1.

(1)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Remaining | End | Active node | Active edge | Active length | Illustration of end of current phase |
| 0 | 0 | root | -1 | 0 |  |
| 0 | 1 | root | -1 | 0 |  |
| 0 | 2 | root | -1 | 0 |  |
| 1 | 3 | Root | 0 (This is my answer for this phase. Not -1) | 1 |  |
| 1 | 4 | Root | 0 | 1 |  |
| 2 | 5 | A | 1 | 1 |  |
| 3 | 6 | A | 1 | 2 |  |
| 0 | 7 | root | 3 | 0 |  |

AP = active point

Source:

https://www.youtube.com/watch?v=aPRqocoBsFQ&t=3069s

(2) Because any string is also a suffix of itself, we know for sure that for any true suffix tree, the original string should be contained in it. The length of every suffix in the tree should be less than or equal to the size of the original string. Therefore, if we can find the longest branch doing some index calculations, then we can arrive at the tip of the longest branch in linear time.

2.

(1)

For every non-matching characters, a replace is answer coming from diagonal, addition is answer coming from right, and removal is answer coming from above.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | ‘’ | C | T | G | C | C | T | G | A |
| ‘’ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| C | 1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| G | 2 | 1 | 1 | 1 | 2 | 3 | 4 | 5 | 6 |
| T | 3 | 2 | 1 | 2 | 2 | 3 | 3 | 4 | 5 |
| C | 4 | 3 | 2 | 2 | 2 | 2 | 3 | 4 | 5 |
| A | 5 | 4 | 3 | 3 | 3 | 3 | 3 | 4 | 4 |

(2)

The number of distinct optimal alignments is the total number of different tracebacks from the answer derived to the starting position. For a 2D array dynamic programming algorithm, this could mean starting from the bottom right hand corner of the array and backtrack all the different paths that lead back to the top left hand corner. An example is shown on page 222 of Algorithms on strings, trees, and sequences by Dan Gusfield. Let D (i,j) denote the minimum total weight for edit opeations transforming S1[1..i] to S2[1..j]. Let t(i,j) handle both replacement and equality, where now t(i,j) = e if S1(i) = S2(j); otherwise t(i,j) = r. Then the base condtions are D(i,0) = i x d and D(0,j) = j x d

The general recursive function is:

D(i,j) = Min [D(i,j-1) + d, D(i-1,j) + d, D(i-1,j-1) + t(i,j)]

This recursive definition is what marks the track for later if we need to back track to count the distinct alignments.

(3)

We can use Bafna’s dynamic programming algorithm with a running time of O(n4) using a binarized tree for the structure S. For the RNA structure S, we can visualize the non-crossing base pairs by adding dashed edges to S, so that each node in S’ has at most two children. The dashed edges fix certain interval doing recursions, when we need to find the branch points. The procedure is shown below:

Binarize(i,j):

begin

for 1 <= u < k do

Binarize(iu,ju)

S’ = S’ unioned with {(i1,ju)}

if (u > 1)

parent((i1, ju-1)) = (i1,ju)

parent((i1, ju)) = (i1,ju)

parent(i1,jk) = (i,j)

end

(4)

Richard E. Bellman.

He invented the name “dynamic programming” to hide the fact that he was doing

mathematical research at RAND under a Secretary of Defense who “had a pathological fear and hatred of the term, research.” He settled on “dynamic programming” because it would be difficult give it a “pejorative meaning” and because “It was something not even a Congressman could object to.”

Sources:

https://editorialexpress.com/jrust/research/papers/dp.pdf

https://press.princeton.edu/books/paperback/9780691146683/dynamic-programming