Project #1

Project 1: Implement suffix trees

Handed out: 10/5/06
Due: 10/26/06

For this project you will have to write software that builds a suffix tree from a DNA string (T) provided in the input, outputs the suffix tree you constructed, and identifies the longest substring of a second DNA string (P) that matches a substring of T.

Deliverables: Source code
Requirements:

1. The source code compiles and executes on a Linux machine (Mac OSX or cygwin under Windows is OK). IMPORTANT: if your code doesn't compile or run as "advertised" I will not try to debug it and will consider the project failed.
2. The running time of the program is linear in size of input, both for constructing the tree and for executing the search. By this, I mean that if I try several inputs of increasing sizes, the running time grows proportionally to the size of the inputs (that's what suffix trees are supposedly good at :)).
3. The output is correct (should go without saying).
4. You also provide me with a user manual (< 1 page) that explains how to install and run your program.
5. The source code is readable: formatted nicely, and contains comments.

Grading: Maximum grade is 100 points. Each of requirements 2, 4, and 5 are worth 10 points. Requirement 3 is worth 20 points (and includes both finding the match, and the correctness of the tree output by the program). Failure to achieve requirement 1 results in an automatic score of 40 for this project.

Note: you can use any programming language that allows you to achieve these requirements.

Details

Input formats
The inputs will be provided in FASTA format (see http://en.wikipedia.org/wiki/Fasta_format) with the following assumptions:

- The header line contains one single identifier immediately following the ">" character.
- The DNA sequence contains only characters A, C, T, and G (alphabet size = 4).
- Each file (pattern and text) contains exactly one sequence.

Here is an example:

```
>Text
ACAGGTAGCAGGGAC
CATGACCAGGGCTGC
GAC
```

Output format
The output of your program should consist of a line like this:
<TextID> <PatternID> <l_text> <r_text> <l_pattern> <r_pattern>

where <TextID> and <PatternID> are the header lines for the text and pattern FASTA files respectively, and l_*, r_* are the left and right coordinates of the longest alignment in the text and the pattern.

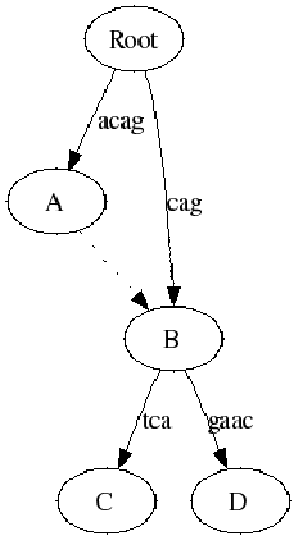
These fields should be separated by TAB characters ("\t" in most programming languages).

Printing the tree
Printing the tree to the output should be left as an option to the user, with the default that the tree is not printed, unless the user specifies the name of a file where the "picture" of the tree will be placed.

To print the tree, use the Graphviz graph format (http://www.graphviz.org). Here is a simple example:

```
digraph Tree {
     Root -> A [label = "acag"]
     Root -> B [label = "cag"]
     B -> C [label = "tca"]
     B -> D [label = "gaac"]
     A -> B [style = dotted, weight = 0]
}
```

which results in the tree shown below (note that suffix links are represented as dotted lines, and given a weight of 0 so they don't mess up the layout of the tree).



The only program you really need from the Graphviz package is “dot”. If the tree is written to file tree.dot, you can use the dot program to build a pretty picture using the command:

```
dot -Tps -o tree.ps tree.dot          - generates a Postscript image
dot -Tgif -o tree.gif tree.dot        - generates a GIF image
```

Please let me know if you need help finding the dot program for the operating system you commonly use.

Final comments

Please start work on this project early. Theory has a knack for failing in practice, so even if you think you’ve got it all figured out there will be many practical annoyances that will slow you down.

Please contact me if you have any questions regarding this project, its scope, my requirements, etc. Once you turn in the project it will be too late to argue, for example, what “commenting the code” means.

GOOD LUCK