```
 1: // Copyright 2020 Michael Zogin
 2: #include "ED.h"  // NOLINT
 3: #include <string>
 4: #include <algorithm>
 5: #include <vector>
 6:
 7: ED::ED(const string &s1, const string &s2) {
 8:   string1 = s1;
 9:   string2 = s2;
10:   N = string1.length();
11:   M = string2.length();
12:
13:   // allocate the matrix
14:   matrix = new vector<vector<int>>(N + 1, vector<int>(M + 1));
15: }
16:
17: ED::~ED() {
18:   delete matrix;
19:   N = 0;
20:   M = 0;
21: }
22:
23: int ED::penalty(char a, char b) {
24:   if (a == b) {
25:     return 0;
26:   } else if (a != b) {
27:     return 1;
28:   }
29:   return -1;
30: }
31:
32: int ED::min(int a, int b, int c) {
33:   if ((a < b) && (a < c)) {
34:     return a;
35:   } else if ((b < a) && (b < c)) {
36:     return b;
37:   } else if (c < a && c < b) {
38:     return c;
39:   }
40:   return a;
41: }
42:
43: int ED::OptDistance() {
44:   int i, j;
45:   int N = string1.length();
46:   int M = string2.length();
47:
48:   for (i = 0; i <= M; i++) {
49:     for (j = 0; j <= N; j++) {
50:       matrix->at(i).push_back(0);
51:     }
52:   }
53:
54:   // this will fill far right column with base cases
55:   for (i = 0; i <= M; i++) {
56:     matrix->at(i).at(N) = 2 * (M - i);
57:   }
58:
59:   // this will fill bottom row with base cases
60:   for (j = 0; j <= N; j++) {
61:     matrix->at(M).at(j) = 2 * (N - j);
```

```
   62:   }
   63:
   64:   // go from bottom row up
   65:   for (i = (M - 1); i >= 0; i--) {
   66:     // go from rigth to left
   67:     for (j = (N - 1); j >= 0; j--) {
   68:       int opt1 = matrix->at(i + 1).at(j + 1) + penalty(string1[j], string2[i
]);
   69:       int opt2 = matrix->at(i + 1).at(j) + 2;
   70:       int opt3 = matrix->at(i).at(j + 1) + 2;
   71:
   72:       matrix->at(i).at(j) = min(opt1, opt2, opt3);
   73:     }
   74:   }
   75:
   76:   // return final calculated ED(top left of matrix)
   77:   return matrix->at(0).at(0);
   78: }
   79:
   80: string ED::Alignment() {
   81:   std::ostringstream ret;
   82:
   83:   int i = 0;
   84:   int j = 0;
   85:   int opt1, opt2, opt3;
   86:   int pen;
   87:   string ret_str;
   88:
   89:   // runs until we hit bottom right corner
   90:   while (i < M || j < N) {
   91:     try {
   92:       pen = penalty(string1[j], string2[i]);
   93:       opt1 = matrix->at(i + 1).at(j + 1) + pen;
   94:     } catch (const std::out_of_range &error) {
   95:       opt1 = -1;
   96:     }
   97:
   98:     try {
   99:       opt2 = matrix->at(i + 1).at(j) + 2;
  100:     } catch (const std::out_of_range &error) {
  101:       opt2 = -1;
  102:     }
  103:
  104:     try {
  105:       opt3 = matrix->at(i).at(j + 1) + 2;
  106:     } catch (const std::out_of_range &error) {
  107:       opt3 = -1;
  108:     }
  109:
  110:     if (matrix->at(i).at(j) == opt1) {
  111:       ret << string1[j] << " " << string2[i] << " " << pen << endl;
  112:
  113:       // moves diagonal
  114:       i++;
  115:       j++;
  116:     } else if (matrix->at(i).at(j) == opt2) {
  117:       ret << "- " << string2[i] << " 2" << endl;
  118:
  119:       // moves down
  120:       i++;
  121:     } else if (matrix->at(i).at(j) == opt3) {
```

```
122:       ret << string1[j] << " - 2" << endl;
123:
124:       // moves right
125:       j++;
126:     }
127:   }
128:   ret_str = ret.str();
129:   return ret_str;
130: }
```