**Programming assignment**

Goal: Given two strings X = x1 x2 . . . xm and Y = y1 y2 . . . yn find alignment of minimum cost.

Example: X= GAATTCAGTTA

Y= GGATCGA

Given- mismatch penalty=1, gap open penalty=0, gap extension penalty=2.

Sol- For this sequence matching problem, there can be 3 cases while aligning the letters with minimum cost:

Def. OPT(i, j) = min cost of aligning strings x1 x2 . . . xi and y1 y2 . . . yj.

* Case 1: OPT matches xi-yj.
  + pay mismatch for xi-yj + min cost of aligning two strings  
    x1 x2 . . . xi-1 and y1 y2 . . . yj-1
* Case 2a: OPT leaves xi unmatched.
  + pay gap for xi and min cost of aligning x1 x2 . . . xi-1 and y1 y2 . . . yj
* Case 2b: OPT leaves yj unmatched.
  + pay gap for yj and min cost of aligning x1 x2 . . . xi and y1 y2 . . . yj-1

Edit Distance: 9

G G 0

A G 1

A A 0

T - 2

T T 0

C C 0

A - 2

G G 0

T - 2

T - 2

A A 0

**PROGRAMMING IMPLEMENTATION**

EditDistance.java is used for calculating this minimum cost alignment of sequence where input file contains:

1. Mismatch penalty
2. Gap open penalty
3. Gap extension penalty
4. 2 string sequences to match separated by line break.

All the 3 penalties are gap separated. The EditDisatance.java class consist of following methods:

* **Main:** Process the input file, send the strings and Penalty values to EditDistance method for calculating distance matrix. Finally call FinalPattern method to display the alignment of 2 strings.
* **EditDistance:** This method calculate the distance matrix for provided input taking care of all the 3 cases- When there is a match penalty=0, When there’s a mismatch then all the 3 cases described in above example are considered.
* **FinalPattern:** This method displays the final mapping of the 2 strings based on the calculated distance matrix. Backtracking could be done using parent pointer storage but this implementation is more space efficient in which the penalty score is calculated based on the same logic which was used in matrix building.

**BOUNDARY CONDITIONS AND PENALTY EFFECT**

1. All the parameters from files are read in try- catch statement.
2. The gaps to be inserted for str1 and str2 in final alignment are maintained separately.
3. If the mismatch penalty and gap penalty generate same penalty then mismatch penalty is considered.
4. During backtracking for sequence alignment, separate parent pointers are not stored. It is implemented space efficiently considering the same logic for calculating penalty as used for matrix calculation.
5. All the cases and example samples given are tested.

EXAMPLES:

Input:

1 2 2

CAGTTA

CGA

Output:

Edit Distance: 9

C C 0

A - 2

G - 2

T - 4

T G 1

A A 0

Matrix computed:

0 4 6 8 10 12 14

4 0 4 6 8 10 12

6 4 1 4 7 9 11

8 7 4 2 5 8 9

When there are 3 continuous gaps the gap penalty for total of 3 gaps is calculates as q+3r where q=2, r=2 , So 2+3\*2=8 equal to 4+2+2=8. Mismatch penalty is 1, thereby total penalty is 8+1=9.

If gap open penalty is considered as 0 i.e.

Input:

1 0 2

CAGTTA

CGA

Output:

Edit Distance: 6

C C 0

A - 2

G G 0

T - 2

T - 2

A A 0

Matrix computed:

0 2 4 6 8 10 12

2 0 2 4 6 8 10

4 2 1 2 4 6 8

6 4 2 2 3 5 6

When gap open penalty is 0, then the alignment change as separating 3 gaps will yield same gap penalty i.e. 0+r –tree times or 3\*r, so in this case mismatch penalty is avoided and gap penalty is used to yield the optimum edit distance as 6.

2nd example (got from net)

Input:

1 0 2

GGGAATCACGAGAGCAGACAGATCACACAGGTTTATGGGTTCTACGACGAGTGTTTA

GGGAATCATGAGAGCAGACGATCACACAAGTTTATGGTTTCTATGATGAATGTTTA

Output:

Edit Distance: 8

G G 0

G G 0

G G 0

A A 0

A A 0

T T 0

C C 0

A A 0

C T 1

G G 0

A A 0

G G 0

A A 0

G G 0

C C 0

A A 0

G G 0

A A 0

C C 0

A - 2

G G 0

A A 0

T T 0

C C 0

A A 0

C C 0

A A 0

C C 0

A A 0

G A 1

G G 0

T T 0

T T 0

T T 0

A A 0

T T 0

G G 0

G G 0

G T 1

T T 0

T T 0

C C 0

T T 0

A A 0

C T 1

G G 0

A A 0

C T 1

G G 0

A A 0

G A 1

T T 0

G G 0

T T 0

T T 0

T T 0

A A 0

**REFERENCES:**

1. <http://en.wikibooks.org/wiki/Algorithm_Implementation/Strings/Levenshtein_distance>
2. <http://en.wikipedia.org/wiki/Levenshtein_distance>
3. <http://bioinfo.ict.ac.cn/~dbu/AlgorithmCourses/Lectures/Lec6-EditDistance.pdf>
4. <http://www.itu.dk/courses/AVA/E2005/StringEditDistance.pdf>
5. <https://cs.uwaterloo.ca/~binma/cs482/02_Alignment.pdf>
6. <http://introcs.cs.princeton.edu/java/assignments/sequence.html>

**DETAILS EXPLANATION OF EACH METHOD WORKING**

1. **Main**

* Input file name is provided as input argument which is accessed in program as args[0].
* The file content is read using java file system- FileReader and BufferReader. The file is assumed to consist of 3 lines as described above. All the penalties are individually saved in variables.
* Dynamic Programming approach is used for calculating edit distance. 2-D matrix according to the length of the strings is created i.e. longerString+1 X shorterString+1.
* This distance matrix, both the strings and all the penalties are passed to EditDistance function which returns the fully calculated distance matrix.
* The distance matrix is passed to FinalPattern function for printing the backtracking and final sequence alignment of the 2 strings.

1. **EditDistance**

* Input are both strings- str1(bigger length string) and str2(smaller length string), mismatch Penalty as mismatchPenalty, open gap penalty as gapPq, gap extension penalty as gapPr and final distance matrix as distance.
* Ist row and column od distance matrix is calculated as gapPq+i\*gapPr where ‘i' increasing with increasing gap. distance[0][0] is kept as 0.
* All the 3 cases for calculating the mismatch or gap penalty are calculated taking care of horizontal gap (gaph) and vertical gap(gapv) separately. While computing the value of distance[i][j], it has to be compared with following values:
* Case1: distance[i - 1][j] : This is the vertical gap condition which means the gap is to be considered in str1. It is calculated as distance[i-1][j]+(gapv==1? gapPq+gapPr:gapPr), i.e. if there is only 1 vertical gap then the computed gap penalty is distance[i-1][j]+gapPq+gapPr (q+1\*r) for first gap and if there are more than 1 continuous gaps then distance[i-1][i]+r i.e. only r is added for rest of the gaps.

For example: Str1 is computed to have 2 continuous gaps: A - - GTT then the penalty for 1st gap is q+r+distance[i-1][j] and for 2nd is only r + distance[i-1][j] and so on for rest of the gaps.

* Case2=distance[i][j - 1] +(gaph==1? gapPq+gapPr:gapPr): This is horizontal gap condition which means the gap to be considered in str2. It is calculated same as above, the only difference is the gaps are encountered with str2.
* Case3=distance[i - 1][j - 1] + ((str1.charAt(j - 1) == str2.charAt(i - 1)) ? 0 : mismatchPenalty): This is the mismatch penalty case. In this case the characters at those positions(i,j) are compared, if they are equal then distance[i][j]=distance[i-1][i-1] i.e. there is no penalty as there is a match found. Else if both the characters are different then the value of distance[i][j] = distance[i-1][j-1]+mismatchPenalty.
* All these 3 cases are compared for minimum value.
* If Case1 is minimum then distance[i][j] is computed as explained above and vertical distance(gapv) is incremented by 1 with horizontal distance(gaph) reset to 1.
* If Case2 is minimum then distance[i][j] is computed as explained above and horizontal distance(gaph) is incremented by 1 with vertical distance(gapv) reset to 1.
* Else if Case3 is minimum or the values of any of the minimum case is equal to Case3 then distance[i][j] is computed according to Case3 computation as explained above. Both gaph and gapv are reset to 1.
* Final distance matrix is returned with minimum edit distance value in distance[distance.length-1][distance[0].length-1].

1. **FinalPattern**
   1. This function is taking as input computed distance matrix, both the strings and all the penalties with output of final alignment of the 2 strings. In this function:
   * Edit distance is printed as distance[distance.length-1][distance[0].length-1].
   * Mapping of each character of both the strings is done by backtracking starting with the last elements of distance matrix and mapping till 1st element. The str1 and str2 character at i (let’s say s1)and j (let’s say s2) position are compared, Also diff=distance[i][j]-distance[i-1][j-1] is computed, Penalties are computed in the following way:

* If both the s1 and s2 are equal and diff==0- this means no penalty case when a match is found. So, penalty=0.
* If s1 is not equal to s2- diff is checked for mismatch or vertical or horizontal gap and penalties are calculated accordingly.
* This backtracking method could also be implemented by keeping track of parent pointer as well.