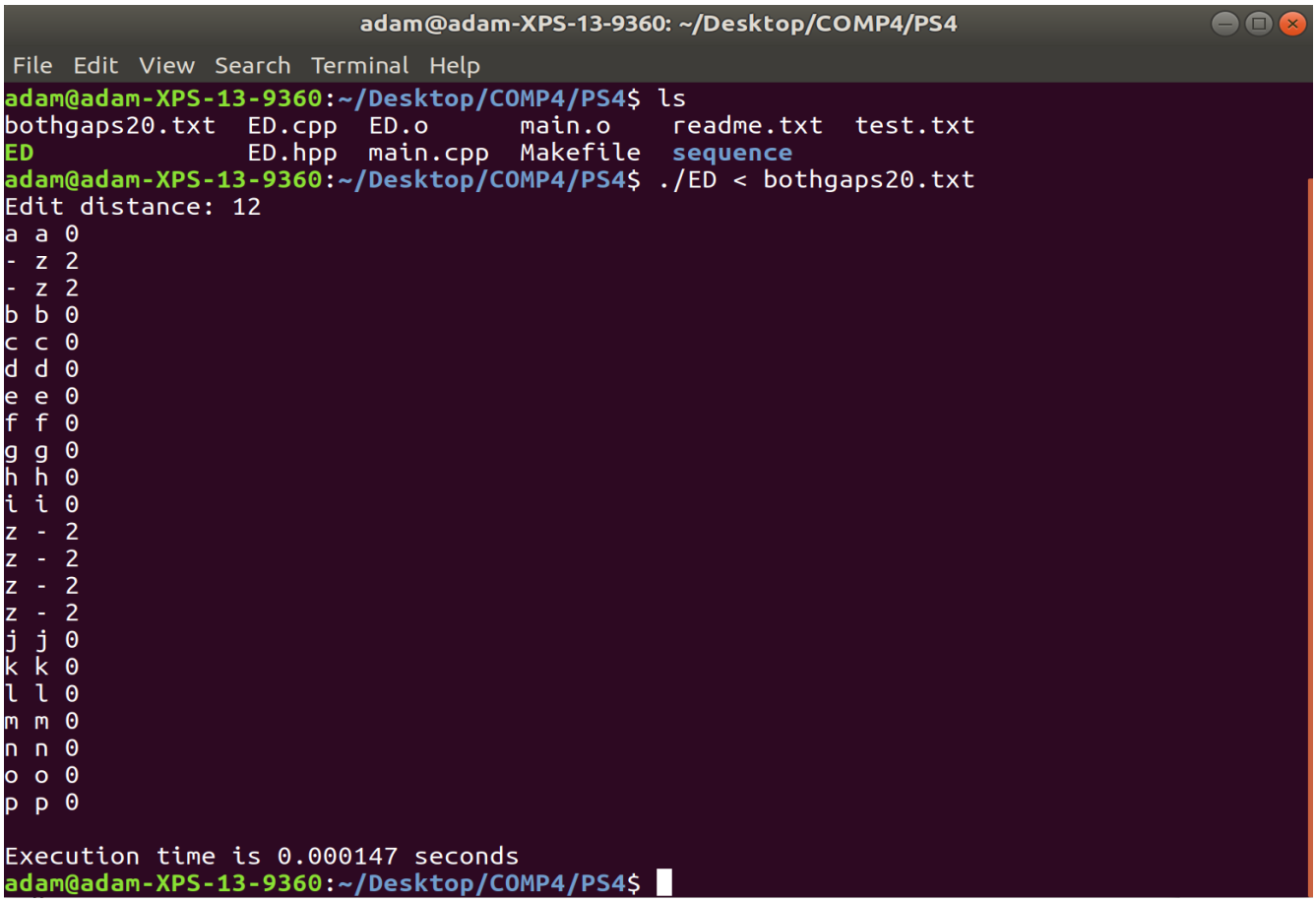
PS4 DNA Sequence Alignment

In this project, I had to use the Needleman-Wunsch method to find the best allignment for two strings. In this assignment OO designs that were implemented were classes, and data structures used were arrays to create a 2d array in order to implement the Needlema-Wunsch method. In this assignment I learned different ways to use valgrind in order to find memory leaks, and the SFML library in order to determine the run time for different sequences.

One problem that I ran into was that I got stuck on how to delete a 2d array to get rid of memory leaks, but to solve this problem I used google to help me with it.



```
 1: CC = g++
 2: CFLAGS = -std=c++11 -c -g -Wall -Werror -pedantic
 3: LIBS = -lsfml-system
 4:
 5: all: ED
 6:
 7: ED: main.o ED.o
 8:         $(CC) main.o -o ED $(LIBS)
 9:
10: main.o: main.cpp
11:         $(CC) -c $(CFLAGS) main.cpp
12:
13: planets.o: ED.cpp ED.hpp
14:         $(CC) -c $(CFLAGS) ED.cpp ED.hpp
15:
16: clean:
17:         rm *.o ED massif*
```

```
 1: #include <SFML/System.hpp>
 2: #include <iostream>
 3: #include <vector>
 4: #include <string>
 5:
 6: using namespace std;
 7: using namespace sf;
 8:
 9: class ED {
10: public:
11:     ED(string a, string b);
12:
13:     int penalty(char a, char b);
14:     int min(int a, int b, int c);
15:     int OptDistance();
16:     string Alignment();
17:     int getCost();
18:
19:     ~ED();
20:
21: private:
22:     string x, y;
23:     int M, N, cost;
24:     int** opt;
25:
26: };
```

```
 1: #include "ED.hpp"
 2:
 3: ED::ED(string a, string b) {
 4:
 5:     this->x = a;
 6:     this->y = b;
 7:
 8:     x += '-';
 9:     y += '-';
10:
11:     M = x.length() + 1;
12:     N = y.length() + 1;
13:
14:     opt = new int*[M];
15:     for (int i = 0; i < M; i++)
16:         opt[i] = new int[N];
17: }
18:
19: int ED::penalty(char a, char b) {
20:     if (a == b)
21:         return 0;
22:     else
23:         return 1;
24: }
25:
26: int ED::min(int a, int b, int c) {
27:     if (a < b && a < c)
28:         return a;
29:     else if (b < a && b < c)
30:         return b;
31:     else
32:         return c;
33: }
34: int ED::OptDistance() {
35:
36:     //add bottom outside
37:     int j = 0;
38:     for (int i = N-1; i >= 0; i--) {
39:         opt[M-1][i] = j;
40:         j += 2;
41:     }
42:
43:     //add right outisde
44:     j = 0;
45:     for (int i = M-1; i >= 0; i--) {
46:         opt[i][N-1] = j;
47:         j += 2;
48:     }
49:
50:     //compare
51:     for (int i = M-2; i >= 0; i--) {
52:         for (int j = N-2; j >= 0; j--) {
53:             if (x[i] != y[j]) {
54:                 opt[i][j] = min(opt[i+1][j] + 2, opt[i][j+1] + 2, opt[i+1][j
+1] + 1);
55:             }
56:             else
57:                 opt[i][j] = opt[i+1][j+1];
58:         }
59:     }
60:
```

```
    string out = Alignment();
    cout << "Edit distance: " << cost << endl;
    cout << out << endl;

    for (int i = 0; i < M; i++)
        delete [] opt[i];

    delete [] opt;
    return 0;
}
string ED::Alignment() {
    vector<char> out;

    int i = 0,
        j = 0,
        pen;;
    while (i < M-2|| j < N-2) {
        if (x[i] == y[j]) {
            pen = penalty(x[i], y[j]);
            out.push_back(x[i]);
            out.push_back(' ');
            out.push_back(y[j]);
            out.push_back(' ');
            i++;
            j++;
        }
        else if (opt[i][j] == opt[i+1][j+1] + 1) {
            pen = penalty(x[i], y[j]);
            out.push_back(x[i]);
            out.push_back(' ');
            out.push_back(y[j]);
            out.push_back(' ');
            i++;
            j++;
        }
        else if (opt[i][j] == opt[i+1][j] + 2) {
            pen = penalty(x[i], y[j]) + 1;
            out.push_back(x[i]);
            out.push_back(' ');
            out.push_back('-');
            out.push_back(' ');
            i++;
        }
        else if (opt[i][j] == opt[i][j+1] + 2) {
            pen = penalty(x[i], y[j]) + 1;
            out.push_back('-');
            out.push_back(' ');
            out.push_back(y[j]);
            out.push_back(' ');
            j++;
        }
        out.push_back('0' + pen);
        cost += pen;
        out.push_back('\n');
    }
    string new_out(out.begin(), out.end());
    return new_out;
}

int ED::getCost() {
    return cost;
```

```
122: }
123:
124: ED::~ED() {}
```

```
    1: #include "ED.cpp"
    2:
    3: int main(int argc, char* argv[]) {
    4:
    5:     string a, b;
    6:     cin >> a;
    7:     cin >> b;
    8:
    9:     ED ed(a, b);
   10:
   11:     Clock clock;
   12:     Time t;
   13:     ed.OptDistance();
   14:     t = clock.getElapsedTime();
   15:     cout << "Execution time is " << t.asSeconds() << " seconds\n";
   16:     //cout << "Edit distance (for longer sequences): " << ed.getCost() << en
dl;
   17:
   18:     return 0;
   19: }
```