BIN702 - Série d'exercices sur la phylogénétique

Manuel Lafond

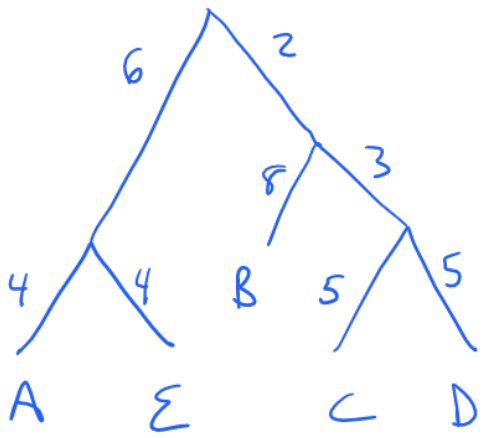
1 Méthodes par distance

Exercice 1: Considérez la matrice de distances suivante :

	A	B	C	D	E
A	0	20	20	20	8
B		0	16	16	20
C			0	10	20
D				0	20
E					0

a. Est-ce que cette distance est ultra-métrique ? Si oui, reconstruisez l'arbre ultramétrique et sinon, dites pourquoi.

Solution. Oui. L'arbre UPGMA est le suivant.



On peut vérifier que les distances dans l'arbre sont les mêmes que dans la matrice, et que les distances racine-feuille sont toutes 10.

□

b. Est-ce que cette distance est additive ?

Solution. Oui. Puisque la distance est ultra-métrique, elle est automatiquement additive. Ceci est parce qu'une distance ultra-métrique est comme une distance additive, mais avec la restriction que chaque distance racine-feuille soit identique. □

Exercice 2: Montrez comment implémenter UPGMA en temps $O(n^2 \log n)$.

Indice 1 : il faut pouvoir mettre à jour les distances après fusion en temps $O(1)$ et utiliser une structure de données qui donne un accès rapide au minimum.

Indice 2 : quand on combine $A \cup B$ et qu'on doit mettre à jour la distance $D_{A\cup B,C}$, au lieu de recalculer la somme des paires de distances possibles pour en faire la moyenne, on peut utiliser $D_{(A\cup B),C} = (|A|D_{A,C} + |C|D_{B,C})/(|A| + |B|)$. Vous pouvez utiliser ce fait en boîte noire. Si vous voulez un défi, montrez que c'est correct.

Solution. Si vous ne le saviez pas déjà, un *monceau* (ou heap en anglais) permet de stocker un ensemble de paires clé-valeur dans lequel on peut ajouter ou supprimer en temps $O(\log n)$, où n est le nombre d'éléments dans la structure. Et, plus important, UPGMA retourne en temps $O(1)$ la paire clé-valeur dont la clé est minimum. Les clés peuvent être répétées. Voir : `https://en.wikipedia.org/wiki/Heap_(data_structure)`.
De plus, lorsqu'on combine deux éléments A et B, on se rappelle qu'on doit mettre à jour la distance entre $(A \cup B)$ et C avec

$$D_{A\cup B,C} = \frac{1}{|A \cup B||C|} \sum_{x\in A\cup B} \sum_{y\in C} D_{x,y}$$

On remarque qu'avant de combiner A et B, on avait déjà accès à $D_{A,C}$, la moyenne des distances de A à C, et à $D_{B,C}$, la moyenne des distances de B à C. On peut calculer la moyenne des distances des $(A \cup B)$ à C en temps constant avec

$$D_{(A\cup B),C} = (|A|D_{A,C} + |C|D_{B,C})/(|A| + |B|)$$

Voir : `https://en.wikipedia.org/wiki/UPGMA`. On peut donc appliquer l'algorithme suivant.

```
fonction upgma(X, D)
    //X est l'ensemble de taxa, D la matrice de distances;
    M = Monceau();
    pour chaque paire d'éléments A, B de X faire
        //D_{A,B} est la clé, (A, B) est la valeur;
        M.insert(D_{A,B}, (A, B)) ;
    fin
    tant que |X| ≥ 2 faire
        done = False;
        tant que done = False faire
            (d, (A, B)) = M.extractMin() ;     // Retire un élément
             avec clé minimum et le retourne
            si A ∈ X et B ∈ X alors
                done = True;
            fin
        fin
        Joindre A à B dans l'arbre;
        Ajouter (A ∪ B) à X;
        Enlever A et B de X;
        pour chaque C ∈ X, où C ≠ (A ∪ B) faire
            D_{(A∪B),C} = (|A|D_{A,C} + |B|D_{B,C})/(|A| + |B|);
            M.insert(D_{(A∪B),w}, ((A ∪ B), w);
        fin
    fin
```

Justification courte.
Soit $n = |X|$. Pour un vétéran d'algorithmique, il n'est pas difficile de voir que le tout prend un temps $O(n^2 \log n)$, car la complexité est dominée par la deuxième boucle. Celle-ci est exécutée $O(n)$ fois et doit faire $O(n)$ insertions dans M, chacune en temps $O(\log n)$. La complexité est donc $O(n^2 \log n)$.

Justification longue.
Il est toutefois instructif d'entrer dans les détails de l'analyse. Notons que la première boucle prendra un temps $O(n^2 \log n)$, car il y a $O(n^2)$ paires d'éléments et chaque insertion prend $O(\log n)$.

La deuxième boucle "tant que $|X| \geq 2$" s'exécute $O(n)$ fois, car à chaque tour on combine deux éléments (et il y a n éléments au total).

La boucle "tant que $done = False$" est un peu plus subtile. Dans le pire des cas, on fera un $extractMin$ pour chaque élément qui sera inséré à un moment

ou à un autre dans M. Le nombre de $extractMin$ que l'on fera sera égal au nombre $O(n^2)$ d'insertions faites au départ, plus le nombre total d'insertions faites dans la toute dernière boucle. On fait $O(n)$ insertions par tout de boucle, donc au total, on fera $O(n^2)$ insertions dans cette dernière boucle. Le nombre de $extractMin$ que l'on fera, au total dans toute l'exécution de l'algorithme, sera $O(n^2 + n^2) = O(n^2)$. Puisque chaque $extractMin$ prend un temps $O(1)$, on passe un temps total de $O(n^2)$ dans les extarctions.

Finalement, à chaque tour de boucle, on joint deux éléments, on enlève de X et on ajoute à X, chaque opération se faisant en temps $O(n)$. Ensuite, encore à chaque tour de boucle, on fait $O(n)$ insertions dans M, chacune en temps $O(\log n)$. Le temps d'un tour de boucle est donc borné par $O(n + n \log n) = O(n \log n)$. Puisqu'il y a $O(n)$ tours de boucle, le temps passé dans la boucle principale est borné par $O(n \cdot n \log n) = O(n^2 \log n)$.

Au total, le temps de l'algorithme est

$$
\begin{aligned}
&O(\text{“temps de la 1re boucle”} + \text{“temps des extractMin”} + \text{“temps de la 2e boucle”}) \\
&= O(n^2 \log n + n^2 + n^2 \log n) \\
&= O(n^2 \log n)
\end{aligned}
$$

□

Exercice 3: Soit D une distance ultra-métrique et soit T l'arbre reconstruit avec UPGMA. Dans la version présentée en classe, nous n'avons pas montré en détail comment étiqueter les arêtes de façon à respecter les distances, et de façon à ce que chaque distance feuille-racine soit la même.
Donnez un algorithme qui trouve un étiquetage de T satisfaisant les conditions d'ultramétricité.

Solution.
On observe que si T est un arbre ultra-métrique, pour n'importe quel sommet x, la distance de x à une feuille descendante de x est toujours la même. Donc la propriété d'ultra-métricité est vraie pour tous les sous-arbres.
On peut donc parcourir T de bas en haut de façon à ce que, pour chaque noeud x, le sous-arbre T_x soit ultra-métrique. Après avoir traité T_x, on stocke la distance de x à ses feuilles et on réutilisera l'information plus haut.

fonction *ultraMetricDist(T, D)*

$R = [\,]$; // $R[x]$ entreposera la dist de x à ses feuilles.
dist_branche $= [\,]$; // Ce qu'on va retourner.
pour chaque *noeud x dans un parcours post-ordre* **faire**
  si *x est une feuille* **alors**
    $R[x] = 0$;
  sinon
    Soient x_1, x_2 les enfants de x;
    Soit a une feuille sous x_1;
    Soit b une feuille sous x_2;
    $d = D_{a,b}$; // d est le même pour tout choix de a, b
    //Pour rendre T_x ultra-métrique, on envoie $d/2$ de chaque ;
    //côté en ajustant les branches selon $R[x_1]$ et $R[x_2]$;
    $dist\_branche[x, x_1] = d/2 - R[x_1]$;
    $dist\_branche[x, x_2] = d/2 - R[x_2]$;
    $R[x] = d/2$;
  fin
fin
return *dist_branche*;

□

Exercice 4: Quelle est la complexité de l'algorithme Neighbor-Joining ? On peut argumenter qu'il s'implémente en temps $O(n^3)$, où n est le nombre de taxa qui étaient présent au départ.

Solution.
Rappelons l'algorithme de Neighbor-Joining tel que présenté en classe.

fonction *NJ(D)*

Soit T un arbre étoile avec une feuille par taxa;
$n =$ nombre de taxa;
tant que *le centre de T a au moins* 4 *voisins* **faire**
  Choisir i, j qui minimisent $(n-2)D_{i,j} - \sum_{k=1}^{n}(D_{i,k} - D_{j,k})$;
  Joindre i et j dans T, appeler le nouveau noeud créé u;
  $n = n - 1$;
  pour chaque $k \neq i, j, u$ **faire**
    $D_{u,k} = 1/2(D_{i,k} + D_{j,k} - D_{i,j})$;
  fin
fin
return T;

On fait $O(n)$ tours de boucle, et la recherche du i, j minimum prend un temps $O(n^3)$. Ceci donne une complexité $O(n^4)$. On peut faire mieux en maintenant un tableau Q des valeurs à optimiser.

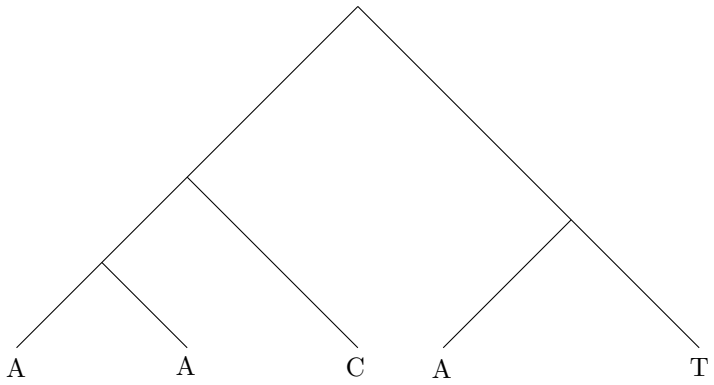
fonction *NJ(D)*
- Soit T un arbre étoile avec une feuille par taxa;
- $n =$ nombre de taxa;
- Soit Q un tableau $2D$;
- **pour chaque** i, j **faire**
  - $Q_{i,j} = (n-2)D_{i,j} - \sum_{k=1}^{n}(D_{i,k} - D_{j,k})$;
- **fin**
- **tant que** *le centre de* T *a au moins* 4 *voisins* **faire**
  - Choisir i, j qui minimisent $Q_{i,j}$;
  - Joindre i et j dans T, appeler le nouveau noeud créé u;
  - $n = n - 1$;
  - **pour chaque** $k \neq i, j, u$ **faire**
    - $D_{u,k} = D_{k,u} = 1/2(D_{i,k} + D_{j,k} - D_{i,j})$;
  - **fin**
  - //Mettre à jour $Q_{u,k}$ pour tout k;
  - **pour chaque** $k \neq i, j, u$ **faire**
    - $Q_{u,k} = (n-2)D_{u,k} - \sum_{l=1}^{n}(D_{u,l} - D_{k,l})$;
  - **fin**
  - //Mettre à jour $Q_{a,b}$ pour les autres paires;
  - //La différence : on ne compte plus $-D_{a,i}, -D_{a,j}, -D_{b,i}, -D_{b,j}$;
  - //la valeur de n a changé, et on compte $-D_{a,u}, -D_{b,u}$;
  - **pour chaque** $a, b \neq i, j, k$ **faire**
    - $Q_{a,b} = Q_{a,b} - D_{a,b}$ //la valeur de n a changé;
    - $Q_{a,b} = Q_{a,b} + D_{a,i} + D_{a,j} + D_{b,i} + D_{b,j}$ //car i, j supprimés;
    - $Q_{a,b} = Q_{a,b} - D_{a,u} - D_{b,u}$ //car on a ajouté u;
  - **fin**
- **fin**
- return T;

La boucle qui initialise Q prend un temps $O(n^3)$. On peut ensuite vérifier que chaque boucle imbriquée prend u ntemps $O(n^2)$, donc la boucle pricipale prend un temps $O(n^3)$.

□

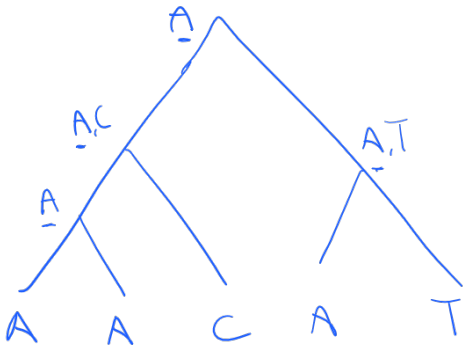
2 Méthodes par caractères

Exercice 5: Soit l'arbre suivant :



a. Exécutez l'algorithme permettant de minimiser la distance Hamming, i.e. les changements sur cet arbre. Donnez les caractères candidats aux ancêtres ainsi qu'un étiquetage optimal.

Solution.
L'étiquetage optimal place un A partout.

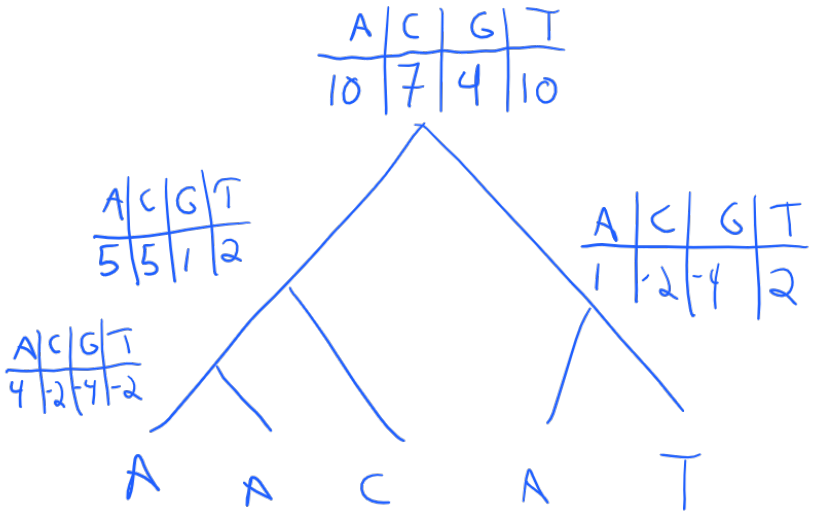


□

b. Exécutez l'algorithme de programmation dynamique permettant de maximiser le score des branches sur cet arbre. Utilisez comme matrice de score

	A	C	G	T
A	2	-1	-2	-1
C	-1	2	-1	-1
G	-2	-1	1	-2
T	-1	-1	-2	3

Solution.



Une assignation optimale serait de placer un A partout.

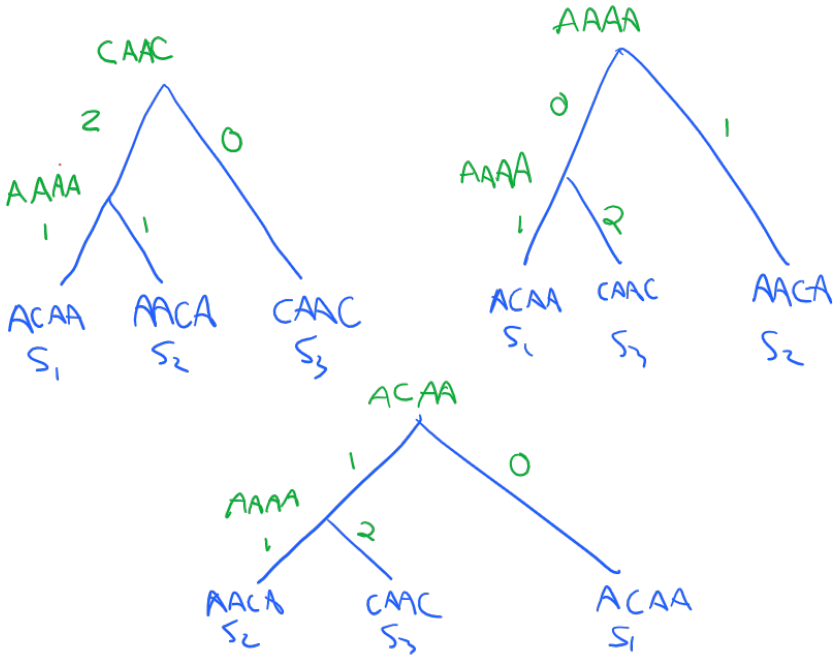
□

Exercice 6: Supposez qu'il n'existe que les caractères A et C. Soient les séquences

$$S_1 = ACAA$$
$$S_2 = AACA$$
$$S_3 = CAAC$$

Quel est l'arbre le plus parcimonieux, selon la distance Hamming ? C'est-à-dire, quel arbre minimise la somme des nombres de changements sur les branches pour l'ensemble des positions ?

Solution.
Il n'y a que 3 arbres, donc on peut se permettre de tous les évaluer. Sur chaque arbre, j'ai manuellement exécuté l'algorithme sur chaque caractère. Le figure ci-bas donne les solutions optimales sur chaque arbre. Il s'avère qu'ils sont tous optimaux et mènent chacun à un minimum de 4 changements.



□

Exercice 7: Dans l'algorithme de Sankoff-Rousseau, qui calcule le score maximum sur les branches d'un arbre, nous avons supposé que les noeuds avaient deux enfants. Montrez comment modifier l'algorithme pour trouver un étiquetage optimal des noeuds internes, et ce même si l'arbre n'est pas binaire (donc chaque noeud peut avoir un nombre arbitraire d'enfants).

Solution. Trop facile. Au lieu d'additionner le max des deux enfants dans la récurrence, on additionne le max de tous les enfants.
Les cas de bases restent les mêmes qu'en classe. Pour un noeud interne v, on dénote par $enf(v)$ les enfants de v. En utilisant les notations présentées en classe, on aura

$$V[v,a] = \sum_{v_i \in enf(v)} \max_{b \in \Sigma}(V[v_i,b] + M[a,b])$$

□

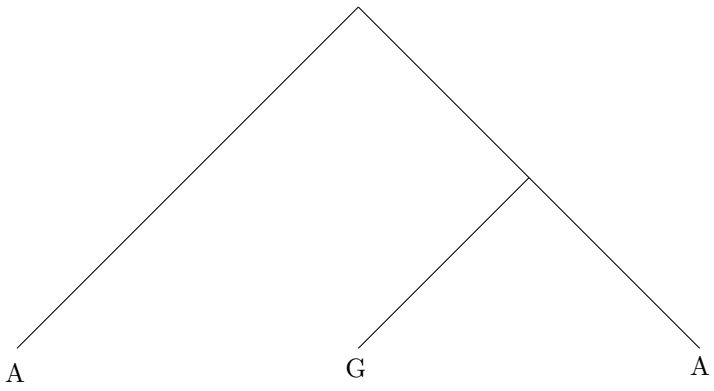
Exercice 8: En classe, nous avons mentionné le nombre d'arbres binaires **enracinés** avec n feuilles étiquetées, qui est de $(2n-3)!! = (2n-3)(2n-$

$5)(2n-7)\ldots(3)(1)$. Démontrez cette identité.
Indice : vous pouvez utiliser le fait qu'un arbre binaire enraciné avec n feuilles a toujours $2n-2$ arêtes.
Par ailleurs, pour obtenir un arbre avec n feuilles, on peut démarrer d'un arbre avec $n-1$ feuilles puis "greffer" une feuille sur une des branches.

Solution.
On suppose qu'on a démontré que tout arbre enraciné binaire avec n feuilles a $2n-2$ arêtes.
Soit $B(n)$ le nombre d'arbres enracinés binaires avec n feuilles étiquetées. On commence avec $n=2$. Le cas $n=1$ est particulier (car $0!!=1$). Pour montrer que ça fonctionne, on regarde $B(2), B(3), B(4)$, puis on généralise.
On a $B(2)=1$ car un seul arbre à 2 feuilles est possible. Aussi, $(2n-3)!! = (4-3)!!=1$. Ça marche.
On a $B(3)=3\cdot B(2)=3$. Ceci est parce que pour obtenir un arbre à 3 feuilles, on peut prendre l'arbre à 2 feuilles, subdiviser une de ses deux branches (i.e., on ajoute un noeud sur une des branches) et on ajoute la 3ème feuille comme enfant du noeud créé. Une autre façon d'avoir un arbre à 3 feuilles est aussi de prendre l'arbre à deux feuilles, de créer une nouvelle racine, et de lui donner comme enfant cet arbre et la 3ème feuille. Ceci donne trois façons, ce qui est égal à $(2n-3)!! = 3!! = 3\cdot 1 = 3$. Ça marche toujours.
On a $B(4)=5\cdot B(3)=15$. Ceci est parce qu'on peut prendre n'importe lequel des $B(3)$ arbres à 3 feuilles et greffer une feuille sur une des $(2\cdot 3-2)=4$ branches, en plus de pouvoir créer une nouvelle racine sous laquelle on met la 4eme feuille. On a $(2n-3)!! = 5!! = 5\cdot 3 = 15$. Ça marche.
Plus généralement, on a $B(n)=(2n-3)B(n-1)$. Ceci est parce qu'on peut prendre un des $B(n-1)$ arbres à $n-1$ feuilles et greffer la feuille n sur une des $2(n-1)-2=2n-4$ branches, en plus d'avoir l'option de créer une nouvelle racine. Pour chaque arbre de $B(n-1)$, il y a donc $2n-4+1=2n-3$ options. Donc, $B(n)=(2n-3)B(n-1)=(2n-3)(2(n-1)-3)!!=(2n-3)(2n-5)!!=(2n-3)!!$.

□

Exercice 9: Supposez qu'il n'existe que deux caractères possibles A et G. Soit l'arbre suivant :



Considérez que sur chaque branche, la probabilité qu'un caractère reste le même est 2/3 et la probabilité de changer est 1/3. On suppose que chacun des caractères a une probabilité 1/2 d'être à la racine. Quel est le score de vraisemblance de cet arbre ?

Solution. Il faut considérer les 4 étiquetages possibles aux noeuds internes et additioner leur probabilité. Pour $x, y \in \{A, G\}$, on dénote par (x, y) le scénario où x est à la racine et y à l'autre noeud interne. Les probabilités des scénarios sont comme suit (on liste les probabilités de générer le caractère à la racine, puis celle de la branche gauche, puis de sa branche droite, puis des branches gauche puis droit au dernier niveau) :

- $(A, A) : \frac{1}{2}\frac{2}{3}\frac{2}{3}\frac{1}{3}\frac{2}{3} = \frac{4}{81}$
- $(A, G) : \frac{1}{2}\frac{2}{3}\frac{1}{3}\frac{2}{3}\frac{1}{3} = \frac{2}{81}$
- $(G, A) : \frac{1}{2}\frac{1}{3}\frac{1}{3}\frac{2}{3}\frac{1}{3} = \frac{1}{81}$
- $(G, G) : \frac{1}{2}\frac{1}{3}\frac{2}{3}\frac{2}{3}\frac{1}{3} = \frac{2}{81}$

La vraisemblance de l'arbre est $\frac{4+2+1+2}{81} = \frac{9}{81} = \frac{1}{9}$

□

Exercice 10: (Défi complètement optionnel que vous devrez regarder seulement si vous êtes très enthousiaste)
Il existe une caractérisation des matrices ultramétriques qui ne regarde que trois points à la fois. Elle s'appelle la condition des trois points et dit qu'une distance D est ultra-métrique si et seulement si pour chaque trois taxa, on peut les nommer x, y, z de façon à ce que $D_{xy} = D_{xz} \geq D_{yz}$. Vous pouvez vérifier ce fait sur les exemples vu en classe.
Démontrez la condition des trois points.

Solution.
Il n'y aura pas de question de ce style à l'examen. Si c'est tout ce qui vous inéresse, passez cette preuve et allez au prochain numéro ! Sinon, voici.

On dénote $lca_T(a, b)$ le dernier ancêtre commun de a et b dans un arbre T. On écrit aussi $dist_T(a, b)$ pour dénoter la somme des poids des branches sur le chemin de a à b dans T. Rappelons que pour démontrer un "si et seulement si", il faut montrer les deux directions de l'implication.

($\Rightarrow$) : supposons que D est ultra-métrique. Soit T un arbre satisfaisant les conditions d'ultra-métricité. Soient x, y et z. Supposons que x, y, z soient choisis de façons à ce que $lca_T(x, y)$ soit un descendant de $lca_T(x, z)$ (sinon, on permute les rôles de x, y, z). On note que $lca_T(x, z) = lca_T(y, z)$. Dénotons $p = lca_T(x, y)$ et $q = lca_T(x, z)$. On peut observer que $dist_T(x, p) = dist_T(y, p)$, parce que T est ultra-métrique (s i ce n'était pas le cas, impossible que $dist(x, racine) = dist(y, racine)$). De la même façon, $dist_T(x, q) = dist_T(y, q) = dist_T(z, q)$. Ceci veut dire que

$$dist_T(x, z) = dist_T(y, z) = 2 \cdot dist_T(x, q)$$
$$dist_T(x, y) = 2 \cdot dist_T(x, p)$$

Il est évident que $dist_T(x, p) \leq dist_T(x, q)$, et donc on a $dist_T(x, z) = dist_T(y, z) \geq dist_T(x, y)$, la propriété recherchée.

($\Leftarrow$) : on doit maintenant prouver l'autre direction. C'est-à-dire, si $D_{xy} = D_{xz} \geq D_{yz}$, alors il est possible de reconstruire un arbre ultra-métrique pour D. On peut le démontrer par induction sur n, le nombre de taxa. Si $n = 2$, D est trivialement ultra-métrique. On suppose donc par induction que si une distance D' a $n - 1$ taxa et satisfait la condition des 3 points, alors on peut construire un arbre T. Prenons D avec ses n taxa. Soient a, b les taxa qui minimisent $D_{a,b}$. Soit D' la distance obtenue de D en retirant a. Alors on peut appliquer l'induction sur D' car elle satisfait toujours la condition des 3 points. Soit T' un arbre pour D. Alors, b est une feuille de T' et la longueur de la branche de b à son parent est au moins $D_{a,b}/2$ (sinon, $D_{a,b}$ ne serait pas minimum - si vous ne voyez pas pourquoi, demandez-moi !). Soit p le parent de b sur T' et soit ℓ la longueur de cette branche. On peut greffer a sur la branche (p, b), c'est-à-dire créer un nouveau noeud q entre p et b et ajouter a comme enfant de q. On attribue une longueur de $D_{a,b}/2$ à (q, b) et à (q, a), et une longueur de $\ell - D_{a,b}/2$ à (p, q). Ceci ne change pas les distances entre b et un autre noeud de T'. De plus, la distances entre a et un noeud x dans ce T' modifié est la même pour a et pour b. Vous pouvez vérifier que $D_{a,b}$ étant

minimum, la distance D devait satisfaire cette propriété. On déduit que T' avec a ajouté contient bien les distances de D.

□