Gaps between homologous proteins are often induced by adaptive mutations. As a result, context-dependent (instead of constant) gap penalty functions are introduced to accommodate various frequency of gap opening and extension at each amino acid residues. Modify the dynamic programming of global alignment of two protein sequences, S = s1,s2...sm and T = t1,t2…tn, assuming the open gap penalty u(si) and the extension gap penalty v(si) for the gap after the amino acid residue si in sequence s, and the open gap penalty u(ti) and the extension gap penalty v(ti) for the gap after the amino acid residue ti in sequence t.

For global alignment we make three recursive calls. One to the left diagonal (i-1,j-1), one to the above (i-1,j) and, one to the left (i,j-1) and we grab the max of the those results for every call. For the top and left calls, we add the gap penalty values to the calls. For the left diagonal call, we add a value if it’s the characters are a match, otherwise we add a penalty for mismatch. The pseudocode is shown below:

[ si-1,j-1 + δ (vi, wj) ]

si,j = MAX [ si-1,j + δ (vi, -) ]

[si,j-1 + δ (-, wj) ]

As mentioned in the slides, the “-“ represents a gap character and that the time complexity increases from O(n2) to O(n3).

2)

To evaluate the statistical significance of certain properties in DNA sequences, generate many random shuffled DNA sequence instances from one or more given DNA sequences. A simple swapping-based algorithm can generate random sequences preserving the same nucleotide frequencies as the input sequences. In some applications, however, it is also expected to preserve the same k-mer (k > 1) frequencies in the shuffled sequences as the input sequences. For example, for k = 2, we expect the 4 x 4 = 16 dinucleotide frequencies remain the same as input sequences. Given a sequence, devise a linear-time sequence shuffling algorithm to generate a precisely uniform instance (randomly) that preserve the k-mer frequencies. Note that you need discuss the randomness of your algorithm and you need use “rand()” to output a random integer between 0 and RAND MAX in your algorithm.

One possible approach is to create a De Bruijn graph from the given DNA sequence, find all possible string sequences from the graph, and then pick a random string from all the generated ones. For example, let sequence S = AACCGGTTAA and k = 3. We need the 2-mers from S to create our nodes from the graph.

The 3-mers from S are {AAC, ACC, CCG, CGG, GGT, GTT, TTA, TAA}

Our 2-mers from S are {AA, AC, CC, CG, GG, GT, TT, TA}

The graph along with all the possible strings is shown below:

Diagram

Description automatically generated

Once all the possible strings are found we can store them in an array use the rand() function to generate a random index between 0 and the length of the array, n, and use it to pick a random string from the array. The time complexity of this algorithm to find each possible strings is the number of k-mers is linear because we traverse thru every edge exactly once. In this example the overall graph forms a cycle so we can start from every node and since there are 8 possible nodes to start from in this graph, the time is O(8n) => O(n).