PS5: DNA-Alignment

Assignment Overview

This project implemented a DNA sequence alignment tool using a dynamic programming algorithm to calculate the edit distance between two genetic strings. Given that DNA mutations often involve substitutions and deletions, the program aims to determine how closely related two sequences are by evaluating the cost to transform one into the other. The program used a type of scoring system where matches cost 0, mismatches cost 1, and gaps cost 2. The result is not only the edit distance value, but also a full visual alignment of the two sequences showing how characters or gaps correspond. This assignment introduced the Needleman-Wunsch method and showed how recursion could be translated into an efficient dynamic programming solution. The over arching goal revolved around reducing time complexity and improving space usage.

Key Algorithms / Data Structures / OO Designs

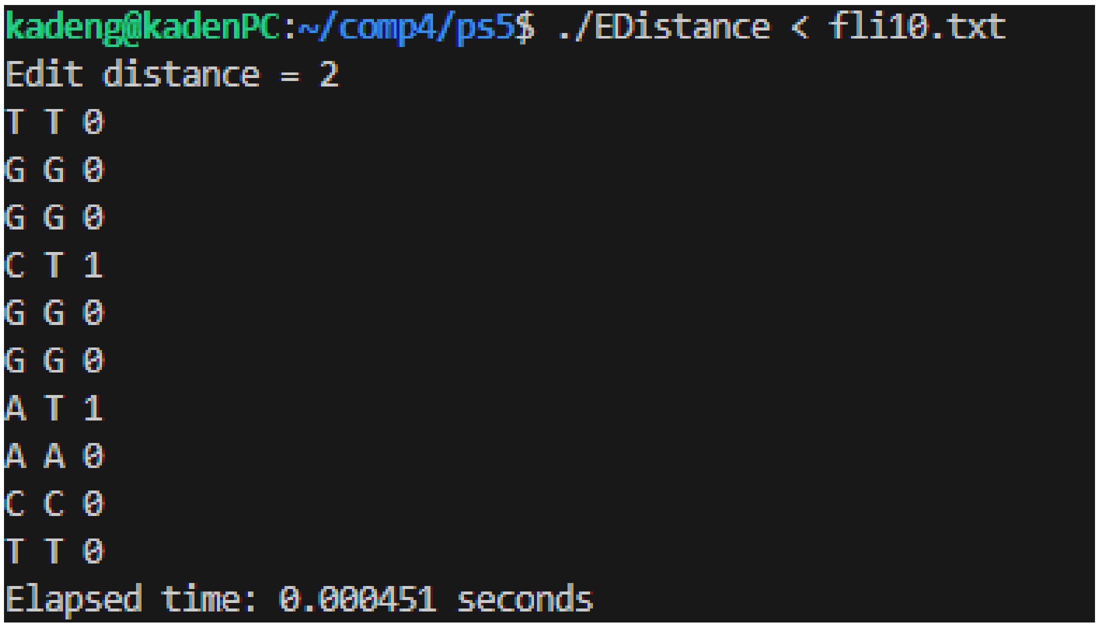
The project centered around the Needleman-Wunsch algorithm implemented via a bottom-up dynamic programming matrix. The ‘EDistance‘ class was created to encapsulate all logic related to the alignment, including a 2D matrix dynamically allocated to store optimal subproblem results, and a recursive scoring relationship translated into loops. It also included static helper functions such as penalty(char, char) to evaluate match/mismatch costs and min3(int, int, int) to find the minimum among three options. A backtracking algorithm in the alignment() method was used to reconstruct the optimal alignment path by walking through the matrix. Additionally, input was taken via standard input to support our section’s autograder compatibility, and output followed the required format: edit distance first, followed by the formatted alignment. Execution time tracking was implemented using SFML’s sf::Clock for performance reporting.

What I Learned

Through PS5, I learned how to design and implement a classic dynamic programming algorithm for sequence alignment, translate recursive formulas into efficient iterative solutions using matrices, construct and walk back through a 2D DP table to reconstruct a trace of the alignment path, dynamically allocate and deallocate 2D memory, use SFML timing utilities to measure and report execution time, write formatted and autograder-compliant output using standard input and output streams, and test for correctness using Boost while understanding how to handle multiple valid optimal alignments.

What Doesn't Work

The core functionality of the alignment tool is correct for most inputs, and the program passes all unit and formatting tests under standard constraints. However, several edge cases fail in the autograder due to incorrect edit distance calculation caused by an extra value being added to the alignment cost. Specifically, for several test files involving large sequences (like the ecoli files), the output cost is consistently off by 2. This suggests that an extra penalty is being appended to the end of the computed alignment, possibly due to misalignment in the traceback logic or a mismatch in how trailing gaps are handled. I was not able to debug this issue. These discrepancies point to a consistent +2 error pattern that may be due to mismanagement of loop bounds or improper alignment termination during backtracking. Additionally, although the implementation is functional, it uses a full $O(n \cdot m)$ matrix for dynamic programming. As a result, it consumes significant memory and cannot handle very large inputs like the 500,000-character case without crashing. Implementing Hirschberg's algorithm would reduce memory to $O(n + m)$, but it was not implemented in this program.



```
kadeng@kadenPC:~/comp4/ps5$ ./EDistance < fli10.txt
Edit distance = 2
T T 0
G G 0
G G 0
C T 1
G G 0
G G 0
A T 1
A A 0
C C 0
T T 0
Elapsed time: 0.000451 seconds
```

Figure 6: Example Output: Aligned DNA sequences with an edit distance of 2

Source Code

```
Makefile:
# Compiler and flags
CC = g++
CFLAGS = --std=c++20 -Wall -Werror -pedantic -g
LIB = -lsfml-system -lboost_unit_test_framework

# Dependencies and objects
DEPS = EDistance.hpp
OBJECTS = EDistance.o

# Program name
PROGRAM = EDistance
TEST = test

.PHONY: all clean lint

# Default target: build the program and static library
all: $(PROGRAM) $(TEST) EDistance.a

# Rule to build the program
$(PROGRAM): main.o EDistance.a
	$(CC) $(CFLAGS) -o $@ main.o EDistance.a $(LIB)

$(TEST): test.o $(OBJECTS)
	$(CC) $(CFLAGS) -o $@ $^ $(LIB)
# Rule to create the static library
EDistance.a: $(OBJECTS)
	ar rcs $@ $^
```

```

# Rule to compile object files
%.o: %.cpp $(DEPS)
	$(CC) $(CFLAGS) -c $< -o $@

# Clean up build files
clean:
	rm -f *.o $(PROGRAM) $(TEST) EDistance.a

# Run cpplint for static analysis
lint:
	cpplint *.cpp *.hpp

main.cpp:
// COPYRIGHT 2025 Kaden Gardiner
#include <iostream>
#include <string>
#include <SFML/System.hpp>
#include "EDistance.hpp"

int main() {
    std::string x, y;
    std::getline(std::cin, x);
    std::getline(std::cin, y);
    sf::Clock clock;

    EDistance ed(x, y);
    std::cout << "Edit distance = " << ed.optDistance() << std::endl;
    std::cout << ed.alignment();


    sf::Time elapsed = clock.getElapsedTime();
    std::cout << "Elapsed time: " << elapsed.asSeconds() << " seconds" <<
       std::endl;

    return 0;
}

EDistance.hpp:
// COPYRIGHT 2025 Kaden Gardiner
#pragma once

#include <string>
#include <iostream>
#include <algorithm>
#include <sstream>

class EDistance {
 public:
    EDistance(const std::string& s1, const std::string& s2);
    ~EDistance();
    static int penalty(char a, char b);
    static int min3(int a, int b, int c);
    int optDistance();
    std::string alignment();

 private:
    std::string _x, _y;
    int _m, _n;
    int** _opt;
};


EDistance.cpp:
// COPYRIGHT 2025 Kaden Gardiner
#include "EDistance.hpp"

EDistance::EDistance(const std::string& s1, const std::string& s2)
    : _x(s1), _y(s2), _m(s1.length()), _n(s2.length()) {
    _opt = new int*[_m+1];
    for (int i = 0; i <= _m; ++i) {
        _opt[i] = new int[_n+1];
    }
}

EDistance::~EDistance() {
    for (int i = 0; i <= _m; ++i) {
        delete[] _opt[i];
    }
    delete[] _opt;
}

int EDistance::penalty(char a, char b) {
    return (a == b) ? 0 : 1;
}

int EDistance::min3(int a, int b, int c) {
    return std::min(std::min(a, b), c);
}

int EDistance::optDistance() {
    const int indel = 2;

    for (int i = 0; i <= _m; ++i) {
        _opt[i][_n] = (_m - i) * indel;
    }
    for (int j = 0; j <= _n; ++j) {
        _opt[_m][j] = (_n - j) * indel;
    }
    for (int i = _m - 1; i >= 0; --i) {
        for (int j = _n - 1; j >= 0; --j) {
            if (_x[i] == _y[j]) {
                _opt[i][j] = _opt[i + 1][j + 1];
            } else {
                int sub = _opt[i + 1][j + 1] + 1;
                int deleteX = _opt[i + 1][j] + indel;
                int deleteY = _opt[i][j + 1] + indel;
                _opt[i][j] = min3(sub, deleteX, deleteY);
            }
        }
    }

    return _opt[0][0];
}

std::string EDistance::alignment() {
    std::ostringstream result;
    int i = 0, j = 0;

    while (i < _m || j < _n) {
        if (i < _m && j < _n &&
            _opt[i][j] == _opt[i + 1][j + 1] + penalty(_x[i], _y[j])) {
            result << _x[i] << " " << _y[j] << " " << penalty(_x[i], _y[j
               ]) << "\n";
            ++i;
            ++j;
        } else if (i < _m && _opt[i][j] == _opt[i + 1][j] + 2) {
            result << _x[i] << " - 2\n";
            ++i;
        } else if (j < _n && _opt[i][j] == _opt[i][j + 1] + 2) {
            result << "- " << _y[j] << " 2\n";
            ++j;
        }
    }
    return result.str();
}

test.cpp:
// COPYRIGHT 2025 Kaden Gardiner
#define BOOST_TEST_MODULE EDistanceTest
#include <boost/test/included/unit_test.hpp>
#include "EDistance.hpp"

BOOST_AUTO_TEST_CASE(wrongConstTest) {
    std::ifstream file("example10.txt");
    std::string string1, string2;
    int x = 7;

    file >> string1 >> string2;
    EDistance e(string1, string2);
    BOOST_REQUIRE_EQUAL(e.optDistance(), x);
}

BOOST_AUTO_TEST_CASE(wrongDirectionTest) {
    std::string string1 = "GTA", string2 = "GTA";
    std::string expected_output = "G G 0\nT T 0\nA A 0\n";

    EDistance e(string1, string2);
    e.optDistance();
    std::string result = e.alignment();
    BOOST_REQUIRE_EQUAL(result, expected_output);
}

BOOST_AUTO_TEST_CASE(testSwappedCoils) {
    std::string string1 = "CAGT";
    std::string string2 = "CGT";

    std::string output = "C C 0\nA - 2\nG G 0\nT T 0\n";
    EDistance e(string1, string2);
    e.optDistance();
    std::string result = e.alignment();
    BOOST_REQUIRE_EQUAL(result, output);
}

BOOST_AUTO_TEST_CASE(testCutEnds) {
    std::string string1 = "TAGCT";
    std::string string2 = "TAGT";
    std::string output = "T T 0\nA A 0\nG G 0\nC - 2\nT T 0\n";


    EDistance e(string1, string2);
    e.optDistance();
    std::string result = e.alignment();
    BOOST_REQUIRE_EQUAL(result, output);
}
```