Disease: **Severe Acute Respiratory Syndrome (SARS)**

Caused by: a virus from a family of viruses called coronaviruses

Coronaviruses, Virus cúm và HIV đều là virus RNA, chúng lây truyền qua RNA thay vì DNA. Sao chép RNA có tỉ lệ lỗi cao hơn sao chép DNA và các virus RNA có khả năng đột biến thành các chủng khác nhanh chóng. Đây là nguyên nhân bệnh cảm cúm biến đổi qua từng năm và có nhiều chủng nhỏ của HIV.

**SARS-CoV** có nguồn gốc từ động vật. Ban đầu được cho là có nguồn gốc từ chim vì sự tương đồng của sự bùng nổ SARS và dịch cúm gia cầm. Tuy nhiên khi các nhà phân tích đoạn gen 29751 nucleotide của SARS-CoV thì thấy nó không đến từ gia cầm vì bộ gen không giống với coronavirus gia cầm.

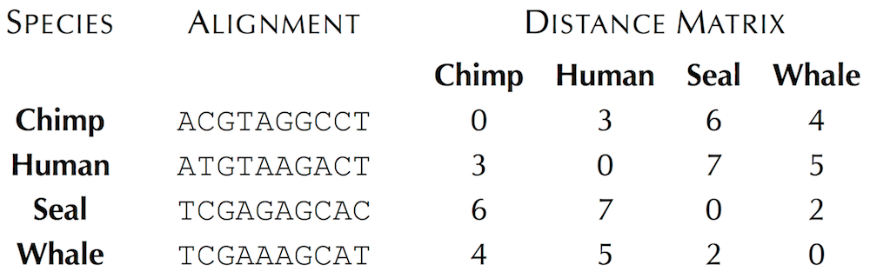
Evolutionary trees (phylogenies)

Các nhà khoa học xét các chủng virus corona từ nhiều loài khác nhau để tìm xem chủng nào giống SARS-CoV nhất. Khó khăn trong việc này là các chuỗi gen thường bị sắp xếp lại, thêm và xoá bớt phần tử. Các nhà khoa học tập trung vào 6 gen trong SARS-CoV, chúng mã hoá Spike protein (protein gai) (theo wiki: đây là protein tạo nên các chân gai của virus), các protein gai thực hiện vai trò xác định và bám lấy khu vực thụ thể trên màng tế bào vật chủ.

Trong SARS-CoV, các protein gai có 1255 amino acid và có ít điểm tương đồng với các chủng virus corona khác. Tuy nhiên các điểm tương đồng nhỏ lẻ này lại đủ để tạo ra sự liên kết đa dạng của protein gai trên nhiều chủng virus corona khác.

Các nhà sinh vật học dựa trên sự căn chỉnh của n chủng loài để tạo nên ma trận khoảng cách D (distance matrix) có kích thước n x n. Di, j thể hiện số kí tự khác nhau giữa 2 gen trên hàng thứ i và cột thứ j.

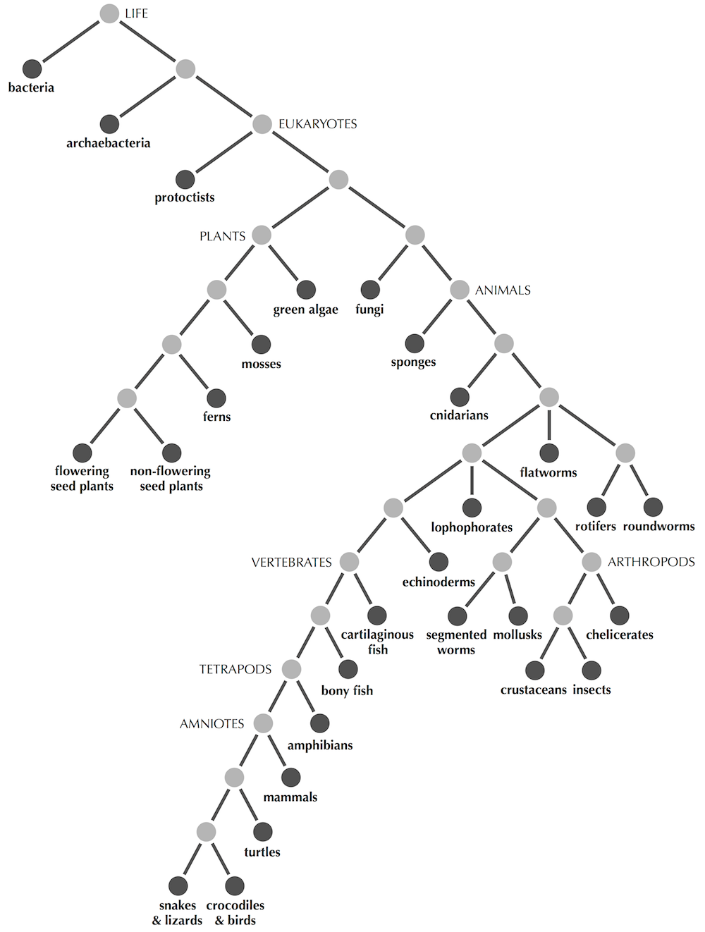
Ma trận khoảng cách có thể được cấu tạo để từ nhiều cách tính khoảng cách khác biệt để phù hợp vào nhiều ứng dụng khác nhau (như khoảng cách thay đổi (edit distance) ở Bioinformatic 3 giữa 2 gens)



Ma trận khoảng cách D phải thoả 3 điều kiện:

* **symmetric** (đối xứng): Di, j = Dj, I với mọi i, j
* **non-negative** (không âm): Di, j > 0 với mọi i, j
* **triangle inequality** (bất đẳng thức tam giác): Di, j + Dj, k > Di, k với mọi i, j, k

Cây tiến hoá dưới dạng biểu đồ



Biểu đồ cây phải thoả 2 điều:

* Các node phải kết nối, có thể đi từ một node bất kỳ sang một node bất kỳ khác.
* Không chứa chu trình.

Các node có 1 cạnh nối vào được gọi là lá.

Không phải mọi ma trận khoảng cách đều có cây. Các ma trận có thể có cây được gọi là khả cộng và ngược lại là không khả cộng.

chúng ta nói rằng một đường đi trong cây là không phân nhánh nếu mọi nút khác ngoài nút bắt đầu và kết thúc của đường đi có bậc bằng 2. Một đường đi không Một đường đi trong cây là không phân nhánh nếu mọi nút khác ngoài nút bắt đầu và kết thúc của đường đi có bậc bằng 2.

Một đường đi không phân nhánh là tối đa nếu nó không phải là một đoạn con của một đường đi không phân nhánh dài hơn.

Nếu thay thế mỗi đường đi không phân nhánh tối đa bằng một cạnh đơn có độ dài bằng độ dài của đường đi đó, thì cây trong hình trên trở thành cây trong hình dưới.

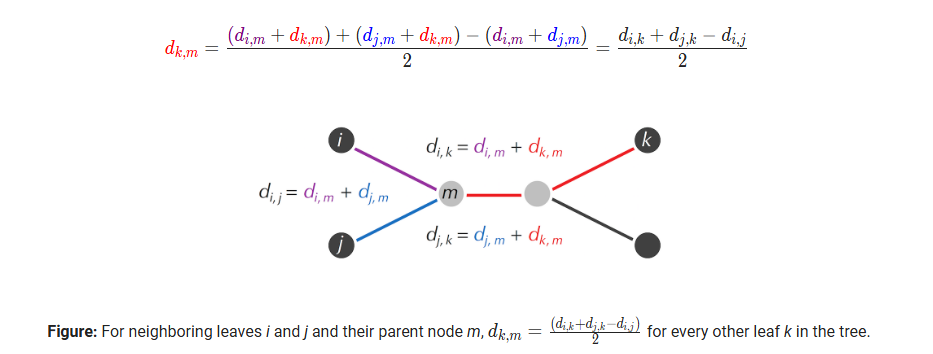
Không còn nút nào có bậc 2; một cây thỏa mãn thuộc tính này được gọi là cây đơn giản.

Để giải quyết Vấn đề Cây Phát Sinh Hệ Dựa Trên Khoảng Cách là đảm bảo rằng hai loài gần nhất với nhau theo ma trận khoảng cách D tương ứng với các nút lá kề nhau trong Cây(D): giá trị nhỏ nhất Di, j nên tương ứng với lá i và j có cùng một nút cha.

Theorem: Mọi cây đơn giản với ít nhất bốn nút đều có một cặp lá kề nhau.

Chứng minh: Giả sử T là một cây đơn giản với ít nhất bốn nút, hãy xem xét một đường đi P = (v₁, ..., vₖ) có số lượng nút lớn nhất trong số bất kỳ đường đi nào trong T. Vì T là liên thông, k phải ít nhất là 3. Hơn nữa, các nút v₁ và vₖ phải là các lá, bởi vì nếu không thì chúng ta có thể kéo dài P thành một đường đi dài hơn. Vì T là đơn giản, mỗi nút bên trong của T phải có bậc ít nhất là 3. Do đó, nút v₂, là nút cha của v₁, phải có ít nhất ba nút kề: v₁, v₃, và một nút khác w.

Chứng minh rằng w là một lá, điều này sẽ dẫn đến việc hai lá v₁ và w là lá kề nhau. Chúng ta sẽ chứng minh bằng phương pháp phản chứng: nếu w không phải là một lá, thì vì T là đơn giản, w sẽ kề với một nút khác u. Kết quả là, chúng ta có thể tạo ra đường đi P' = (u, w, v₂, v₃, ..., vₖ), chứa k + 1 nút và điều này mâu thuẫn với giả thiết ban đầu rằng P có số lượng nút lớn nhất. Do đó, w phải là một lá, điều này dẫn đến việc v₁ và w là lá kề nhau và chứng minh kết quả.



Thuật toán đệ quy cho Vấn đề Cây Phát Sinh Hệ Dựa Trên Khoảng Cách:

1. Tìm một cặp lá kề nhau i và j bằng cách chọn phần tử nhỏ nhất Di,j trong ma trận khoảng cách.
2. Thay thế i và j bằng nút cha của chúng, và tính lại khoảng cách từ nút cha này đến tất cả các lá khác như đã mô tả ở trên.
3. Giải quyết Vấn đề Cây Phát Sinh Hệ Dựa Trên Khoảng Cách cho cây nhỏ hơn.
4. Thêm lại các lá i và j đã được loại bỏ vào cây.

Phương pháp đệ quy trên có thể đã thất bại, vì vậy sẽ khám phá một thuật toán đệ quy khác:

Thay vì tìm kiếm một cặp lá kề nhau trong *Tree*(D), chúng ta sẽ giảm kích thước của cây bằng cách cắt bớt các lá từng cái một. Tất nhiên, chúng ta không biết *Tree*(D), vì vậy chúng ta phải tìm cách cắt các lá trong *Tree*(D) bằng cách phân tích ma trận khoảng cách.

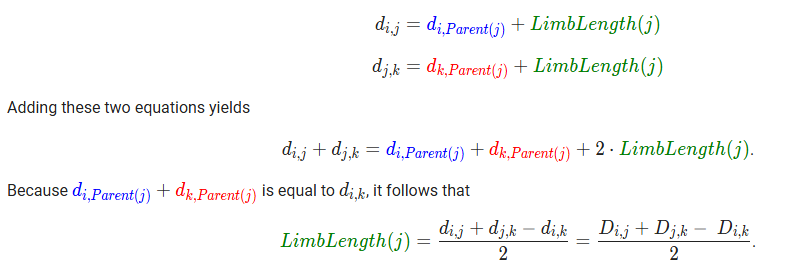
Bước đầu tiên để xây dựng Cây(D) là tính chiều dài của các cành trong Cây(D). Giả sử có một lá j trong cây, gọi chiều dài của cành nối j với nút cha của nó là *LimbLength*(j). Các cạnh không phải cành sẽ nối hai nút bên trong và do đó được gọi là cạnh bên trong.

**Limb Length Problem**: Tính chiều dài của một cành trong cây được định nghĩa bởi một ma trận khoảng cách cộng tính.

Input: Một ma trận khoảng cách cộng tính D và một số nguyên j.

Output: *LimbLength*(j), chiều dài của cành nối lá j với nút cha trong *Tree*(D).

Để tính *LimbLength*(j) cho một lá j cụ thể, ta lưu ý rằng vì *Tree*(D) là đơn giản, chúng ta biết rằng *Parent*(j) có bậc ít nhất là 3 (trừ khi *Tree*(D) chỉ có hai nút). Do đó, ta có thể coi *Parent*(j) như là nút chia các nút khác của *Tree*(D) thành ít nhất ba cây con, hoặc các cây nhỏ hơn sẽ còn lại nếu chúng ta loại bỏ *Parent*(j) cùng với các cạnh kết nối nó với các nút khác (xem hình minh họa bên dưới). Vì j là một lá, nó phải thuộc về một cây con riêng lẻ; ta gọi cây con này là Tj.



Chúng ta có thể tìm Tree(D) theo cách đệ quy trong bốn bước:

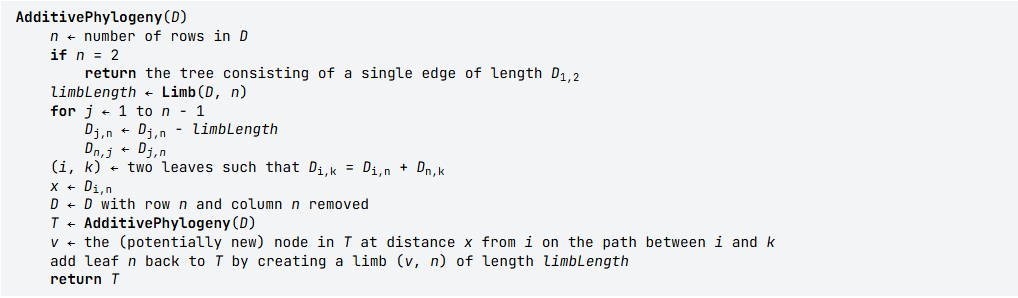
1. Chọn một lá tùy ý j , tính *LimbLength*(j), và xây dựng ma trận khoảng cách *Dtrimmed.*

2. Giải quyết Vấn đề Cây Phát Sinh Hệ Dựa Trên Khoảng Cách cho *Dtrimmed*.

3. Xác định điểm trong *Tree*(Dtrimmed) nơi lá j nên được gắn vào *Tree*(D).

4. Thêm một cành có chiều dài *LimbLength*(j) từ điểm gắn kết này trong *Tree*(Dtrimmed) để tạo thành Tree(D).

Sử dụng thuật toán đệ quy **AdditivePhylogeny** để tìm cây đơn giản phù hợp với ma trận khoảng cách cộng tính n×n. Giả định rằng bạn đã triển khai một chương trình **Limb(D, j)** tính **LimbLength(j)** cho một lá j dựa trên ma trận khoảng cách D. Thay vì chọn một lá tùy ý j từ **Tree(D)** để cắt, **AdditivePhylogeny** chọn lá n (tương ứng với hàng và cột cuối cùng của D).



Thời gian thực hiện lâu: O(n3)

Lập bảng ma trận với một số chủng virus corona từ vài loài, thấy bảng ma trận không cộng tính => Không giải được bằng giải thuật trên.