Phylogenetic Trees

All species of organisms on our planet undergo slow transformation throughout ages. This process has been identified by the biologist as *evolution.* One central problem in biology is to explain the evolutionary history of species and in particular, how species are related to each other and whether or not they shared a common *ancestor*. This is depicted by constructing a tree whose leaves represent the present day species and whose internal nodes represent possible ancestors. Such a tree is called a **phylogenetic** tree. With the advent of molecular biology, the evolutionary processes have been linked to several basic processes at the genome level such as insertion, deletion, substitution, inversion and transposition of its DNA. All these operations are grouped under a common name called *mutation.* In the past, biologists used morphology data (the so-called phenotypes: color of hair, skin, eye, physical characteristics like presence of wings, length of arms, legs etc) or biochemistry data (such as amino acid synthesis pathways) to come up with taxonomy and ancestral relationship. In recent times, use of molecular sequence data has given rise to a more precise science of phylogenies which incorporates mathematical and algorithmic approaches. There are also many software tools that have been designed based on these algorithms to re-create phylognetic trees. There is a vast amount of literature and competing theories of evolutionary biology and classification of species. Our aim here is very much limited to studies of a few algorithmic approaches for constructing phylogenetic trees.

An example of a phylogenetic tree is shown in Fig.1. The tree is an undirected acyclic graph. The leaves of the tree denote denote “objects” under study which could be a group of seemingly related organism, mammals, birds, reptiles or DNA or amino acid sequences which also undergo evolution along with the evolution of the associated organisms. An internal node represents a hypothetical common ancestor of all the leaf objects under it. The topology or *the branching patterns* shows the history of evolution of the object via the unique path from it to the *root of the tree* which is supposed to be the common ancestor of all the objects under study. The existence of a root node in practical situations is not always guaranteed because of lack of enough biological evidence but this is a challenge that the biologists constantly face with. .If we assign a *weight* to each edge of the tree, we can define the concept of *evolutionary distance* between a pair of nodes. If the weight of an edge denotes estimated elapsed time to evolve or transform the two objects connected by the edge, the distance between an internal node to a leaf in the sub-tree under the internal node, may be interpreted as the total estimated elapsed time for the object representing the internal node to evolve to the leaf object.

Tree-Building Algorithms

The phylogenetic tree construction algorithms can be classified into two broad classes: ***maximum parsimony*** based methods and ***distance-based*** methods.

Parsimony Based Method

This is also known as the *character-based* method. The input is a set of characters or *attributes* that the objects may posses. The input characters are chosen for biological significance for evolutionary studies. The characters used have traditionally been *morphological* features such as having a back-bone or wings . But characters can also be based on DNA or protein sequences possessed by different species. For example presence of a particular amino-acid sequence for a given protein as a substring may be an attribute of relevance.

We will make some simplifying assumptions contrary to certain counter-examples found in nature: first, we assume that the attributes or characters can be inherited independently from one another and that reverse inheritance is not possible. Reverse inheritance means that an attribute can be gained by inheritance, then lost and then gained back again. We also assume that that the characters are *homologous*, that is, all observed states of a given character (viz. if the beak of a bird can assume four different structures, the *state* for this attributes has four values) from one original state of the nearest common ancestor for the objects under study. We also make a third assumption: *parallel evolution or convergence state* does not happen. This says that if two objects share an attribute, they should have a common ancestor.

The characters could be *ordered* or *unordered*. In general, if a character has *r* values, the character can assume any one of the *r* values. For an unordered character, we assume that any state can change to any state (although the same state may not repeat – reverse inheritance). For an ordered character, the state changes may follow a particular specified total or partial order. For example, a linear order $3\leftrightarrow4\leftrightarrow1\leftrightarrow2$ means that a transition from state 3 to 1 or from 1 to 3 has to go through an intermediate state 4. Note the state transitions are not **directed,** simply ordered. Thhe transition is **undirected** if there is a precedence relation between pairs of state transitions. For example, state 3 precedes state 4 which precedes state1 etc. The unordered, ordered and directed characters are also known as qualitative, cladistic and polar characters , respectively. The **perfect phylogeny** can now be formally defined as follows:

Definition: Let *M* be an *n* by *m* *r*-ary matrix, representing *n* objects each having *m* attributes or characters. The *i*-th row of *M* represents the *i*-th object. The *j*-th column represents the *j*-th character. A *perfect phylogenetic* tree for *M* is a rooted tree *T* with exactly *n* leaves that obeys the following properties:

1) Each of the *n* objects labels exactly one leaf of *T.*
2) For each state *s* (s=0,1,..,r) of each character c(c=1,2,…, m), the set of all *u* nodes (leaves and interior nodes) for which the state is *s* with respect to *c* must form a subtree of *T* (that is, a connected subgraph of *T*). This means that the edge leading to this subtree is uniquely associated with a transition from some state *w* to state *s*.

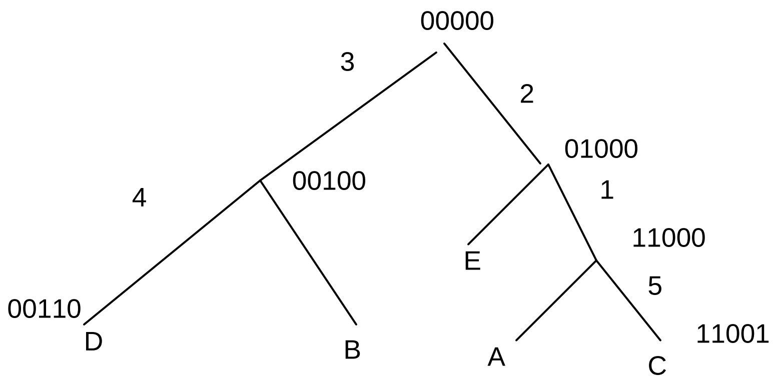
The perfect phylogeny is special case of maximum parsimony tree. The perfect phylogeny tree with binary characters is defined as:
Definition: Let *M* be an *n* by *binary* (0-1)- matrix, representing *n* objects each having *m* attributes or characters. The *i*-th row of *M* represents the *i*-th object. The *j*-th column represents the *j*-th character. A *perfect phylogenetic* tree for *M* is a rooted tree *T* with exactly *n* leaves that obeys the following properties:

1) Each of the *n* objects labels exactly one leaf of *T*.
2) Each of the *m*- characters labels exactly one edge.
3) For any object, the characters that label the edges along the unique path from the root to leaf , specify all the characters of the object whose state is '1'

	1	2	3	4	5
A	1	1	0	0	0
B	0	0	1	0	1
C	1	1	0	0	1
D	0	0	1	1	0
E	0	1	0	0	1

	1	2	3	4	5
A	1	1	0	0	0
B	0	0	1	0	0
C	1	1	0	0	1
D	0	0	1	1	0
E	0	1	0	0	1

Example: The first matrix has no perfect phylogenetic tree. The second matrix *M* has a solution tree as shown below.



Perfect Phylogeny Problem

For n objects, it can be shown that there are $\prod_{i=3}^{n}(2i-5)$ possible labeled trees which grows faster than $n!$, so building all trees and then deciding which one of them is a perfect phylogeny is out of question. For unordered characters, the problem is NP-complete. For ordered character, there exist algorithms with complexity polynomial in *n*, *m* and *r*. For the binary matrix case, the problem is: Given a *n* by *m* 0-1 matrix *M*, determine whether there is a phylogenetic tree for *M*, and if so, build one. This problem has a *O*(*nm*) solution. For convenience of proof, it will be simpler to transform *M* to a matrix $\overline{M}$ whose columns are sorted in a in decreasing order from left to right. For our example, we have $\overline{M}$ as shown below

Cols. of M Cols. of $\overline{M}$	2 1	1 2	3 3	5 4	4 5
A	1	1	0	0	0
B	0	0	1	0	0
C	1	1	0	1	0
D	0	0	1	0	1
E	1	0	0	0	0

It is obvious if M has a perfect tree so does $\overline{M}$ which is simply a reordering of the columns. The edges of the tree has to be renamed according to the permutation given on top of the table.

Definition: For any column k of $\overline{M}$, let O_k be the set of objects (rows) that have '1' in column k.

Theorem: Matrix M (or $\overline{M}$) has a phylogenetic tree if and only if for every pair of columns i,j either O_i and O_j are disjoint or one contains the other.

Proof: A detailed proof is given in Gusfield p.462. The proof can be understood easily if we note that each character or attribute appears only once in the phylogenetic tree. Thus, the characters that appear in the left side of a branching point must be disjoint from those appearing on the right side of the same branching point and this is true for every interior node. Second, once a character *c* appears in an edge, this introduces a '1' in all object vectors that are in the leaves of the subtree below this edge. The succeeding nodes in any path of this subtree must obey the containment requirement.

A straightforward algorithm to test the validity of the above theorem would take $\Omega(nm^2)$ since there are $O(m^2)$ pairs of columns and testing for the validity of the conditions will take $O(n)$ time for each pair. A more efficient algorithm exists which is described below.

Perfect Phylogeny Algorithm taking $O(nm)$ time

1. Sort matrix M to obtain matrix $\overline{M}$ using radix sort taking $O(nm)$ time.
2. For each row of $\overline{M}$, construct the character strings corresponding to '1' in the row from left to right. Use the characters of original M to write these strings.

There will be *n* such strings each of maximum length *m*. (For our example, these strings are '21', '3','215', '34', '2').

3. Build a 'Keyword Tree *T* for these 'patterns' constructed in step 2. This step takes *O*(*nm*) time.
4. Test whether *T* is a perfect phylogeny (each character appears once and only once in an edge in the tree, the tree is rooted and connected).

If the characters are unordered, one can also develop a *O*(*nm*) algorithm to obtain a perfect phylogeny. (Prove)

Tree Compatibility Problem

The problem is : given two or more phylogenetic trees, do they represent a consistent evolution history and if so how to connect these trees into one phylogenetic tree incorporating all the evolutionary history of the constituent trees. Problems like this arises in handling real data because people use different tree building methods .

Definition: A phylogenetic tree *T'* is a refinement of *T* if *T* can be obtained by a series of contraction of edges of *T'*.

If *T'* refines *T* then *T'* contains all the revolutionary history contained in *T* and it will have additional history not displayed in *T*. Let T_1 and T_2 be two phylogenetic trees on *n* objects in "reduced" form (both binary trees, no node except the root node can have exactly one child).

Definition:Trees T_1 and T_2 are **compatible** if there exists a phylogenetic tree T_3 refining both T_1 and T_2.

Let M_1 and M_2 be matrices corresponding to the given trees T_1 and T_2. Let M_3 be a matrix formed as the union of matrices M_1 and M_2. Then, prove

Theorem: Trees T_1 and T_2 are compatible if and only if there is a phylogenetic tree for M_3. Further, a phylogenetic tree T_3 is a refinement of both T_1 and T_2.

(See Gusfield for example and further discussion).

Generalized Perfect Phylogeny

Problem: Given a character matrix *M* where each character may take upto *r* states, determine if there isa perfect phylogeny for *M* and if so, construct one.

In this case, a perfect phylogeny for *M* isa directed tree where each edge is labeled by an *ordered triple* (*c,x,y*) of *character-state transition* indicating that the character *c* changes from state *x* to state *y* and this triple appears on only one edge of the tree. All other conditions of binary trees remain valid viz. one root, the path from root the node labeled *i* describe the character states of the object *i*.

If r is fixed at 3 or 4 , there is a polynomial bounded solution. If r is fixed and constant, then the solution is again polynomially bounded by n and m with an exponential factor in r. But, if r is variable, the problem has been shown to be NP-cokmplete. (See Gusfield, p.465).

Parsimony

Algorithms with fixed parameters may seem good enough but real practical data does not fit with perfect phylogeny model most of the times. The reasons are that the experimental data always have errors, and convergence and reversals sometimes happen. If we ignore the errors, one way to handle the problem is to minimize convergence and reversal occurrences. This is called maximum *parsimony criterion.* Maximizing parsimony is equivalent to minimizing *mutations* The other approach is not to use those characters in the construction of the tree that cause these problems. This is equivalent to finding a maximum set of characters which allows perfect phylogeny. This is known as the *compatibility criterion.* Use of these criteria lead to *optimization problems* rather than decision problems and remain NP-complete both for ordered and unordered characters. Proofs of NP-completeness will not be discussed here.

Distance-Based Methods

In distance-based approach, the input consists of evolutionary distance data (viz. edit distances of DNA or RNA sequences, melting temperature from DNA hybridization etc.). The problem is to construct a weighted tree whose pair wise distances agree with pair wise input distance data. We will later characterize the data as *ultra-metric* or *additive* and show that for this kind of data trees can be constructed efficiently. Real data do not satisfy these characterizations and approximate algorithms have been proposed for these cases. We will discuss some of them.

Ultrametric Tree

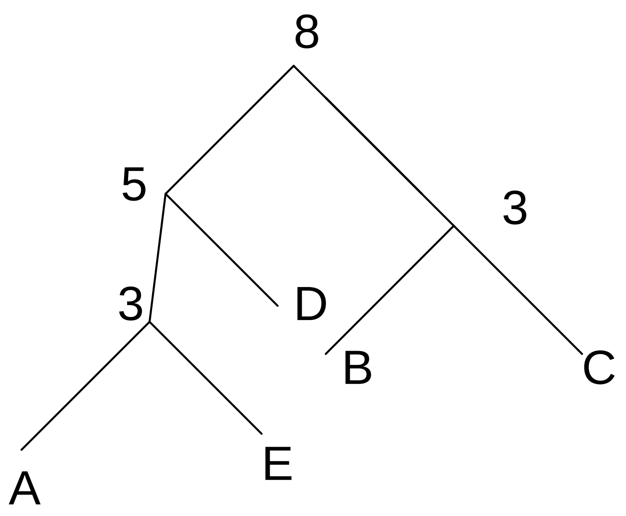
Ultrametric trees have applications in many numerically-based tree construction methods, and can be used to find the branching patterns of evolutionary history and measures of elapsed time among nodes in the tree. Although the input data is a set of numbers, these numbers are usually the output of some string algorithm such as sequence comparisons or pair wise distance data of multiple alignment of a set of sequences.

Definition: Let D be a n by n symmetric matrix of real numbers. An ultrametric tree for D is a rooted tree T with the following properties:

1) T contains n leaves, each labeled by a unique row of D.
2) Each internal node of the tree is labeled by one entry in D and has two children.
3) The numbers labeling internal nodes in the tree along any path from the root to leaf are *strictly decreasing.*
4) For any two leaves i and j of T , $D(i,j)$ is the label of the least common ancestor of i and j in T.

Example:

	A	B	C	D	E
A	0	8	8	5	3
B		0	3	8	8
C			0	8	8
D				0	5
E					0



If the property 3) above is changed to : labels in the internal nodes must strictly increase, then the tree is called a *min-ultrametric tree* (in absence of any better name). Obviously, not all matrices are untra- or min-ultrametric. Since with *n* leaf nodes the tree must have only *n*-1 internal nodes, if the matrix has more than *n*-1 distinct values in *D,* the matrix cannot be untra- or min-ultrametric.

Evolutionary Trees as Ultrametric Trees.

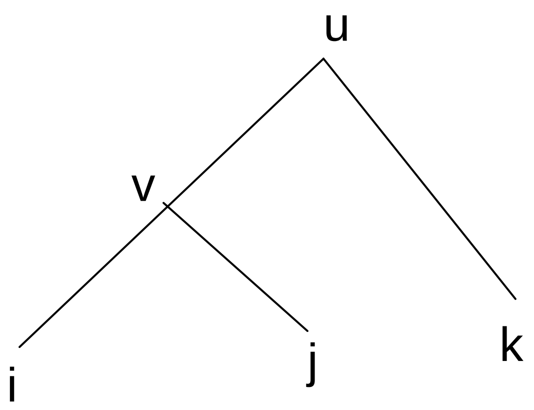
If the evolutionary history of *n* taxa form a rooted directed tree, with extant taxa represented at the leaf nodes, then each internal node can be looked upon as a *divergence event.* A divergent point is a point *v* in time when two taxa p and q diverge. It simply means that before the point *v* the two taxa *p* and *q* shared a common ancestor. If we know the absolute or relative times when the divergences took place and if we append these times at the internal nodes (the branching nodes), these times must be strictly increasing for the nodes in a path from root to a leaf . Further, the label at common ancestor *v* of leaf nodes *p* and *q*, is the time when *p* and *q* diverged. Thus, *T* is a min-ultrametric tree for the *n* by *n* matrix *D* and *D*(*p,q*) is the time that *p* and *q* diverged. On the other hand, if we want to interpret the label at each node to represent *elapsed time* since the divergent *v* happened, then the times must be strictly decreasing in the path from the root to the leaf node and the corresponding matrix becomes ultrametric.

Test for an Ultrametric Tree

Definition: A symmetric matrix *D* of real numbers defines an *ultrametric distance* if and only if for every three indices *i, j* and *k* , there is a tie for the maximum of *D*(*i,j*), *D*(*i, k*) and *D*(*j,k*). Similarly, *D* defines a *min-ultrametric* distance if and only if for every three indices *i, j* and *k* , there is a tie for the miniumum of *D*(*i,j*), *D*(*i, k*) and *D*(*j,k*).

Theorem: A symmetric matrix *D* has an ultrametric tree (or a min-ultrametric tree) if and only if *D* is an ultrametric (or min-ultrametric) matrix.

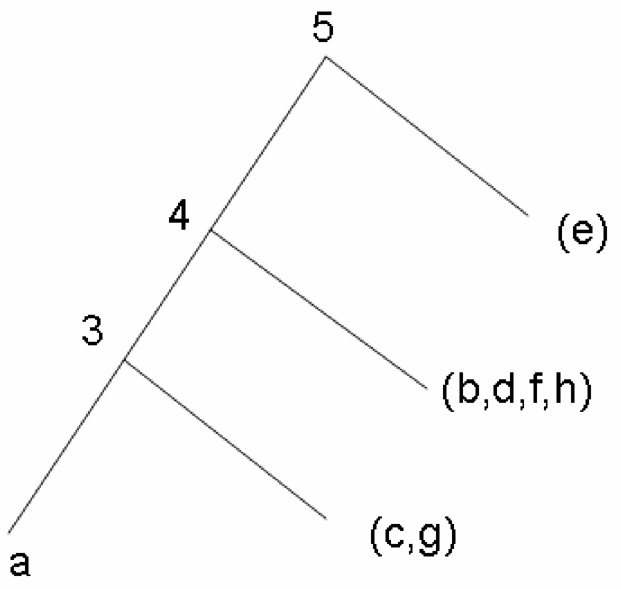
Proof: First, show that if *D* has an ultrametric tree, then *D* is an ultrametric matrix.



Here *v* and *u* are common ancestors of *i* and *j*, and *i* and *k*, respectively. Since it is an ultrametric tree, *u*>*v,* strictly. By definition of *D*, *D*(*i*,*k*)= *D*(*j*,*k*)=maximum. Hence *D* is an ultrametric matrix.

Now, assume *D* is an ultrametric matrix. We have to show that there exists an ultrametric tree. By definition, if in a row *a* of *D*, there are *d* distinct entries, then any ultrametric tree for *D* must have these numbers from the leaf representing the row *a* to the root in strictly increasing order. This also induces a partition of the remaining (besides row a) rows in *d*-1 disjoint partitions, as shown below

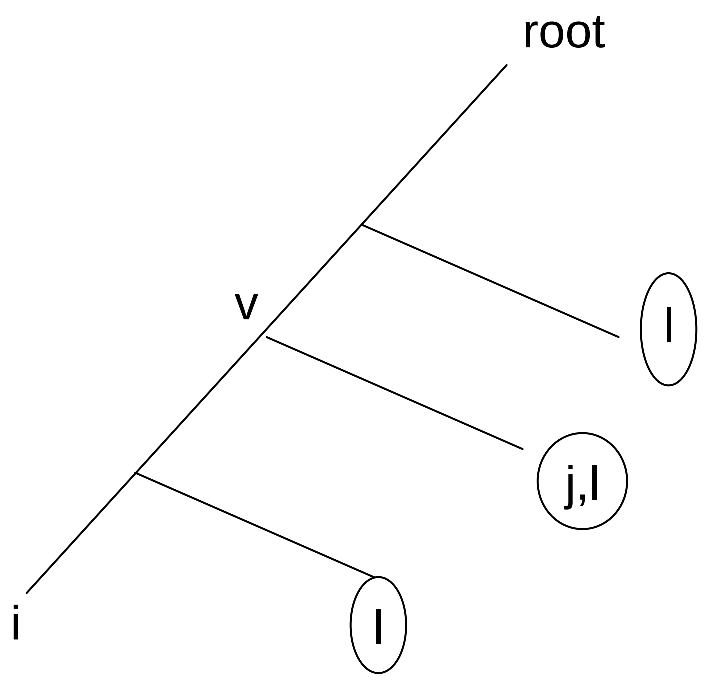
	a	b	c	d	e	f	g	h
a	0	4	3	4	5	4	3	4
b		0	4	2	5	1	4	4
c								
d								
e								
f								
g								
h								



Now, if we can find ultrametric tree for each of these partitions, we can simply attach these trees and get an ultrametric tree for the entire matrix. We would like to apply the procedure recursively. We show now that the approach works correctly.

Consider the class defined by the internal node *v* and assume the leaf node *j* is a member of this class . Let *l* be some other leaf node. We have three cases:

1) *l* is in the same class as that of *j*: This means that $D(i, j)= D(i,l)$. There fore, $D(j,l) \le D(i, j)$ since *D* is ultrametric. If we can now attach an ultrametric subtree containing *j* and *l*, we are done. If $D(j,l) = D(i, j)$, then node v will have degree greater than 2



2) *l* is in a class between the leaf node *i* and node *v*: In this case, $D(i,l)<D(i,j)$ and $D(j,l) = D(i, j)$ and *v* must be the least common ancestor of *j* and *l*. Therefore, if the ultrametric tree containing *j* is connected at *v* , then $D(j,l)$ will be correctly written at the least common ancestor of leaves *j* and *l*.

3) *l* is in a class between *v* and the root node : In this case, $D(i,l)>D(i,j)$ and so $D(j,l) = D(i,l)$. So, if an ultrametric tree for the class containing *j* is connected at node *v* , $D(j,l)$ will be correctly the least common ancestor of *j* and *l*.

This completes the proof.

Additional Notes: http://www.cs.ucdavis.edu/~gusfield/ultraerrat/ultraerrat.html

On page 451, the proof of Theorem 17.1.1 presents an algorithm for building an ultrametric tree. The algorithm is correct, however Theorem 17.1.3 states that the algorithm can be implemented in $O(n^2)$ time. In fact, I don't see how to do that, although $O(n^2 log\ n)$ is an easy bound for the algorithm. Rather than continuing to try to find a clever implementation of that algorithm, here is another combinatorial algorithm that I claim is correct and that does run in $O(n^2)$ time. The algorithm is described in terms of a graph G, based on matrix D, but it can be implemented without an explicit graph.

Let each row i of matrix D be represented by a node i in G, and each edge (i,j) be given the value $D(i,j)$. In $O(n^2)$ time, the algorithm will find a very particular path in graph G:

set N equal to all the indices 1 through n; set L to the empty path; set i to any node.

repeat n-1 times: begin remove i from N; find an index j in N such that
$D(i,j) \leq D(i,k)$
for any k in N. place edge (i,j) in the path L; set i to j; end;

What this produces is a path L of exactly n edges, and the algorithm can be implemented in $O(n^2)$ time. It turns out that L is a minimum spanning tree of G, but that fact is not needed.

We will now use L to create the ultrametric tree recursively.

Concentrate on an edge (p,q) in the path L with the largest edge weight of all edges in L, and let P be the set of nodes at or to the left of p in L, and let Q be the set of nodes at or to the right of q in L. The fact that D is an ultrametric matrix implies that for any pair of nodes (i,j) where i is in P and j is in Q, $D(i,j) = D(p,q)$. One way to prove this is by induction on the number of edges between i and j in L (applying the ultrametric condition that the in any triangle, the max of the three edge weights is not unique). What this means is that in the ultrametric tree we are building (and in any ultrametric tree for D), any pair of leaves (i,j) where i is in P and j is in Q must have their least common ancestor at the root of the ultrametric tree, and that root must be labelled $D(p,q)$.

If there are $k > 1$ ties for the global max edge weight in L, then removing those k edges creates k+1 subpaths of nodes, and applying the above argument, any two nodes i and j which are in different subpaths must have their least common ancestor at the root of the tree, which again must be labeled $D(p,q)$. Hence, any ultrametric tree T for D must have exactly k+1 edges out of D, and the leaf set below any such edge must be exactly the (distinct) set of nodes in one of the k+1 subpaths.

No matter what k is, removing the k max weight edges in L, and partitioning N, takes only $O(n)$ time.

To continue the description of the algorithm, we assume for convenience that $k = 1$. Let LP and LQ denote the two subpaths created by removing the max weight edge in L. Now we want to find an ultrametric tree for set P and one for set Q; these two ultrametric trees will then be attached to the root to creat the full ultrametric tree for D. But note that we already have the needed paths LP and LQ that would be created if we were to recursively apply the above method (clearly LP could result if we applied the path building algorithm to P alone, and similarly for LQ and Q). So we only need to find the max weight edge(s) in LP and the max weight edge(s) in LQ. Those two edges can be found in $O(n)$ total time. Again, because the nodes were partitioned in the first step, this time bound holds even for $k > 1$.

Continuing, we build the ultrametric tree in $O(n^2)$ total time.

Note that at each step of the algorithm, the node partitions that are created, and the associated edges that are put into T, are forced. Hence if D is an ultrametric matrix, the ultrametric tree T for D is unique.

Additive Distance Tree

If the data giving time-since-divergence is correct, the ultrametric tree gives the true evolutionary history. But, in practice, data is rarely ultrametric. This is handled by imposing a weaker requirement on the evolutionary data, that is, data is *additive*.

(To be continued)