PR1: BLAST

CS 182/282 Spring 2026

Released: Tuesday, February 10th, 2026

Due: 11:59 pm on Tuesday, February 24th, 2026

Overview

In this PR, you'll be implementing a simplified version of BLAST. When completed, you'll be able to query a given sequence against a database and identify all homologous regions!

This assignment is worth a total of 100 points.

Reading

- Basic Local Alignment Search Tool (Altschul et al., 1990)

Setup

To access test case files and data for P4, grab the files zipped on the website!

README

Your README should contain the following information:

- the command calls for any scripts required to run your program
- a general overview of your program's methodology
- a description of any auxiliary files you created or employed
- any known bugs
- anything else you want the TAs to know about your handin

Do not include any identifying information in your README.

Gradescope

To hand in your project, submit all of your project files to Gradescope. Make sure that any required shell scripts are in the root directory of your submission (or uploaded as individual files). We have made a small number of test cases visible to you after you submit so that you can ensure your program is working properly.

Our Gradescope autograder is currently configured to accept solutions written in Python 3, Java, R, and Julia. If you plan to use a different programming language that is installed on the department machines, you must let the TA staff know as soon as possible so we can update our autograder settings. However, note that all of the TAs have completed this course in Python 3, so we can't promise any help with other languages.

P1: Local Alignment (15 points)

Your first task will be to implement the local alignment (Smith-Waterman) algorithm. You should be able to adapt your code from CS 181; just **make sure that it is generalized to work with both DNA and protein sequences**. You should submit a shell script called `local.sh` which can be run as follows:

```
$ sh local.sh <seqs.txt> <mat.m> <gap penalty>
```

where `seqs.txt` is a text file containing two upper-case sequences from the same alphabet separated by a newline, `mat.m` is a text file containing the scoring matrix to be used (with all letters in uppercase, all rows separated by newlines and all columns separated by spaces, and an `X` in the top-left cell), and `gap penalty` is either some non-positive integer or the string `negInf` (which denotes $-\infty$). Note that this latter functionality was not implemented in 181!

When called as above, the script should print out the optimal local alignment of the two sequences, followed by the optimal local alignment score (as an integer—you may assume that all finite scores and gap penalties will be integers). If more than one optimal local alignment exists, any one will be accepted.

Test Cases

Note that relative paths to each file may need to be modified to accommodate your directory structure.

```
$ sh local.sh localDNA1.txt unitary.m -1
C
C
1
```

```
$ sh local.sh localDNA1.txt unitary.m negInf
C
C
1
```

```
$ sh local.sh localDNA2.txt unitary.m -1
ACGT-GTCAACGT
ACGTCGT-AGC-T
5
```

```
$ sh local.sh localDNA2.txt unitary.m negInf
ACGT
ACGT
4
```

```
$ sh local.sh localProt1.txt pam250.m -1
AW-C
AWTC
30
```

```
$ sh local.sh localProt1.txt pam250.m negInf
AW
AW
19
```

You may find it helpful to break down your code into several functions with distinct purposes, in order to be able to import specific functions into later parts of this PR.

P2: Seeding (35 points)

Your second task will be to implement the seeding phase of the BLAST algorithm. You should submit a shell script called `seeding.sh` which can be run as follows:

```
$ sh seeding.sh <db.txt> <query.txt> <mat.m> <k> <T>
```

where `db.txt` is a text file containing a "database" of sequences from a common alphabet (each sequence on its own line), `query.txt` is a text file containing a query sequence from the same alphabet, `mat.m` is as described in P1, k is a positive number which will be used to construct k-mer "words" from the query sequence, and T is a (possibly negative) threshold score which will be used to filter seeds (i.e., only seeds whose alignment scores with any k-mers of the query **exceed *T*** should be retained).

When called as above, the script should print out the following on successive lines:

- the query sequence
- the value of k
- the value of T
- the total number of seeds found
- a list of all seeds, each formatted as `Sequence <#> Position <#> Q-index <#>`, where all numbers are zero-indexed, the two position indices correspond to the first character of the k-mer of interest, and seeds are sorted by increasing sequence, position and query index in descending order of importance

You may find it useful to also write results to a file to streamline input to your extension script in P3!

Test Cases

```
$ sh seeding.sh db1.txt query1.txt pam250.m 4 2 # > seeds1.txt
ISTRAILSRIN
4
2
21
Sequence 1 Position 0 Q-index 5
Sequence 2 Position 0 Q-index 3
Sequence 3 Position 0 Q-index 4
Sequence 3 Position 0 Q-index 5
Sequence 3 Position 1 Q-index 4
Sequence 3 Position 1 Q-index 5
Sequence 3 Position 1 Q-index 6
Sequence 3 Position 2 Q-index 6
Sequence 4 Position 0 Q-index 3
Sequence 4 Position 1 Q-index 4
Sequence 4 Position 2 Q-index 5
Sequence 5 Position 0 Q-index 3
Sequence 5 Position 1 Q-index 4
Sequence 5 Position 1 Q-index 5
Sequence 5 Position 2 Q-index 4
Sequence 5 Position 2 Q-index 5
Sequence 5 Position 2 Q-index 6
Sequence 5 Position 3 Q-index 6
Sequence 5 Position 4 Q-index 0
Sequence 5 Position 4 Q-index 4
Sequence 5 Position 4 Q-index 5
```

```
$ sh seeding.sh db2.txt query2.txt pam250.m 3 3 # > seeds2.txt
DEHTSQ
3
3
36
Sequence 0 Position 0 Q-index 0
Sequence 0 Position 0 Q-index 1
Sequence 0 Position 1 Q-index 0
Sequence 0 Position 1 Q-index 1
Sequence 0 Position 2 Q-index 0
Sequence 0 Position 2 Q-index 1
Sequence 0 Position 3 Q-index 0
Sequence 0 Position 3 Q-index 1
Sequence 0 Position 4 Q-index 0
Sequence 0 Position 4 Q-index 1
Sequence 0 Position 5 Q-index 0
Sequence 0 Position 5 Q-index 1
Sequence 0 Position 6 Q-index 0
Sequence 0 Position 6 Q-index 1
Sequence 0 Position 7 Q-index 0
Sequence 0 Position 7 Q-index 1
Sequence 0 Position 8 Q-index 0
Sequence 0 Position 8 Q-index 1
Sequence 0 Position 9 Q-index 0
Sequence 0 Position 9 Q-index 1
Sequence 0 Position 10 Q-index 0
Sequence 0 Position 10 Q-index 1
Sequence 0 Position 11 Q-index 0
Sequence 0 Position 11 Q-index 1
Sequence 1 Position 0 Q-index 0
Sequence 1 Position 1 Q-index 0
Sequence 1 Position 2 Q-index 0
Sequence 1 Position 3 Q-index 0
Sequence 1 Position 4 Q-index 0
Sequence 1 Position 5 Q-index 0
Sequence 1 Position 6 Q-index 0
Sequence 1 Position 7 Q-index 0
Sequence 1 Position 8 Q-index 0
Sequence 1 Position 9 Q-index 0
Sequence 1 Position 10 Q-index 0
Sequence 1 Position 11 Q-index 0
```

You may find it helpful to break down your code into functions that accomplish the following tasks (as presented in class):

- creating all possible k-mers from the query
- creating all possible k-mers from the alphabet
- calculating the ungapped alignment score of two words
- identifying all k-mers from the alphabet which have an ungapped alignment score greater than T with any k-mer from the query
- finding the positions of all seeds in the database

P3: Extension (35 points)

Your third task will be to implement the extension phase of the BLAST algorithm. You should submit a shell script called `extension.sh` which can be run as follows:

```
$ sh extension.sh <db.txt> <mat.m> <seeds.txt> <X> <S>
```

where `db.txt` and `mat.m` are as described in P2, `seeds.txt` is a text file containing seed positions (formatted as output from `seed.sh`), X is a positive falloff score which will be used to terminate extension, and S is a non-negative cutoff score which will be used to filter alignments and identify High-Scoring Segment Pairs (HSPs).

You should extend each seed as follows:

- compute the score of the seed alignment
- extend the query and database sequence simultaneously to the right one base at a time, computing the new score at each index
- stop extending to the right when the falloff from the maximum score seen thus far **exceeds *X*** *or* the end of either sequence is reached, and trim back to the alignment which produced the maximum score
- re-initialize the current score to the original score of the seed alignment
- extend the query and database simultaneously to the left one base at a time, computing the new score (including the original seed but **not** the rightward extension) at each index
- stop extending to the left when the falloff from the maximum score seen thus far *during leftward extension only* **exceeds *X*** *or* the end of either sequence is reached, and trim back to the alignment which produced the maximum score
- compute the final score of the extension and retain only if it **exceeds *S***.

Note that you could equally choose to extend first to the left and then to the right; this procedure should be agnostic as to the initial direction of extension, since the initial score for each extension is that of the original seed by itself.

When called as above, the script should print out the following on successive lines:

- the first three lines of `seeds.txt` (containing the query sequence and the values of k and T)
- the number of extended seeds whose ungapped alignment scores are greater than S (HSPs)
- 35 dashes (`-`)
- a list of all **unique** HSPs, sorted in decreasing order of score and then length (break any additional ties by implementing the index-based sorted order as described in P2), formatted as described below, and each followed by 35 dashes (i.e., the final line of your output should be dashes)

Each HSP should be printed as follows:

```
Sequence <#> Position <#> Q-index <#>
<query alignment>
<database sequence alignment>
<alignment score>
```

You may observe that multiple seeds may ultimately be extended into the same HSP. If this occurs, only print the alignment once.

Test Cases

```
$ sh extension.sh db1.txt pam250.m seeds1.txt 2 7 # using output of seeding example 1
ISTRAILSRIN
4
2
8
-----------------------------------
Sequence 4 Position 0 Q-index 3
RAILSR
HGFMGH
11
-----------------------------------
Sequence 5 Position 0 Q-index 3
RAILSRI
WIIFMII
9
-----------------------------------
Sequence 3 Position 0 Q-index 5
ILSRI
IIFMI
9
-----------------------------------
Sequence 5 Position 1 Q-index 5
ILSRI
IIFMI
9
-----------------------------------
Sequence 3 Position 1 Q-index 5
ILSRI
IFMII
8
-----------------------------------
Sequence 1 Position 0 Q-index 5
ILSR
FMGH
8
-----------------------------------
Sequence 2 Position 0 Q-index 3
RAIL
HGFM
8
-----------------------------------
Sequence 5 Position 0 Q-index 3
RAIL
WIIF
8
-----------------------------------
```

```
$ sh extension.sh db2.txt pam250.m seeds2.txt 1 9
DEHTSQ
3
3
9
-----------------------------------
Sequence 0 Position 0 Q-index 0
DEHTSQ
DDDDDD
10
-----------------------------------
Sequence 0 Position 1 Q-index 0
DEHTSQ
DDDDDD
10
-----------------------------------
Sequence 0 Position 2 Q-index 0
DEHTSQ
DDDDDD
10
-----------------------------------
Sequence 0 Position 3 Q-index 0
DEHTSQ
DDDDDD
10
-----------------------------------
Sequence 0 Position 4 Q-index 0
DEHTSQ
DDDDDD
10
-----------------------------------
Sequence 0 Position 5 Q-index 0
DEHTSQ
DDDDDD
10
-----------------------------------
Sequence 0 Position 6 Q-index 0
DEHTSQ
DDDDDD
10
-----------------------------------
Sequence 0 Position 7 Q-index 0
DEHTSQ
DDDDDD
10
-----------------------------------
Sequence 0 Position 8 Q-index 0
DEHTSQ
DDDDDD
10
-----------------------------------
```

You may find it helpful to break down your code into functions that accomplish the following tasks (as presented in class):

- extracting the relevant information from the output of the seeding phase
- calculating the ungapped alignment score of two sequences
- extending seeds in one direction at a time according to the guidelines above
- filtering and sorting unique HSPs from the full list of extended seeds

P4: Statistics (15 points)

Your final task will be to explore the statistics underlying the BLAST algorithm. You should submit a script called `statistics.sh` which can be run as follows:

```
$ sh statistics.sh P4db.txt pam250.m
```

where `P4db.txt` is a "database" text file containing sequences of 10 human proteins and `pam250.m` is the PAM250 scoring matrix (both provided).

When called as above, the script should carry out the following:

- determine the frequency of each amino acid in the database
- generate 100 random protein sequences using the frequencies computed above, each of random length between the minimum and maximum length of sequences in the database
- run ungapped local alignment using the PAM250 matrix on sequences 1 & 2, 3 & 4, . . . , 99 & 100

In addition to your shell script, you should submit the two text files: `random.txt`, containing your 100 randomly-generated protein sequences (separated by newlines), and `statistics.txt`, containing your answers to the following questions:

1. What are the frequencies of each amino acid (in fractional—not decimal—form)? List them in decreasing order.
2. What are your 50 alignment scores?
3. Do your data points appear to be clustered? Are there outliers?
4. How might these statistics be relevant to the BLAST algorithm?

This exercise is intended as an open-ended introduction to how the values associated with BLAST are empirically determined; feel free to comment on any other observations you find interesting!