CS146: Spring 2023
Homework 5: Dynamic Programming
Due Friday, April 14, at 11:59PM
50 points

For this homework you will implement a dynamic program to compute the minimum edit distance between two strings. There are many ways to calculate the minimum edit distance, we will be using a method that allows for insertion, deletion, and substitution. Each insertion and deletion has a cost of 1. A substitution has a cost of 0 if the characters match and a cost of 2 if the characters don't match. See below for an example

Compute the Minimum Edit Distance Between Two Strings (50 points)

Algorithm Description

We are looking for the lowest cost (in terms of insertions, deletions, and mismatches) that can convert String A into String B. The example below illustrates how to transform the string TUESDAY into THURSDAY by inserting an H and substituting an R.

```
T - U E S D A Y
T H U R S D A Y
```

The minimum edit distance between two strings. Given two strings, the source String A of length n, and target String B of length m, we'll define D[i, j] as the edit distance between A[1..i] and B[1.. j], i.e., the first i characters of A and the first j characters of B. The edit distance between A and B is thus D[n,m]. Using this, we can create the following recurrence relation.

$$D[i,j] = \min \begin{cases} D[i-1,j]+1 & deletion \\ D[i][j-1]+1 & insertion \\ D[i-1,j-1]+\begin{cases} 0 \text{ if source}[i]=\text{target}[j] \\ 2 \text{ if source}[i]\neq\text{target}[j] \end{cases} \end{cases}$$

We can translate this recurrence relation into the following bottom up dynamic program

```
minEditDistance(source, target)
      n = source.length
      m = target.length
      Create a distance matrix D[n+1,m+1]
      D[0,0] = 0
      for i from 1 to n
            D[i,0]=D[i-1,0] + DELETE_COST
      for j from 1 to m
            D[0,j]=D[0, j-1] + INSERT_COST
      for i from 1 to n
            for j from 1 to m
                  D[i, j] = MIN( D[i-1, j] + DELETE_COST,
                                 D[i, j-1] + INSERT_COST,
                                 D[i-1, j-1] + SUB_COST)
      return D[n,m]
```

This is what the full distance matrix d would look like for the example above. The bold numbers are the path I take (corresponding to the transformation of Tuesday into Thursday above).

	-	T	H	U	R	S	D	A	Y
-	**0**	1	2	3	4	5	6	7	8
T	1	**0**	**1**	2	3	4	5	6	7
U	2	1	2	**1**	2	3	4	5	6
E	3	2	3	2	**3**	4	5	6	7
S	4	3	4	3	4	**3**	4	5	6
D	5	4	5	4	5	4	**3**	4	5
A	6	5	6	5	6	5	4	**3**	4
Y	7	6	7	6	7	6	5	4	**3**

Implementation Task (40 points)

To get started, import the starter file, EditDistance.java, into the distance package you create in a new Java Project. You CANNOT modify the inner Path class or the Enum Please do not change any of the method signatures in the EditDistance class, but you are free to add additional helper methods.

`public Path minEditDistance(String source, String target)`
You will implement the pseudocode described above with one one alternation. Not only do we want to return the minimum edit distance, we also want to know how we got there. To do this we will store Path Objects in our Distance matrix instead of just numbers. Path objects contain the characters from the source and target string (or '-' if there is an insertion or deletion), the minimum edit distance up until that point in the two strings, whether or not this cell received its value through an insertion, deletion, or substitution (match or mismatch), and a pointer to the next entry in the path (which entry in the matrix did I come from (above [deletion], left[insertion], or diagonal[(mis)match]).

Notes

- Sometimes the resulting min could have the same value for substitution, insertion, and deletion. In that case, you **MUST** prioritize substitutions, then deletions, then insertions in that order. If you don't do this, you will fail the test cases that I will run to grade your programs.

- You can assume that all strings consist of capital letters A-Z only (e.g., no lower case letters, no spaces, no special characters) and that both the source and target strings have at least length 1.

Testing (10 points)

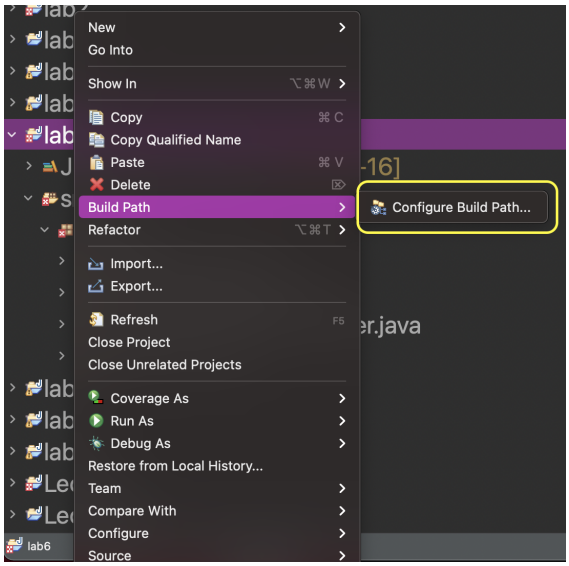
While you won't get points for writing test cases for your edit distance method, it is very important that you test your code before you submit it. You will receive 5 points for creating test cases and 5 points for passing your created test cases.

Submitting

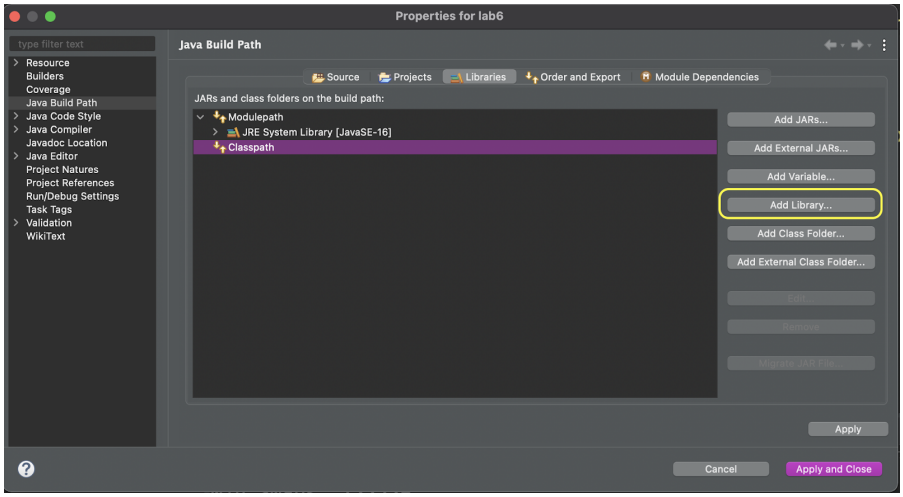
Export your project and upload it along with your solutions to the written questions. You can export your project as either a jar or a zip file. Make sure that the file you submit contains all your .java files in the package. If you upload a file that is missing any of the java source files you will get an automatic deduction of 5 points.

JUnit Setup

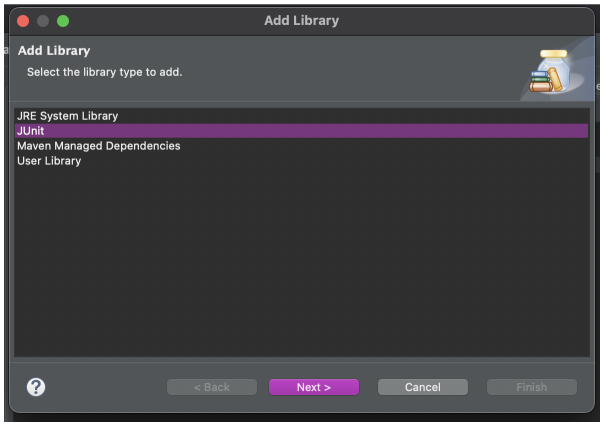
These instructions are for the Eclipse IDE. While you are welcome to use any IDE or programming environment, I will only provide instructions for Eclipse. To add the JUnit library to your build path, right-click on your homework project, look for Build Path and click on the Configure Build Path option.



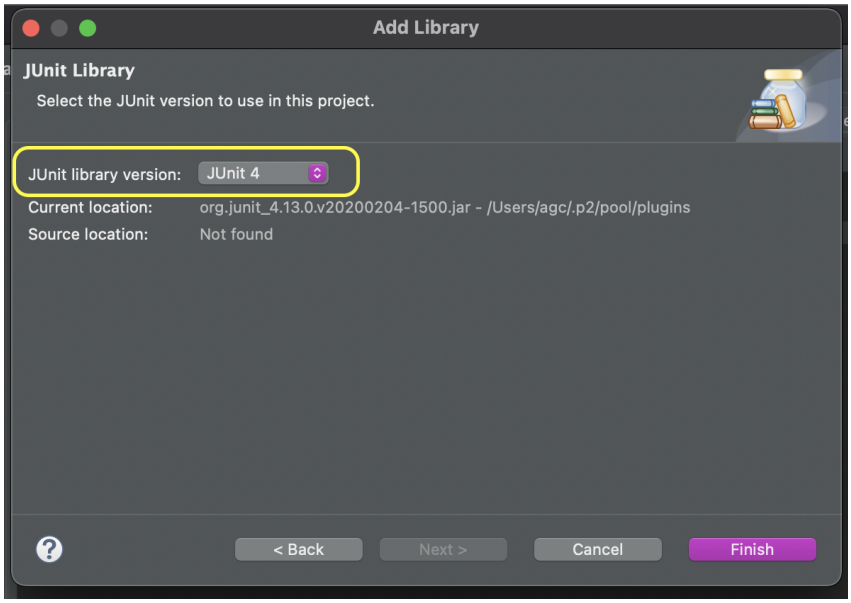
Highlight Classpath and click Add Library



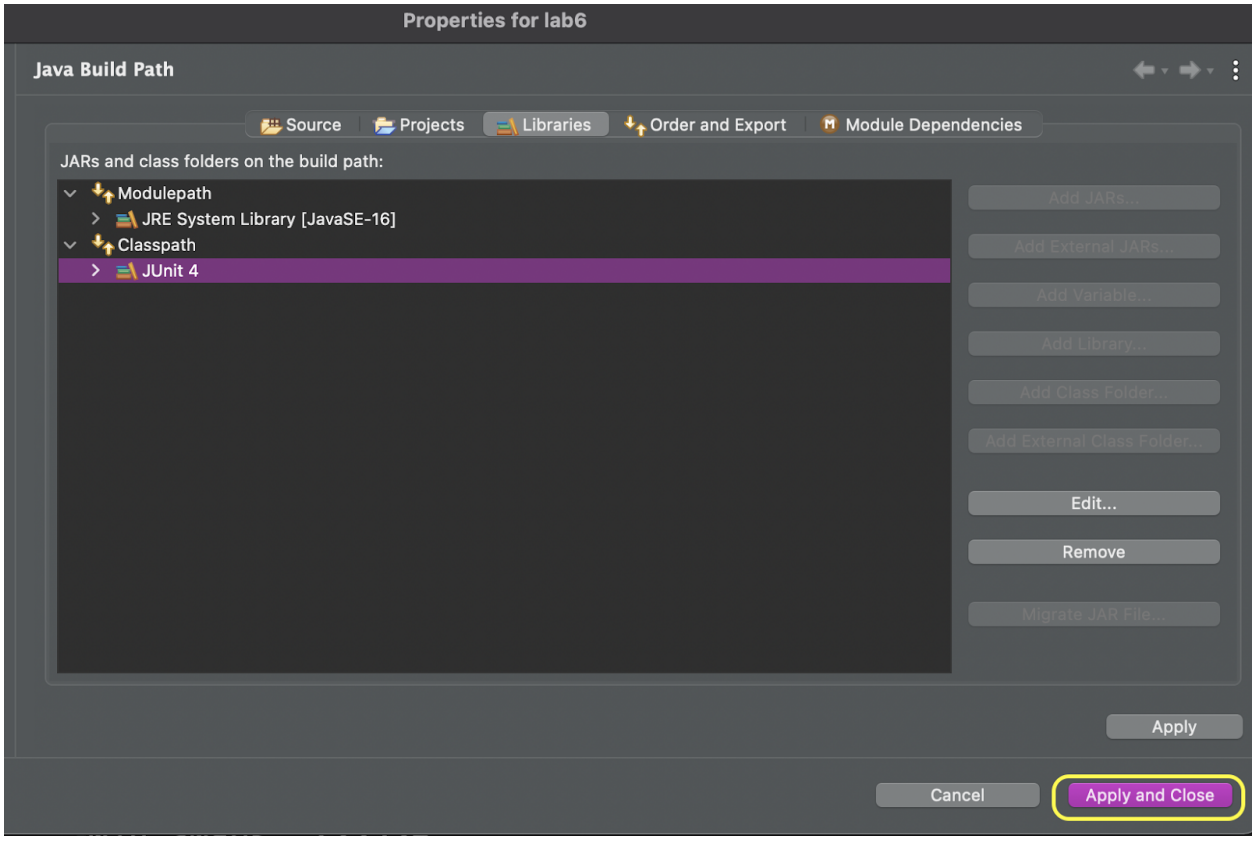
Select JUnit and click Next



Make sure you select JUnit 4. Then click Finish.



Your build path should look something like this. Click Apply and Close.



That should have gotten rid of any compilation errors you had when you originally copied over the files.