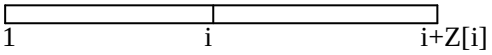
Homework 1 answers:

1. For each of the n prefixes of P, we want to know whether prefix P[1..i] is a periodic string. That is, for each i we want to know the largest k > 1 (if there is one) s.t. P[1..i] can be written as a^k for some string a. Of course, we also want to know the period. Show how to determine this for all n prefixes in linear time in the length of P.

Use Z values.

Case1



Z[i] is the period

Case2

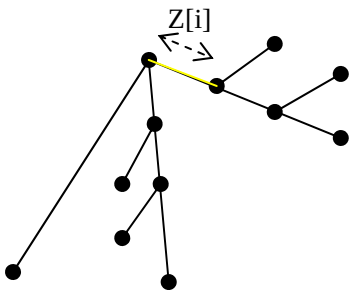
$Z[i] - i > \frac{Z[i]}{2}$

i is the period

2. Given two strings of equal length A and B, describe a linear time algorithm that can determine if one of the strings is a circular rotation of the other string.

Assume
A = PS, B = SP, where P is the prefix string and S is the suffix string.
BB = SPSP, use KMP or any linear exact match algorithm to find whether A is in BB.

3. Describe an algorithm that uses a suffix tree to compute the sp_i values used by the KMP algorithm.
Hint: Match on the suffix tree



From Z value, we can get sp_i value.

4. Given two input strings S1 and S2 and a parameter k, a k-cover C is a set of substrings of S1, each of length k or greater, such that S2 can be expressed as the concatenation of the substrings of S1 in some order. Note that the substrings can overlap in S1 but not in S2. Give a linear-time algorithm to find a k-cover from two strings, or to determine that no such cover exists.

Let m(i) be the matching statistic for string S2, i.e. the length of the longest match starting at position i between S2 and a substring of S1. For simplicity I'll use m(i) to also mean the substring that matches between S2 and S1.

First note that m(i + 1) >= m(i) - 1 as it is the second suffix of the string m(i). Also note that if any m(i) < k no k-cover is possible as this implies that some portion of S2 does not match at least k consecutive characters in S1 (note - the last k m(i)s violate this property)

Given these observations, the algorithm is fairly simple. Let j be the value where the current piece of the cover started, then find the next value to chain as follows: Start walking through the m(i) values for i>j, keeping track of how far in the future the value extends (i + m(i)) and find the first i that satisfies the following properties: (i) i + m(i) > j + m(j); and (ii) i - j >= k. Now switch to the interval represented by m(i) and continue as before until i + m(i) extends to the end of the string.

Continue on Dynamic Programming.

ACTAA-CT
S D I
AG-AATCT

The recursion expression of the edit distance:

$$E[i,j] = \min\{ \begin{cases} E[i,j] + 1 \text{ if } S1[i] != S2[j] \text{ Replacement} \\ E[i,j-1] + 1 \text{ Deletion,} \\ E[i-1,j] + 1 \text{ Insertion} \end{cases} \}$$

Concepts of PAM and BLOSUM(from wikipedia)

PAM

One of the first amino acid substitution matrices, the PAM *(Point Accepted Mutation)* matrix was developed by Margaret Dayhoff in the 1970s. This matrix is calculated by observing the differences in closely related proteins. The PAM1 matrix estimates what rate of substitution would be expected if 1% of the amino acids had changed. The PAM1 matrix is used as the basis for calculating other matrices by assuming that repeated mutations would follow the same pattern as those in the PAM1 matrix, and multiple substitutions can occur at the same site. Using this logic,

Dayhoff derived matrices as high as PAM250. Usually the PAM 30 and the PAM70 are used. A matrix for divergent sequences can be calculated from a matrix for closely related sequences by taking the second matrix to a power. For instance, we can roughly approximate the WIKI2 matrix from the WIKI1 matrix by saying $W_2 = W_1^2$ where W_1 is WIKI1 and W_2 is WIKI2. This is how the PAM250 matrix is calculated.

BLOSUM

Dayhoff's methodology of comparing closely related species turned out not to work very well for aligning evolutionarily divergent sequences. Sequence changes over long evolutionary time scales are not well approximated by compounding small changes that occur over short time scales. The BLOSUM (BLOck SUbstitution Matrix) series of matrices rectifies this problem. Henikoff and Henikoff constructed these matrices using multiple alignments of evolutionarily divergent proteins. The probabilities used in the matrix calculation are computed by looking at "blocks" of conserved sequences found in multiple protein alignments. These conserved sequences are assumed to be of functional importance within related proteins. To reduce bias from closely related sequences, segments in a block with a sequence identity above a certain threshold were clustered giving weight 1 to each such cluster (Henikoff and Henikoff). For the BLOSUM62 matrix, this threshold was set at 62%. Pairs frequencies were then counted between clusters, hence pairs were only counted between segments less than 62% identical. One would use a higher numbered BLOSUM matrix for aligning two closely related sequences and a lower number for more divergent sequences. It turns out that the BLOSUM62 matrix does an excellent job detecting similarities in distant sequences, and this is the matrix used by default in most recent alignment applications such as BLAST.

Differences between PAM and BLOSUM

1. PAM matrices are based on an explicit evolutionary model (i.e. replacements are counted on the branches of a phylogenetic tree), whereas the BLOSUM matrices are based on an implicit model of evolution.
2. The PAM matrices are based on mutations observed throughout a global alignment, this includes both highly conserved and highly mutable regions. The BLOSUM matrices are based only on highly conserved regions in series of alignments forbidden to contain gaps.

3. The method used to count the replacements is different: unlike the PAM matrix, the BLOSUM procedure uses groups of sequences within which not all mutations are counted the same.
4. Higher numbers in the PAM matrix naming scheme denote larger evolutionary distance, while larger numbers in the BLOSUM matrix naming scheme denote higher sequence similarity and therefore smaller evolutionary distance. Example: PAM150 is used for more distant sequences than PAM100; BLOSUM62 is used for closer sequences than Blosum50.

Considering that n gap (deletion) in a row has cost function

$$\mathbf{g(n) = g_o + g_e n}$$

, how we can calculate the edit distance respectively?

We can rewrite the recursion expression of the edit distance:

$E[i,j]$ =min {

$$E[i,j] + 1 \text{ if } S1[i] != S2[j] \text{ Replacement}$$

$$\min_{k<j} \{ E[i, j-k] + g(k)\} \text{ Deletion,}$$

$$E[i-1, j] + 1 \text{ Insertion}$$

}

Matrix D is the scores of alignment ending gaps in S1
Matrix F is the scores of alignment ending gaps in S2
Matrix G is the scores of alignment on the diagonal
Matrix E is the scores of alignment
E = min(D,F,G)

$$D[i,j] = \min\{D[i,j-1] + g_e, E[i,j-1] + g_e + g_o\}$$