20.181 Lecture 4

Contents

Review of Homework #1

1. Fibonacci question
   - Using recursion is good with Fibonacci if you're returning (and passing) a list.
2. Parsimony:
   - Review of how to assess the most parsimonious explanation of nodes' states.
   - Solution/notes on the Parsimony question.

Parsimony Continued

- Why parsimony?
  - we can suggest infinite alternative explanations involving increasing numbers of mutations but they become less and less plausible
  - this algorithm is sometimes called "model-free" because the model is implicit
  - is parsimony better than just an extreme case of maximum likelihood? This is still debated.

- One **not**-optimal way to look for the most parsimonious tree: try all four possibilities at every node. But that would require looking at 4^n possibilities! Using recursion in this problem, we can avoid that.

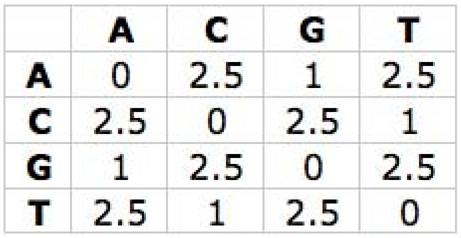
Down-Pass Algorithm

- Formalizing the idea of what you did by intuition in HW1.

```
def downPass(tree): #pseudocode!
      if tree is a leaf:
      return seq
      l = downPass(left child)
      r = downPass(right child)
      i= l <intersect> r
      if i==None:
      return l <union> r
      return i
```

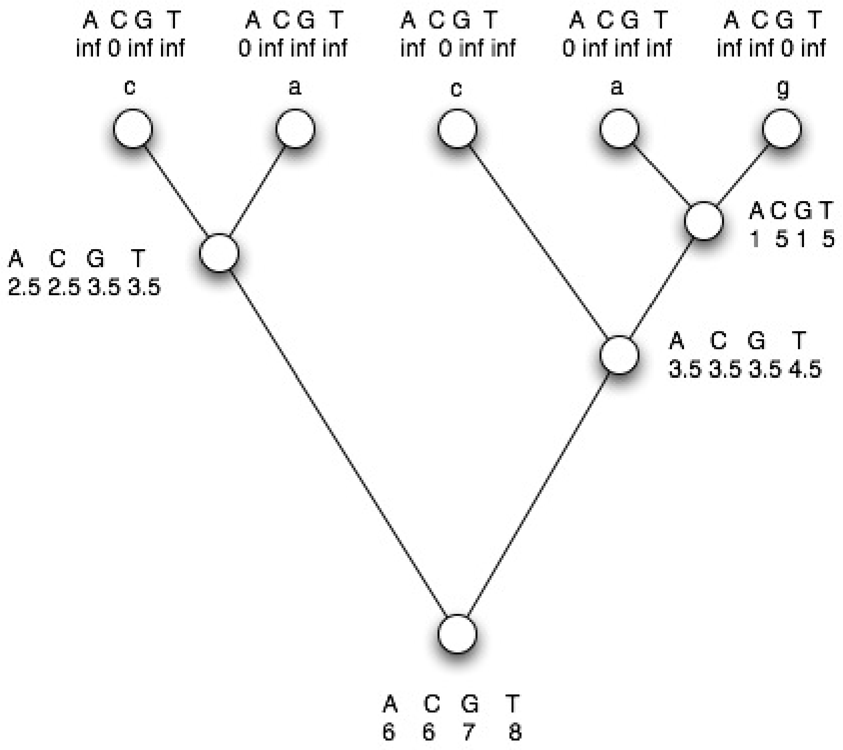
Sankoff Downpass Algorithm

1. Consider different weights for different mutations: P(transitions) > P(transversions)



	A	C	G	T
A	0	2.5	1	2.5
C	2.5	0	2.5	1
G	1	2.5	0	2.5
T	2.5	1	2.5	0

   1.
      - The cost of no mutation will be **zero**.
      - The cost of a transition we'll set to **1**.
      - The cost of a transversion will be **2.5** times as much as a transition.
2. Each leaf is associated with a vector of length 4 which stores the cost (so far) of that node being A, C, G or T
3. Initialize tree
4. On the leaves we want to do something different from the internal nodes, because we are 100% sure of what the sequence is; it's not a guess.
   - put a score of zero for the letter that we KNOW it is
   - put a score of infinite for the other three.
5. Work down from leaves to root. At each internal node, each entry in the vector will store the cost of the mostly likely sequence of mutation events required in its subtree to get to the that particular internal node state (A,C,G, or T).



Calculating the cost vector at the first level of internal nodes should be straightforward. At the next level (the right child of the root node) consider the possibility that the ancestor had "A" at this position. There are four ways to get there, but we only care about which was the most likely! First consider the costs along the two branches that attach at this node:

1. The cost along the left branch is easy to find: we know its left child is C, so the cost of mutation from C to A is 2.5
2. The cost along the right branch depends on the identity of the right child. If the right child were (A,C,G,T) that would entail additional costs of mutation (0,2.5,1,2.5), respectively.
3. Additionally we have to accrue the costs of each possibility at the right child node: (1,5,1,5)
4. Therefore our four possible paths have cost:
   - (2.5,2.5,2.5,2.5) + (0,2.5,1,2.5) + (1,5,1,5) = (3.5,10,4.5,10)
5. The path of least cost to get to an "A" at this node is 3.5. That becomes the first entry in our vector for the root's right child node. This process must be iterated for C,G,T at this node.
6. A counter-intuitive observation: we get the right answers at the root of the tree, but not at the other internal nodes. What is different about them?