```
    1: #include "ED.hpp"
    2: #include <vector>
    3: #include <iostream>
    4: #include <string>
    5:
    6: const int MATCH = 0;
    7: const int GAP = 2;
    8: const int SWITCH = 1;
    9:
   10: ED::ED(std::string a, std::string b) : x(a), y(b)
   11: {
   12:         x.append("-");
   13:         y.append("-");
   14:
   15:         m = x.length();
   16:         n = y.length();
   17:
   18:         grid = new int*[m];
   19:         for (unsigned int i = 0; i < m; i++)
   20:         {
   21:                 grid[i] = new int[n];
   22:         }
   23: }
   24:
   25: ED::~ED()
   26: {
   27:         for (unsigned int i = 0; i < m; i++)
   28:         {
   29:                 delete [] grid[i];
   30:         }
   31:
   32:         delete grid;
   33: }
   34:
   35: void ED::perimeterCases()
   36: {
   37:         for (unsigned int i = m; i > 0; i--)
   38:         {
   39:
   40:                 for (unsigned int j = n; j > 0; j--)
   41:                 {
   42:
   43:                         if(i == m)
   44:                                 grid[i-1][j-1] = (n - j) * GAP;//makes bo
ttom row
   45:
   46:                         if(j == n)
   47:                                 grid[i-1][j-1] = (m - i) * GAP;//makes ri
ghtmost column
   48:                 }
   49:         }
   50: }
   51:
   52: int ED::penalty(char a, char b)
   53: {
   54:         return ((a == b)) ? MATCH : SWITCH;
   55:
   56: }
   57:
   58: int ED::min(int& a, int& b, int& c)
   59: {
   60:         if(a < b)
   61:                 return (a < c) ? a : c;//if a is less than c return a, el
se return c
   62:         return (b < c) ? b : c;//if b is less than c return b, else retur
```

```
n c
   63: }
   64:
   65: int ED::optimalDistance()
   66: {
   67:         int optDistance;
   68:         perimeterCases();
   69:
   70:         for(unsigned int i = m -1; i > 0; i--)
   71:         {
   72:                 for(unsigned int j = n-1; j > 0; j--)
   73:                 {
   74:                         int bottom = grid[i][j-1] + GAP;
   75:                         int right = grid[i-1][j] + GAP;
   76:                         int diagonal = grid[i][j] + (penalty(x.at(i-1), y
.at(j-1)));
   77:
   78:                         optDistance = min (right, bottom, diagonal);//fin
d the smallest cost
   79:                         grid[i-1][j-1] = optDistance;
   80:                 }
   81:         }
   82:
   83:         return optDistance;
   84: }
   85:
   86: std::string ED::Alignment()
   87: {
   88:         std::string result;
   89:
   90:         unsigned int i = 0, j = 0;
   91:
   92:         int *current = &grid[i][j];
   93:         int *right = &grid[i][j+1];
   94:         int *bottom = &grid[i+1][j];
   95:         int *diagonal = &grid[i+1][j+1];
   96:
   97:         int *correctptr;//current;
   98:
   99:         while(i < m-1 || j < n-1)
  100:         {
  101:                 //check to see if the current number and right number = 2
  102:                 if(j < n-1 && (*current - *right) == GAP)
  103:                 {
  104:                         result.append("-");//x is a gap
  105:                         result.append(" ");
  106:                         result.push_back(y.at(j));//y is a letter
  107:                         result.append(" ");
  108:                         result.append("2");//save the cost
  109:                         result.append("\n");
  110:                         correctptr = right;
  111:                         j++;
  112:                 }
  113:                 else if(i < m-1 && (*current - *bottom) == GAP)
  114:                 {
  115:                         result.push_back(x.at(i));//x is a letter
  116:                         result.append(" ");
  117:                         result.append("-");//y is a gap
  118:                         result.append(" ");
  119:                         result.append("2");//save the cost
  120:                         result.append("\n");
  121:                         correctptr = bottom;
  122:                         i++;
  123:                 }
  124:                 else if(penalty(x.at(i), y.at(j)) == (*current - *diagona
```

```
l))
  125:                  {
  126:                          result.push_back(x.at(i));//x is a letter
  127:                          result.append(" ");
  128:                          result.push_back(y.at(j));//y is a letter
  129:                          result.append(" ");
  130:                          if(*current - *diagonal == SWITCH)
  131:                                  result.append("1");//save the cost
  132:                          if(*current - *diagonal == MATCH)
  133:                                  result.append("0");//save the cost
  134:                          result.append("\n");
  135:                          correctptr = diagonal;
  136:                          i++;
  137:                          j++;
  138:                  }
  139:                  //update pointers based on new current pointer
  140:                  current = correctptr;
  141:                  right = &grid[i][j+1];
  142:                  bottom = &grid[i+1][j];
  143:                  diagonal = &grid[i+1][j+1];
  144:          }
  145:          return result;
  146: }
  147:
  148: std::ostream& operator <<(std::ostream& left, const ED& ed)
  149: {
  150:          std::cout << std::endl;
  151:          std::cout << "   ";
  152:          for(unsigned int i = 0; i < ed.n; i++)
  153:          {
  154:                  std::cout << "   " << std::left << ed.y.at(i);
  155:          }
  156:          std::cout << std::endl;
  157:
  158:          for(unsigned int i = 0; i < ed.m; i++)
  159:          {
  160:                  std:: cout << std:: endl << ed.x.at(i) << "   ";
  161:                  for(unsigned int j = 0; j < ed.n; j++)
  162:                  {
  163:                          if(ed.grid[i][j] > 9)
  164:                                  std::cout << "  " << std::left << ed.grid
[i][j];//make two spaces if the grid number is 2 digits
  165:                          else
  166:                                  std::cout << "   " << std::left << ed.gri
d[i][j];//make 3 spaces if the grid is a single digit
  167:                  }
  168:          }
  169:          std::cout << std::endl << std::endl;
  170:          return left;
  171: }
```