1 Biology

DNA - ATCG - Written 5' to 3'. Humans have 3G base pairs, 23 chromosome, avg gene is about 1K-2Kbp

RNA - UAGC - Single stranded. Introns are bits that ultimately get cut out of mRNA, begins with AG ends with GT.

Central Dogma - Transcription: DNA → RNA, Translation: RNA → protein, Post Translation Modification: clean up protein

	T	C	A	G	
T	TTT Phe [F]	TCT Ser [S]	TAT Tyr [Y]	TGT Cys [C]	T
	TTC Phe [F]	TCC Ser [S]	TAC Tyr [Y]	TGC Cys [C]	C
	TTA Leu [L]	TCA Ser [S]	TAA Ter [end]	TGA Ter [end]	A
	TTG Leu [L]	TCG Ser [S]	TAG Ter [end]	TGG Trp [W]	G
C	CTT Leu [L]	CCT Pro [P]	CAT His [H]	CGT Arg [R]	T
	CTC Leu [L]	CCC Pro [P]	CAC His [H]	CGC Arg [R]	C
	CTA Leu [L]	CCA Pro [P]	CAA Gln [Q]	CGA Arg [R]	A
	CTG Leu [L]	CCG Pro [P]	CAG Gln [Q]	CGG Arg [R]	G
A	ATT Ile [I]	ACT Thr [T]	AAT Asn [N]	AGT Ser [S]	T
	ATC Ile [I]	ACC Thr [T]	AAC Asn [N]	AGC Ser [S]	C
	ATA Ile [I]	ACA Thr [T]	AAA Lys [K]	AGA Arg [R]	A
	ATG Met [M]	ACG Thr [T]	AAG Lys [K]	AGG Arg [R]	G
G	GTT Val [V]	GCT Ala [A]	GAT Asp [D]	GGT Gly [G]	T
	GTC Val [V]	GCC Ala [A]	GAC Asp [D]	GGC Gly [G]	C
	GTA Val [V]	GCA Ala [A]	GAA Glu [E]	GGA Gly [G]	A
	GTG Val [V]	GCG Ala [A]	GAG Glu [E]	GGG Gly [G]	G

SNP - single nucleotide polymorphism - ie. mutation.

Tools - Restriction Enzymes: EcoRi digests protein and breaks it at a specific point. Adds sticky ends - Shotgun: randomly break DNA - PCR: polymerase chain reaction. - DNA Array: put specific target sequence in a cell, if DNA has it it reacts so it is a way to see if DNA has genes.

2 Z-Algorithm

Solves exact pattern matching algorithm.

```
1:  function Z-Algorithm(S)
2:      l, r = 0
3:      for k = 2 .. n do
4:          if k > r then
5:              find z_k by comparing S[k .. n] and S[1 .. n]
6:              If z_k > 0, r = k + z_k − 1, l = k
7:          else
8:              k' = k − l + 1
9:              β = r − k + 1
10:             If z_{k'} < β: z_k = z_{k'}
11:             if z'_k ≥ β then
12:                 compare S[β + 1..n] with S[r + 1..n] until mismatch at q ≥ r + 1
13:                 z_k = q − k
14:                 l = k
15:                 r = q − 1
16:             end if
17:         end if
18:     end for
19:     return Z
20: end function
```

This solves prefix matching problem returns an array of the matches. For exact pattern matching do $S + \$ + T$ and return locations where score $= |S|$ after the \$ position. This runs in $O(n)$ time, or $O(|S| + |T|)$ time for the exact pattern matching problem.

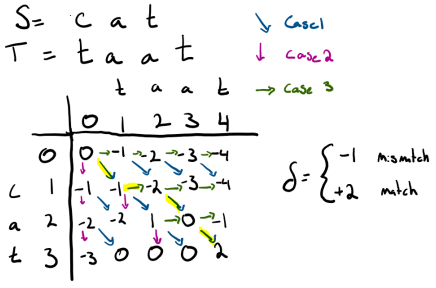
3 Y-Algorithm

Solves multiset matching problem. Goal is to check if all the letters in a pattern appear consecutively, without regard to order. The intuition of the program is to initialize C to store counts of each letter in pattern. Then we slide a window along and adjust counts. We track a left and right end of the box of matches found so far in order to reduce redundant checks. If all the counts are 0, then we have a match. This runs in $O(n)$ time

4 Global Alignment

The goal is to find the optimal alignment of two strings. This can be edit distance: where $\delta(x, x) = 0$, $\delta(x, y) = -1$ Which essentially counts the number of differences in the string. Or it can be a more complicated metric. The global alignment score problem has $\delta(x, x) = 2$, $\delta(x, y) = -1$ Running time is $O(nm)$

```
1: function Needleman-Wunch(S, T)
2:     V = n x m matrix.
3:     V[0, 0] = 0, V[0, j] = V[0, j − 1] + δ(_, T[j]), V[i, 0] = V[i − 1, 0] + δ(S[i], _)
4:     Loop over rows/cols
5:     V[i, j] = max { V[i − 1, j − 1] + δ(S[i], T[j])
                     { V[i − 1, j] + δ(S[i], _)
                     { V[i, j − 1] + δ(_, T[j])
6:     return V[n, m]
7: end function
```



4.1 Four Russians Speedup

Four Russians does same as Needleman Wunsch. But computes in t x t blocks. The idea is to compute just the perimeter of the block. Optimal value of $t =$

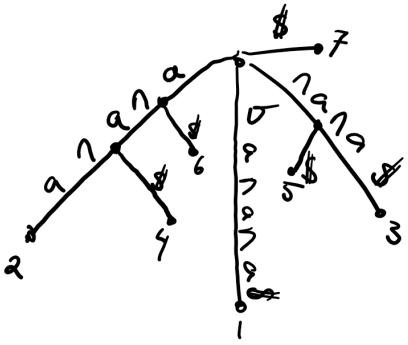
$\frac{\log_{3|\Sigma|} n}{2}$. Edit distance can be computed in $O(n^2/\log n)$

5 Local Alignment

Goal is to find optimal alignment of any substrings within the words. This is Smith-Waterman algorithm. It is almost identical to the Needleman Wunsch, but with a case 0 added. $V[i,j] = max \begin{cases} 0 \\ V[i-1,j-1] + \delta(S[i], T[j]) \\ V[i-1,j] + \delta(S[i], \_) \\ V[i,j-1] + \delta(\_, T[j]) \end{cases}$

6 Suffix Trees

Build a trie with all the possible suffix's of an input. Typically append an extra special character to the end \$ to mark the end of a string. This can be used in a generalized suffix tree to identify which bits belong to which input based on unique end character. To save space, store the letters as index slices into the string instead of raw characters. Naiive way to build takes $O(n^2)$, but can be done in $O(n)$.



Exact pattern matching queries can be done quite fast $O(m + \#occ)$. Just follow pattern down suffix tree. Return number of nodes below the pattern.

Longest common substring. Use a generalized suffix tree (put both S and T in). Use DFS to label height of internal nodes, and to label which strings can be reached from the node. Longest common substring is the depth of deepest internal node from which both strings are reachable.

Longest common prefix. Similar concept as LCS. LCP(i, j) = LCP of $S[i..n]$ and $S[j..n]$. So find leaf nodes corresponding to i, j. Then LCP is the depth of lowest common ancestor. (lowest internal node from which node i and j can be reached)

Palindrome algorithm. Kind of complicated. Goal is to find longest palindrom centered around i for all i. Build generalized suffix tree for S, S^{rev}. Find depth of all internal nodes. Build magic datastructure that allows constant time lookup to compute Lowest Common Ancestor (LCA) (this is for LCP computations) For even palindromes: Compute $k = LCP(S_i, S^{rev}_{n-i+2})$, report $S[i-k..i+k-1]$. For odd palindromes: Compute $k = LCP(S_i, S^{rev}_{n-i+1})$, report $S[i-k+1..i+k-1]$.

7 Suffix Array

Create all suffix of string S. Sort by lexicographical order. At i^{th} spot in the array, put the start position of the i^{th} suffix in lexicographical order.