**HW#3 Part 1**

**Algorithms, Spring 2016, SNU**

**Due**: May 31 23:59, 2016

**Submission to**: Hongryul Ahn ([hrahn@snu.ac.kr](mailto:hrahn@snu.ac.kr))

(mail title: [Algorithm HW3] StudentID Name)

**What to submit**: a zip file that includes README and your program. The file should be named with your university id first then a string “HW3-1”.

e.g.) 2013-12345-HW3-1.zip (that include README and 2013-12345-HW3-1.c)

**README file:**

It should contain execution environment and all command lines that should run by cut-and-paste. If you use C, then the contents will be

OS: linux (or windows, ios)

Machine: PC (or server with domain address like snubi2@snu.ac.kr)

Program language and version: python 2.7(or …)

External library: No

$ gcc –o 2013-12345-HW3-1 2013-12345-HW3-1.c

$ 2013-12345-HW3-1 geneA.fa geneB.fa

**Programming language**: any of C, C++, Java, Perl, Python, Matlab, R

**Delay penalty**: 10% per day

**Input**: Two long DNA sequences that will be posted on the web at <http://epigenomics.snu.ac.kr/teaching/2016/algorithm/HW3/geneA.fasta>

<http://epigenomics.snu.ac.kr/teaching/2016/algorithm/HW3/geneB.fasta>

(FASTA format sequences begin with a single-line sequence id (starts with “>”), followed by lines of DNA sequences)

**Output**: The pairs of bins and their alignments (see IMPLEMENTATION GUIDELINE)

e.g.) The format of output is repetition of three lines, one line for an id of a pair (comma separated) followed by the alignment of the pair sequences. Notice that the below result is just an example.

$ 2013-12345-HW3-1 geneA.fasta geneB.fasta

A0,B50

ACCTTTA\_\_\_ATCAGA….

\_ACCTTTA\_\_\_ATCAGA….

A50,B150

ATCG\_\_AT\_CA….

\_\_AT\_C\_\_GAT\_CA….

…

…

…

B1000,A950

\_\_CCTAG\_CT\_CTG…

CC\_\_TAGCT\_CT\_G…

**IMPLEMENTATION GUIDELINE**

1. For each of the two input sequences, make bins at positions of multiples of 50 starting at 50 with length of 100 called like A0, A50, A100, … , B0, B50, B100, ..., and so forth (A and B mean the first and the second input sequences and 0, 50, 100, … mean the starting positions of the bins). The reason why bins start every 50 characters rather than 100 characters is to identify LCS more completely by chopping searching region more densely.

e.g.) Assume the program get two input sequences (e.g. “$ 2013-12345-HW3-1 geneA.fasta geneB.fasta”) and the abbreviations of two sequences are A and B, respectively. Then,

The bin of sequence A ranging 0 to 99 is called A0

The bin of sequence A ranging 50 to 149 is called A50

…

\*Notice. The index of position starts from 0. The last bin of the input sequence would be less than 100 characters long. Make sure that the length of the last bin *l* should be 51≤ *l* ≤100.

2. Select the first bin of sequence A (i.e. A0). Compute LCS (longest common sequence) for all pairs of A0 and Bk (k=0,50,150,…,K). Then, print out the pair that has the longest LCS among all the pairs of A0 and Bk and print out its alignment also. The form of alignment is as follows.

Matching characters in LCS are printed and all regions between two matching LCS pairs are simply are left most aligned with filling in a space character ‘\_’ for the remaining region.

e.g.) Let A0 = ‘ATCC’ and B150 = ‘ATGC’ have the longest LCS among all the pairs of A0 and Bk. Then, the LCS is ‘ATC’ and the alignments of A0 and B150 w.r.t. LCS are ‘ATC\_’ and ‘AT\_C’, respectively. What to be printed out is,

A0,B150

ATC\_

AT\_C

\*Notice. If several candidate pairs have the same length LCS, choose the smallest-index bin. For instance, when the three candidate pairs, A0-B50, A0-B150, and A0-B500 have the LCS with the same length, A0-B50 should be printed out.

3. repeat Step 2 for all the other bins of A, then for all the bins of B.

\*Notice. If you work for a bin of B, the bin of B is printed out first and the bin of A next.

e.g.) Let we work B300 = ‘ACTC’. The pair of B300 and A350 (A350 = ‘ACCC’) have the longest LCS among all the pairs of B300 and Ak. Then, what to be printed out is,

B300,A350

AC\_C

ACC\_