

```
1: CC = g++
2: CFLAGS = -Wall -Werror -pedantic --std=c++14
3: LIBS = -lboost_unit_test_framework
4: DEPS = EDistance.h
5: SFMLFLAGS = -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
6:
7: %.o: %.cpp $(DEPS)
8:     $(CC) $(CFLAGS) -c $<
9:
10: all: EDistance
11:
12: EDistance: main.o EDistance.o
13:     $(CC) -g $(CFLAGS) -o EDistance $^ $(LIBS) $(SFMLFLAGS)
14:     cpplint --filter=--runtime/references *.cpp *.h
15:
16: clean:
17:     rm *.o EDistance
```

```
1: // Copyright 2022 Anson Cheang, and Andy Nguyen
2: #include <iostream>
3: #include <SFML/System.hpp>
4: #include "EDistance.h"
5:
6: using namespace std; //NOLINT
7:
8: int main(int argc, const char* argv[]) {
9:     sf::Clock clock;
10:    sf::Time t;
11:
12:    string input1, input2;
13:    cin >> input1;
14:    cin >> input2;
15:    EDistance dna(input1, input2);
16:    int distance = dna.optDistance();
17:    cout << "Edit Distance = " << distance << endl;
18:    cout << dna.alignment();
19:
20:    t = clock.getElapsedTime();
21:    // cout << "Edit Distance = " << distance << endl;
22:    cout << "Execution time is " << t.asSeconds() << " seconds \n";
23: }
```

```
1: // Copyright 2022 Anson Cheang, and Andy Nguyen
2: #ifndef _HOME_IIFORCE_BADNAME_COMP4_PS5_EDISTANCE_H_
3: #define _HOME_IIFORCE_BADNAME_COMP4_PS5_EDISTANCE_H_
4:
5: #include <iostream>
6: #include <string>
7: #include <vector>
8: #include <algorithm>
9:
10: using namespace std; //NOLINT
11:
12: class EDistance {
13: public:
14:     /* accepts the two strings to be compared, and allocates any
15:        data structures necessary into order to do the work
16:        (e.g., the NÃ\227M matrix).*/
17:     explicit EDistance(string N, string M);
18:
19:     /* returns the penalty for aligning chars a and b
20:        (this will be a 0 or 1)*/
21:     static int penalty(char a, char b);
22:
23:     // returns the minimum of the three arguments
24:     static int min(int a, int b, int c);
25:
26:     /* populates the matrix based on having the two strings, and returns
27:        the optimal distance (from the [0][0] cell of the matrix when done).
*/
28:     int optDistance();
29:
30:     /* traces the matrix and returns a string that can be printed to disp
lay
31:        the actual alignment. In general, this will be a multi-line string â
\200\224 i.e.,
32:        with embedded \n's.*/
33:     string alignment();
34:
35: private:
36:     string n, m;
37:     vector<vector<int>> matrix;
38: };
39:
40: #endif // _HOME_IIFORCE_BADNAME_COMP4_PS5_EDISTANCE_H_
```

```
1: // Copyright 2022 Anson Cheang, and Andy Nguyen
2:
3: #include <iostream>
4: #include <algorithm>
5: #include <cmath>
6: #include "EDistance.h"
7:
8: using namespace std; //NOLINT
9:
10: /* accepts the two strings to be compared, and allocates any
11:  data structures necessary into order to do the work
12:  (e.g., the NÅ\227M matrix).*/
13: EDistance::EDistance(string input_N, string input_M) {
14:     // n = row, m = column
15:     n = input_N;
16:     m = input_M;
17: }
18:
19: /* returns the penalty for aligning chars a and b
20:  (this will be a 0 or 1)*/
21: int EDistance::penalty(char a, char b) {
22:     if (a == b) {
23:         return 0;
24:     } else {
25:         return 1;
26:     }
27: }
28:
29: // returns the minimum of the three arguments
30: int EDistance::min(int a, int b, int c) {
31:     return std::min({a, b, c});
32: }
33:
34: /* populates the matrix based on having the two strings, and returns
35:  the optimal distance (from the [0][0] cell of the matrix when done).*/
36: int EDistance::optDistance() {
37:     int r = n.length();
38:     int c = m.length();
39:     int indel = 2;
40:     int match, del, insert;
41:
42:     for (int i = 0; i <= c; i++) {
43:         vector<int> temp;
44:         matrix.push_back(temp);
45:
46:         for (int j = 0; j <= r; j++) {
47:             matrix.at(i).push_back(0);
48:         }
49:     }
50:
51:     for (int i = 0; i <= c; i++) {
52:         matrix.at(i).at(r) = (c - i) * indel;
53:     }
54:
55:     for (int i = 0; i <= r; i++) {
56:         matrix.at(c).at(i) = (r - i) * indel;
57:     }
58:
59:     for (int i = c - 1; i >= 0; i--) {
60:         for (int j = r - 1; j >= 0; j--) {
61:             match = matrix.at(i + 1).at(j + 1) + penalty(m.at(i), n.at(j)
);
62:             del = matrix.at(i + 1).at(j) + indel;
63:             insert = matrix.at(i).at(j + 1) + indel;
64:             matrix.at(i).at(j) = min(match, del, insert);
```

```
65:         }
66:     }
67:     return matrix.at(0).at(0);
68: }
69:
70: /* traces the matrix and returns a string that can be printed to display
71: the actual alignment. In general, this will be a multi-line string â\200
\224 i.e.,
72: with embedded \n's.*/
73: string EDistance::alignment() {
74:     int indel = 2;
75:     string retStr;
76:
77:     int i = 0;
78:     int j = 0;
79:     int r = n.length();
80:     int c = m.length();
81:
82:     while (i < c || j < r) {
83:         if (i < c && j < r && matrix.at(i).at(j) == matrix.at(i + 1).at(j)
+ 1) + penalty(m[i], n[j])) { //NOLINT
84:             retStr = retStr + n[j] + " " + m[i] + " " + to_string(penalty
(m[i], n[j])) + "\n"; //NOLINT
85:             i++;
86:             j++;
87:         } else if (i < c && matrix.at(i).at(j) == matrix.at(i + 1).at(j)
+ indel) { //NOLINT
88:             retStr = retStr + "-" + " " + m[i] + " " + "2" + "\n";
89:             i++;
90:         } else if (j < r && matrix.at(i).at(j) == matrix.at(i).at(j + 1)
+ indel) { //NOLINT
91:             retStr = retStr + n[j] + " " + "-" + " " + "2" + "\n";
92:             j++;
93:         }
94:     }
95:     return retStr;
96: }
```