Documentation

C Program that, given 2 strings, computes their best global alignments using BLOSUM62 as score substitution matrix. The algorithm has time complexity equal to $O(NK)$ where N is the length of the shortest string in input and K is the number of indel in the best alignment.

Input Format

Four arguments: `n`, `s`, `m` and `t` separated by a space:

- `n` is the length of the first string.
- `s` is the first string, its characters must in `{A,R,N,D,C,Q,E,G,H,I,L,K,M,F,P,S,T,W,Y,V,B,Z,X}`.
- `m` is the length of the second string.
- `t` is the second string, its characters must in `{A,R,N,D,C,Q,E,G,H,I,L,K,M,F,P,S,T,W,Y,V,B,Z,X}`.

Output Format

The score of the best global alignment and the 2 strings that produce it. **INDEL** charcater is represented using '_'.

Implementation

- Function `band_align(char *s, char *t, int band)`:
  This function computes the score of the best global alignment and the 2 strings that produce it using at most `band` **INDEL**. First of all let's define the solution for this problem without caring about tha band, in this case we can define a matrix `NoBandDp[N + 1][M + 1]` where `NoBandDp[i][j]` contains the best score for the prefixes S_i and T_j for each $i \in 0, ..., N$ and $j \in 0, ..., M$. Now, for the implementation of `band_align()` function note how we are intrested only in the alignments which contain at most `band` indels and that, startring from the top-left corner and going towards the bottom-right corner of the matrix adding an INDEL means moving down or right, while not using and indel means moving

diagonally(down and right). It means that instead of using the complete matrix like how we did for `NoBandDp[][]` we can only store the $N * (2 * band + 1)$ cells that we are interested in (for each i of the N rows we keep the $2 * band + 1$ contiguous cells with the cell (i, i) as its center. Indeed we can define the matrix $dp[N][band * 2 + 1]$ and compute its cells in $O(N * band)$ through a bottom-up process. The implementation result simple if we consider the cell (i, j) in `dp[][]` as the cell $(i, j - i + band)$ in the "original matrix".
For example with this original matrix and $band = 1$ we are intersted in the upper-case cells.

$$\begin{matrix} X_{0,0} & X_{0,1} & x_{0,2} & x_{0,3} & x_{0,4} \\ X_{1,0} & X_{1,1} & X_{1,2} & x_{1,3} & x_{1,4} \\ x_{2,0} & X_{2,1} & X_{2,2} & X_{2,3} & x_{2,4} \\ x_{3,0} & x_{3,1} & X_{3,2} & X_{3,3} & X_{3,4} \\ x_{4,0} & x_{4,1} & x_{4,2} & X_{4,3} & X_{4,4} \end{matrix}$$

And we store them like this:

$$\begin{matrix} / & X_{0,0} & X_{0,1} \\ X_{1,0} & X_{1,1} & X_{1,2} \\ X_{2,1} & X_{2,2} & X_{2,3} \\ X_{3,2} & X_{3,3} & X_{3,4} \\ X_{4,3} & X_{4,4} & / \end{matrix}$$

- The function band_align() is called with band = $abs(M - N) + 2^x$ until its value is greater or equal to virtually best allignment using at least $2^x + 1$ indels.