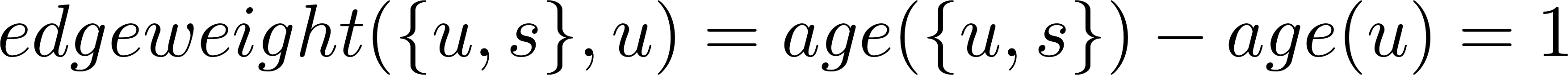
Ogni volta che si verifica una speciazione una linea evolutiva viene divisa in due e quindi da un antenato deriveranno due discendenti.

L’algoritmo UPGMA può essere diviso in più fasi. Si prenda in considerazione la seguente matrice non additiva [mostrare matrice] (da notare che è la stessa usata nel NJ). L’obiettivo è quello di costruire un albero T con radice che approssimi al meglio D.

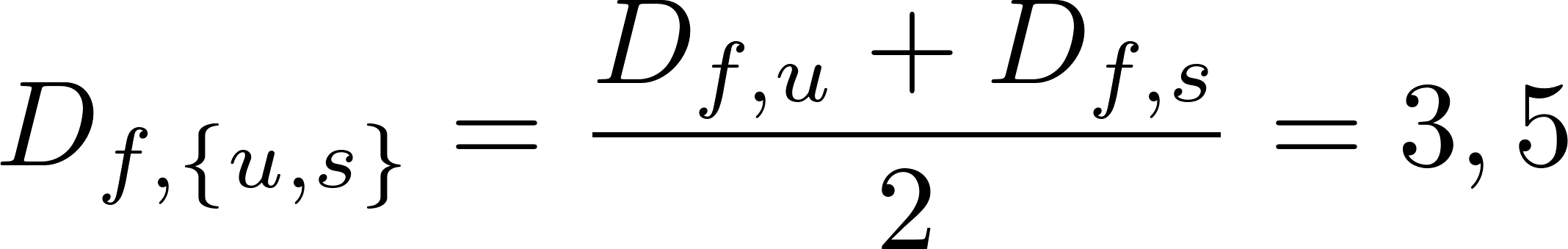
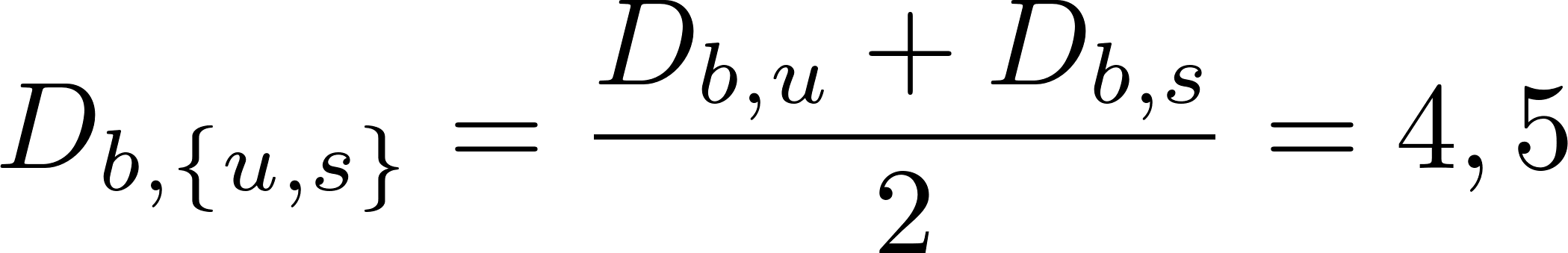
L’algoritmo può essere suddiviso in più step:

1. A partire da si crea cluster, uno per ogni foglia. Nel nostro caso abbiamo 4 foglie, quindi si creano 4 cluster [mostrare immagine]

## Slide 18: « Unweighted Pair Group Method with Arithmetic Mean» - Parte 2

1. Si scelgono i due cluster X ed Y più vicini secondo la seguente definizione di distanza [mostrare formula ]. Poiché in questo caso ciascun cluster è formato da un solo elemento, prendere i due cluster più vicini equivale a scegliere l’elemento più piccolo in D, ovvero D(u,s)=2. Quindi U ed S sono vicini.
2. Crea un cluster Z che è dato dall’unione tra il cluster X ed Y. Nel nostro caso quindi {u,s}={u}unione{s}.
3. Crea in un nodo interno per il cluster , calcola la sua età (ovvero ) ed il peso degli archi di ed . Nel nostro caso quindi: [mostrare formule. Prima  e poi sia  che ]. A questo punto l’albero risultante è [mostrare albero].

## Slide 19: « Unweighted Pair Group Method with Arithmetic Mean» - Parte 3

1. Aggiorna 𝐷 eliminando X ed Y e calcolando la distanza tra 𝑍 e gli altri elementi presenti in D usando la formula dello step 2. Nel nostro caso:[mostra le formule. Prima , poi , ed infine la matrice]

Esegui gli step fino a che non ottieni una matrice di dimensione 2×2. Alla fine si ottiene la seguente matrice [mostrare la matrice] A questo punto l’ultima coppia rimasta (esclusa la diagonale) sarà un unico cluster contenente tutte le specie (ovvero {f,b,u,s}). Essa sarà la radice. L’albero finale è il seguente: [mostrare albero]. L’algoritmo è terminato!

## Slide 20: « Unweighted Pair Group Method with Arithmetic Mean» - Parte 4

Complessità temporale:

Ad ogni iterazione vengono effettuate una serie di operazioni, tra cui aggiornare 𝐷 calcolando la distanza tra il cluster appena inserito in 𝐷 e gli altri elementi, quindi la complessità è 𝑂(𝑛).

Queste iterazioni vengono fatte 𝑛−2 volte, ovvero fino a che non si ottiene una matrice 2×2, quindi 𝑂(𝑛−2)

Allora la complessità totale dell’algoritmo sarà la seguente: .

Con quest’ultima slide la discussione della tesi è terminata!!