

# PS4

---

## DNA SEQUENCE ALIGNMENT

We will:

- implement the DNA sequence alignment assignment described at

<http://www.cs.princeton.edu/courses/archive/fall13/cos126/assignments/sequence.html>

- install [Valgrind](#), a memory analysis tool
- measure and report the space and time performance of the implementation

Some implementation notes:

- Download the test files from here:

<ftp://ftp.cs.princeton.edu/pub/cs126/sequence.zip>

- Your code should accept the two strings to be matched from stdin, e.g., as piped in from a test file:

```
% ED < example10.txt
```

## IMPLEMENTATION

- You may elect to implement any of four solution approaches:
  - Recursive without memoization (too slow to be practical, but order  $N$  in space)
  - Recursive using memoization
  - Dynamic programming using an  $N \times M$  matrix, per the Princeton problem set (aka, the Needleman-Wunsch method)
  - Using Hirschberg's algorithm, which is linear in space

We're going to use most of class time discussing the Needleman-Wunsch approach per the Princeton PS.

- If you implement the dynamic programming with a matrix approach, remember that the solution is in two parts:
  1. Filling out the  $N \times M$  matrix per the min-of-three-options formula, bottom to top, right to left (this gives you the optimal edit distance in the upper-left,  $[0][0]$  cell of the matrix);
  2. Traversing the matrix from top-to-bottom, left-to-right (i.e.,  $[0][0]$  to  $[N][M]$ ) to recover the choices you made in filling it, and thereby also recovering the actual edit sequence.
- **In any case**, you should create a class file ED (for “Edit Distance”) with the

- A method `int OptDistance()` which populates the matrix based on having the two strings and returns the optimal distance (from the `[0][0]` cell of the matrix when done)
- A method `string Alignment()` which traces the matrix and returns a string that can be printed to display the actual alignment. In general, this will be a multi-line string—i.e., with embedded `\n`'s.
- You should have a main routine that accepts the two strings from `stdin`, uses your ED (Edit Distance) class to do the work, and then prints the result to `stdout`. Remember that your final output should look like this:

```
% ./ED < example10.txt
Edit distance = 7
A T 1
A A 0
C - 2
A A 0
G G 0
T G 1
T T 0
A - 2
C C 0
C A 1
```

- You have to allocate the memory for the `opt` matrix dynamically, after you read in the two strings and figure out how long they are. There are a few different ways to do this:
  - vector of columns, each containing a row vector
  - one long vector, with internal calculations to treat it as a matrix

- The dynamic programming solution we discussed requires filling the whole matrix with values (step 1 of the two-part solution), so it's  $N^2$  in space (or more precisely,  $N \times M$ ).

So you shouldn't expect to have your code work for the test case with two 500,000 char strings. Assuming you use a 32-bit int to hold edit distance values in your matrix, that's 2 MB of data squared, or 4,000 GB.

Probably your computer can't allocate this much RAM.

The largest problem you should be able to handle is in `ecoli28284.txt`. This should cause you to allocate an array of approx. 800 million values. Assuming you're using 4-byte ints, that's 3.2 GB of data.

Don't worry about larger cases unless you want to explore alternate approaches. If so, there is a solution which computes the optimal alignment in *linear space* (and quadratic time). This is known as *Hirschberg's algorithm* (1975).

- After you have things working, add code to calculate and print execution time. You may use SFML's `sf::Clock` and `sf::Time` classes, as follows:
  - To your main routine, `#include <SFML/System.hpp>` at the top.
  - Then define the following two objects in your main:

```
sf::Clock clock;  
sf::Time t;
```

At the end of main, after computing the solution, capture the running time:

```
t= clock.getElapsedTime();
```

are the steps:

computing4summer2018

Home

portfolio

psX

ps7b

ps7a

ps6

ps5

- Install Valgrind:

```
sudo apt-get install valgrind
```

- Make sure your code is compiled and linked with debugging information (-g flag).
- Use Valgrind to run your code with the “massif” heap analysis tool; e.g.:
  - `valgrind --tool=massif ./ED < ../sequence/ecoli28284.txt`
- Then Valgrind will produce a log file named `massif.out.XXXXX`, where XXXXX is the process ID from your run. View with file with the `ms_print` utility that's part of the Valgrind distribution; e.g.:
  - `ms_print massif.out.11515 | less`
- By examining the `massif.out` file, confirm that the amount of memory used matches your expectations.
- Also, you can use the program `massif-visualizer` to view the logs (see <http://milianw.de/tag/massif-visualizer>) -- install with `sudo apt-get install massif-visualizer`.
- More documentation on Valgrind and massif is here: <http://valgrind.org/docs/manual/ms-manual.html>.
- Submit a `ps4-readme.txt` file, and include it with your code submission.

## MAKING THE GRADER HAPPY

The grader will compile your code and check it with the following input files:

```
bothgans20.txt  ecoli25000.txt  ecoli70000.txt  example10.txt  fl18.txt
```

```
% ./ED < example10.txt
```

```
Edit distance = 7
```

```
A T 1
```

```
A A 0
```

```
C - 2
```

```
A A 0
```

```
G G 0
```

```
T G 1
```

```
T T 0
```

```
A - 2
```

```
C C 0
```

```
C A 1
```

[Home](#)[portfolio](#)[psX](#)[ps7b](#)[ps7a](#)[ps6](#)[ps5](#)

You must follow this format (and capitalization of “Edit distance”) exactly, and you can't have trailing spaces at the ends of the lines.

- You should print the elapsed time after all this. These lines will be ignored.
- Everything must be in a directory named ps4, and per usual, make sure to remove object files before tarring up your work.

## REPORTING YOUR RESULTS

**Please include the following data in the `ps4-readme.txt` file to get full credit.**

- CPU speed in MHz
- input-size, the length of the problem string, for these test problems:
  - `ecoli2500.txt`
  - `ecoli5000.txt`

Wunsch, Hirschberg, other

Home

portfolio

psX

ps7b

ps7a

ps6

ps5

- array-method, e.g.: vectors, c-arrays, hash-table, other
- operating-system, e.g.: x86-unix-native, mac-os-x, lubuntu, other
- cpu-type, e.g.: core i3, core i7, AMD (and model), etc.

## SUBMITTING

Put all of your work into a directory named `ps4`: all of your code files, any `Makefile`, and the completed `ps4-readme.txt` file. **Make sure to run** `make clean` **before archiving**.

The executable file that your `Makefile` builds should be called `EditDistance` (the grading script checks that this executable builds successfully.)

Submit using the submit utility as follows:

**submit schakrab ps4 ps4**

## GRADING RUBRIC

### Core implementation: 6

3 pts for filling the matrix and getting the edit distance;

3 pts for traversing the matrix to retrieve the path

### Makefile: 2

**Performance time & space in** `readme`: 2