CSC 2417 Algorithms for Genome Analysis PS1
Due February 13th in class or by email before class start time

Don't Panic

This is an individual assignment. While you may discuss this assignment with classmates, please do not give away answers. You are NOT allowed to use the internet, besides to look up things not directly related to the assignment, such as a generic formula or a well-known algorithm. This homework may have bugs. If you spot something that looks wrong or is not clear please contact me or post to the newsgroup.

1. Assembly & Overlaps

You are assembling a genome (DNA) using the de bruijn graph approach using k-mers. K is sufficiently small that each (k+1)-mer is actually present in the genome. Remember, DNA is bi-directional, and recall we defined de Bruijn graphs as having k-mer nodes (k+1) mer edges.

(a) Draw the resulting graph for k=2.

(b) How many nodes and edges will your graph have for arbitrary k?

(c) Assume that some genome has length G, and that N reads, each of length L, will be assembled to get this genome. The coverage of the genome is NL/G. Assume that G is much larger than L. The reads are assumed to be taken at random from the genome so that, ignoring end-effects, the position of the left-hand end of any read is uniformly distributed in [1,G]. A contig is a maximal covered subsequence in the genome.
i. What is the mean proportion of the genome covered by contigs? How much should the coverage be for 99% of the genome to be covered? How much should the coverage be for 99.9% of the genome to be covered?
ii. What is the mean number of contigs, in terms of N,L and G?
iii. What is the mean contig size?

(d) Finding overlaps:
Because in reality sequenced reads can have sequencing errors, we must allow for non-exact matches between reads. We may, for example, say that two reads overlap if they fewer than D discrepancies (substitutions, insertion or deletions, not counting any initial and final gap). This is typically done using a simple variation on the standard Needleman--Wunsch algorithm where we do not penalize the initial or final gap, with running time $O(n^2)$ for two reads of length n. Demonstrate an algorithm that is asymptotically faster than this approach for a small constant D.

2. Alignment Variations

(a) The score of a local alignment is not normalized over the length of the matching region. As a result, a local alignment with score 100 and length 100 will be chosen over a local alignment with score 99 and length 10, although the latter one is probably more important biologically. To reflect the length of the local alignment in scoring, the score

F(I,J) of local alignment involving substrings I and J may be adjusted by dividing F(I,J) by the total length of the aligned regions: F(I,J)/(|I|+|J|). The *normalized local alignment problem* is to find substrings I and J that maximize F(I,J)/(|I|+|J|) among all substrings I and J with |I|+|J| >= k, where k is a threshold for the minimum overall length of I and J. Devise an algorithm for solving the normalized local alignment problem.

(b) Two sequences of length n & m may have more than one optimal alignment. Give an algorithm to compute the number of optimal alignments between two such strings in O(mn) time.

(c) Since the biological significance of the optimal alignment is sometimes uncertain, and optimality depends on the choice of (often disputed) weights, it is useful to efficiently produce or study a set of *suboptimal* (but close) alignments in addition to the optimal one. Given two strings X and Y (of lengths n and m) and a parameter δ, show how to construct the following matrix in O(nm) time: M(i,j) = 1 if and only if there is an alignment of X and Y in which characters X[i] and Y[j] are aligned with each other and the value of the alignment is within δ of the maximum value alignment of X and Y. That is, if F(n,m) is the is the value of the optimal alignment, then the best alignment that puts X[i] opposite Y[j] should have value at least F(n,m) – δ.

(d) Bacterial DNA is often organized into circular molecules. Given two strings x and y of length n and m, respectively, there are n circular shifts of x, and m circular shifts of y; therefore, there are nm pairs of circular shifts.

Build an efficient algorithm to find the best global alignment among the nm different pairs of circular shifts. The trivial $O(n^2m^2)$ algorithm is not good enough. A cubic time solution (i.e. $O(n^2m)$) will only get partial credit.

(e) In the linear space alignment, the original problem of size mn is reduced to two sub-problems of sizes km/2 and (n-k)m/2. In a fast, parallel implementation of sequence alignment, it is desirable to have a *balanced partitioning* that breaks the original problem into sub-problems of equal sizes. Design a linear space alignment algorithm with balanced partitioning.

3. Longest & Shortest Paths

In alignments, it is common to give matches a positive score, and substitutions and gaps a negative score (penalty). If one converts the cells of the alignment matrix to nodes, and adjacencies to edges, the optimal alignment corresponds to the longest path in the graph. It is possible to convert the longest path problem to a shortest path by negating all of the scores (edge lengths). However it is often preferable to not have any negative edge lengths when computing shortest paths. Show how to modify the edge weights so that the best alignment corresponds to the shortest path but all of the edge lengths are positive.