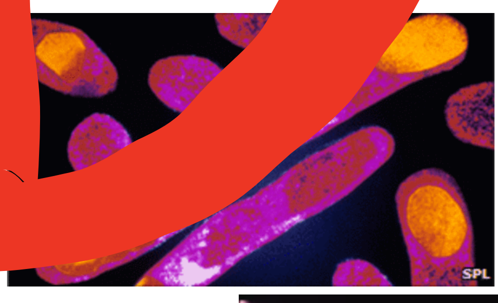
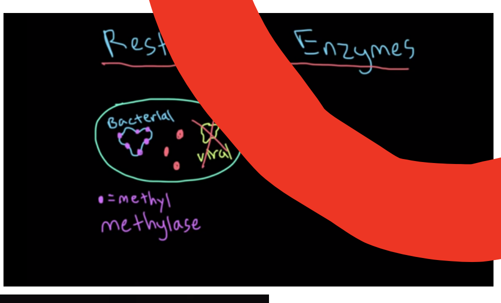
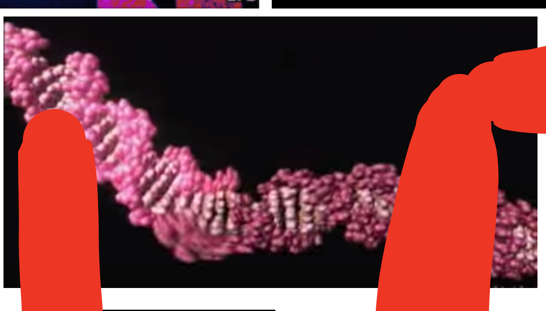
SIGNM T DESCRIPTION

s assignm u will be developing code to two important t enetics - restriction enzymes and DNA ng. The K cademy has a straight forw ction to restricti mes and how they can be used to get E ria to cre lin. The Howard Hughes I tute also has an e animation of the process of splicing v ction enzy





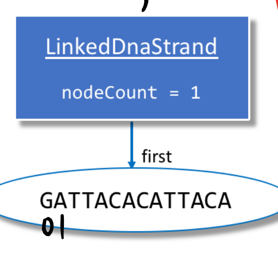


be c g a single Java class (edDnaStrand) that use list of nodes to represent a strand of DNA that s splic node in the list will a string of one or more s (A, C, G, or T). The class will be r ible for s such as appen d cutSplice(), whic al-world restriction enzyme processing. To r e proble le, we will be mak umber of assumptions th r an splittin riction enzyme. e also going to be repres

A ation of a c ice operation inkedDnaStrand (lin shown below. In the diagram, a L DnaStran list is shown ter it's constructor has be ed. The list is shown again after a call to c ice(). Note box in the m is an instance of Lin trand, and each oval is an instance of D e.

After Construction:

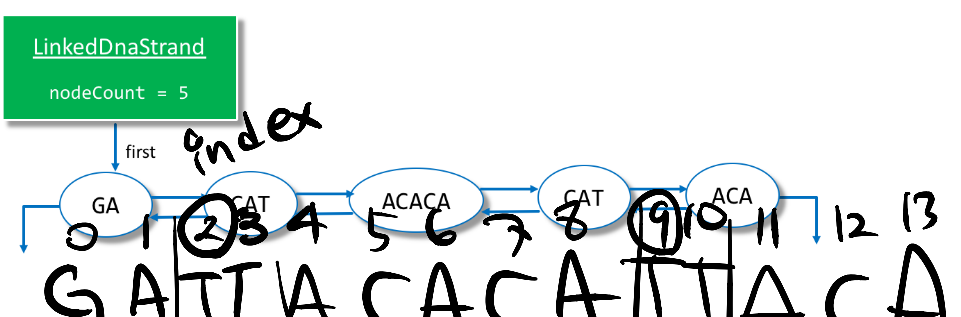
A LinkedDnaStrand wi DnaNode



After `cutSplice("TT", "CAT")` is called on the `LinkedDnaStrand` above:

A **new** `LinkedDnaStrand` is returned that has 5 `DnaNodes`. `TT` is the enzyme, and `CAT` is the splicee in this example.

For every instance of `TT` in the original DNA, a node with preceeding nucleotides and a node with `CAT` are added.



GETTING STARTED

1. Naming is critical in the tasks and requirements described below. If the names don't match those described below exactly, your project will not compile and can't be graded.

2. Right-click on the **JavaCoreTemplate** project and select Copy. Right-click in an empty area of the Package Explorer and select Paste.

   The project name must follow this pattern: **`{FLname}_DnaSplicing_{TERM}`**, where {FLname} is replaced by the first letter of your first name plus your last name, and {TERM} is the semester and year. E.g. if your name is Maria Marciano and it's Fall of 2025, your project name must be **`MMarciano_DnaSplicing_F25`**.

3. Create a new package in the **`src`** folder called **`dnasplicing`**.

4. **`DnaNode`**
   1. The **`LinkedDnaStrand`** class that you will be writing (see below) must use a **`DnaNode`** to hold each item in the linked list.
   2. Save **DnaNode.java** (right-click | Save Link As...) into your **`dnasplicing`** package folder.
   3. Refresh your project.
   4. **You do not modify `DnaNode.java`**.

5. **`DnaStrand`**
   1. The **`LinkedDnaStrand`** class that you will be writing (see below) must implement the **`DnaStrand`** interface.
   2. Save **DnaStrand.java** (right-click | Save Link As...) into your **`dnasplicing`** package folder.
   3. Refresh your project.
   4. You will need to read the comments above each method declaration in **`DnaStrand.java`** to see what each method is supposed to do.
   5. **You do not modify `DnaStrand.java`**. You will be adding your code to **`LinkedDnaStrand.java`** (see below).

6. **`LinkedDnaStrand`**
   1. This is the class that you must write.
   2. In the Package Explorer, select the **`dnasplicing`** package | New | Class.
   3. ~~In the wizard, set the name to LinkedDnaStrand. Then click the Add (interface) button~~, and search for DnaStrand. ~~Check the "Inherited abstract methods" box~~. ~~Click Finish.~~ ~~Eclipse will create the class and automatically add method stubs that meet the **`DnaStrand`** interface~~.
   4. You will need to read the comments above each method declaration in **`DnaStrand.java`** to see what each method is supposed to do.

7. **`SimpleDnaStrand`**
   1. **`SimpleDnaStrand`** is an array-based, **inefficient** implementation of **`DnaStrand`**. It is used by the grading script, **`LinkedDnaStrandTester`**, to verify that your code is giving correct output.
   2. Save **SimpleDnaStrand.java** (right-click | Save Link As...) into your **`dnasplicing`** package folder.
   3. Refresh your project.
   4. **You do not modify `SimpleDnaStrand.java`**.

8. **`LinkedDnaStrandTester`**.
   1. This is the grading script. It verifies that LinkedDnaStrand performs its operations correctly and efficiently.
   2. Create a new package in the **`src`** folder called **`sbccunittest`**. To be clear: **`sbccunittest`** should be a child of the **`src`** folder, not of the **`dnasplicing`** package.
   3. Save **LinkedDnaStrandTester.java** (right-click Save Link As...) to the **`sbccunittest`** package.
   4. **You do not modify `LinkedDnaStrandTester.java`**. And be sure that you don't accidentally allow Eclipse to modify it for you by accepting any Eclipse Quick Fixes that that it may offer that would change **`LinkedDnaStrandTester`**.

9. NOTE: **`LinkedDnaStrandTester`** should have compilation errors at this point. Not to worry - you'll be sorting those out below.

TASKS

1. Get the grading tester to work:
   1. Your overall goal is to pass all of the unit tests in **`LinkedDnaStrandTester`**. Right now, the tester won't even run because **`LinkedDnaStrand`** lacks a constructor that takes a DNA sequence (String) as a parameter. So hover over one of the constructor errors in **`LinkedDnaStrandTester`**, and have Eclipse add a **`LinkedDnaStrand(String)`** constructor to your class.
   2. This should clear all of the compilation errors in your project. If not, debug any remaining problems.
   3. Run the grade script by right-clicking on **`LinkedDnaStrandTester`** | Run As.. | JUnit Test. All of the tests should fail, and you should get 0 points... Not to worry, though, your first points are close at hand.

2. Get **`testNewDnaStrandSimple`**() to pass:
   1. **`testNewDnaStrandSimple`**() ensures that your constructor properly creates a linked list with one node containing the DNA sequence that was passed in. Figure 1 above shows what **`LinkedDnaStrand`**'s linked list might look like when instantiated like this: **`LinkedDnaStrand l = new LinkedDnaStrand("GATTACACATTACA");`**
   2. Edit **`LinkedDnaStrand`**'s constructor so that it:
      1. Creates a **`DnaNode`**, initializing it with the DNA sequence that was passed into **`LinkedDnaStrand()`**.
      2. Make the **`LinkedDnaStrand`**'s first node pointer point to the new **`DnaNode`**.
      3. Set a variable that keeps track of the node count to 1.
   3. In order to pass **`testNewDnaStrandSimple`**, you'll need to modify **`getFirstNode()`** and **`getNodeCount()`** to return their respective values.
   4. Run **`LinkedDnaStrandTester`**. Did you get **`testNewDnaStrandSimple`**() to pass? ~~If not, review the steps above and look for any other bugs that might be in your code.~~

3. To get **`testNewDnaStrand()`** to pass, you'll need to add a **`toString()`** method to your class. The **`toString()`** method simply needs to return entire DNA sequence represented by a **`DnaStrand`**.

4. It is recommended, but not required, that you work on passing the tests in the order shown in the **Scoring** section below. I.e. once you get **`testNewDnaStrand()`** to pass, work on getting **`testAppend()`** to pass.

HINTS / SUGGESTIONS

1. **`cutSplice()`** will only be called when the linked list has just one node, and it will only be called once for a **`DnaStrand`**. This means that you do not need to worry about searching for enzyme matches across node boundaries.

2. Feel free to add any additional methods, including constructors, that you need to implement the program.

3. Be sure to read the comments for each method in **`DnaStrand`** - they describe what your methods are supposed to do.

4. Also read the comments for the various tests in **`LinkedDnaStrandTester`**. They will help you understand what the test is trying to verify. It can also be very useful to read or step through the test code itself, as this will give the deepest insight into how your code is being graded.

5. **`StringUtils`** has a **`reverse()`** method that returns the reverse of a **`String`**.

EXTRA CREDIT

The **`createReversedDnaStrand()`** method creates a new **`DnaStrand`** whose DNA sequence is a reverse of the original. So if a **`DnaStrand`** contains "CGAT", then **`createReversedDnaStrand()`** would return a **`DnaStrand`** that contains "TAGC". You will receive extra credit if the returned **`LinkDnaStrand`** contains a reversed version of both the DNA sequence and the nodes. E.g. if a **`LinkedDnaStrand`** contained this linked list:

```
first -> "TT" -> "TTAAGG" -> "CC" -> null
```

then its reverse would be a new **`LinkedDnaStrand`** containing the following linked list:

```
first -> "CC" -> "GGAATT" -> "TT" -> null
```

SCORING

`testNewDnaStrandSimple` - 2 pts

`testNewDnaStrand` - 5 pts

`testAppend` - 5 pts

`testAppendRandom` - 5 pts

`testGetNucleotideCount` - 3 pts

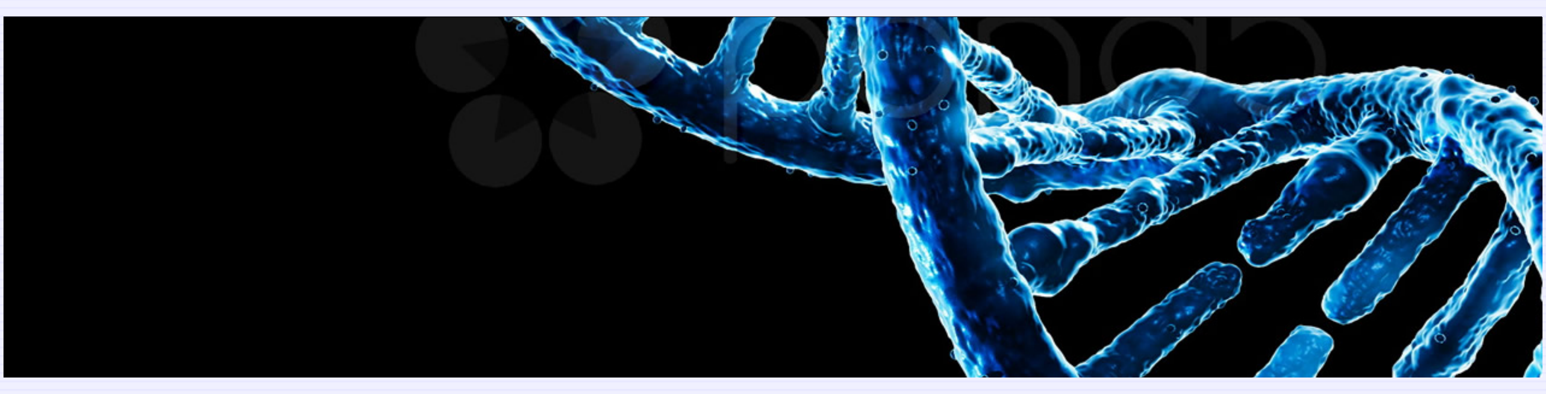
`testCutSplice` - 5 pts

`testSpliceInsulinIntoEcoli` - 10 pts

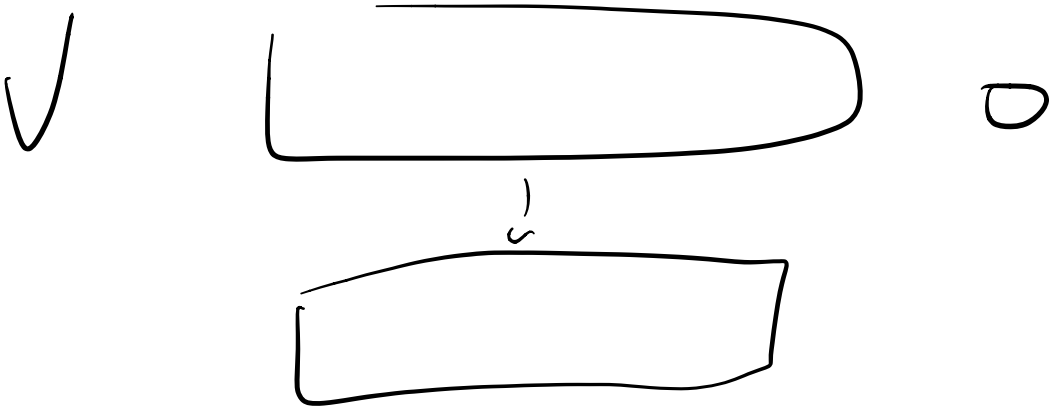
`testSplicingTime` - 5 pts

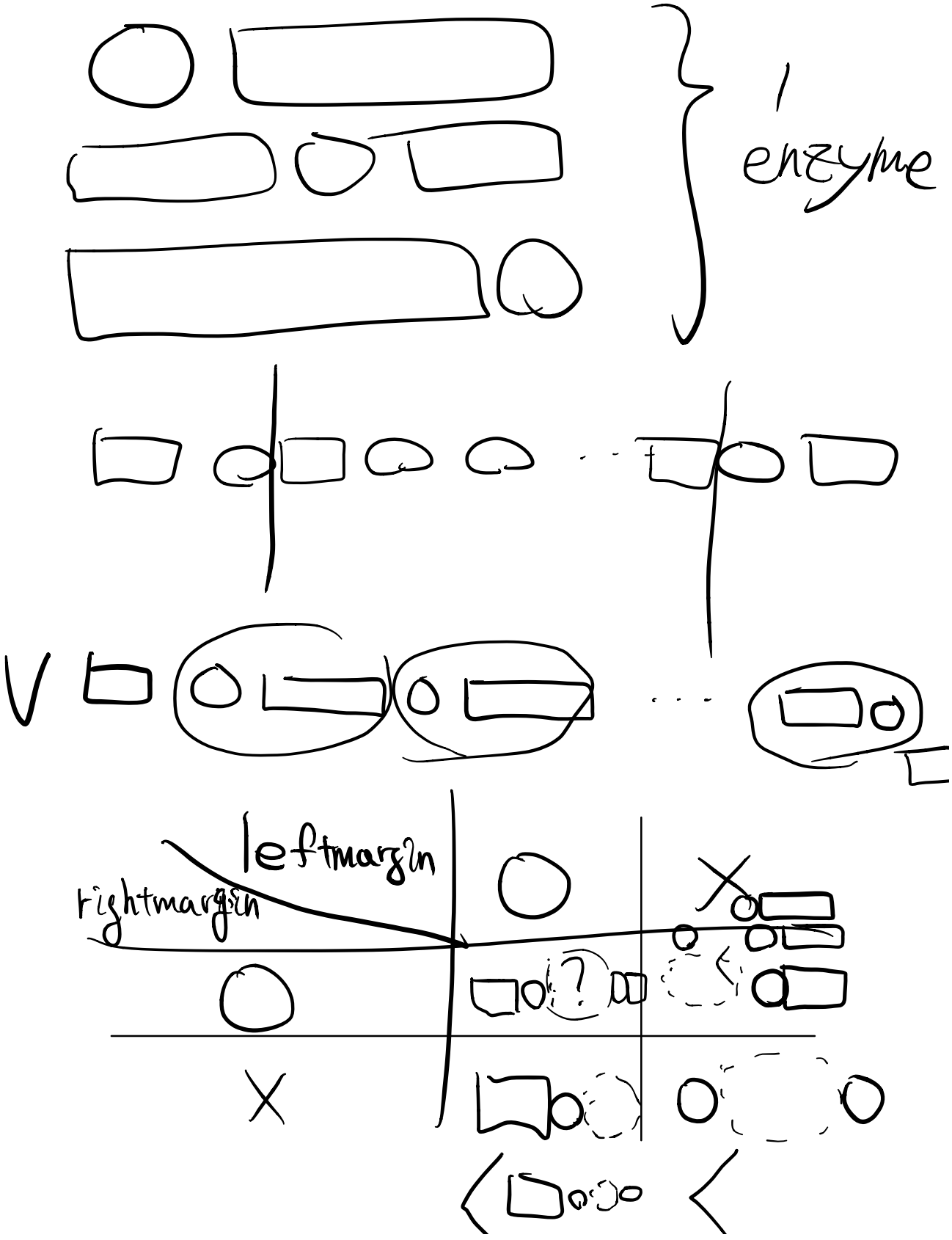
`testReverse` - 10 pts with possibility of 5 extra credit points

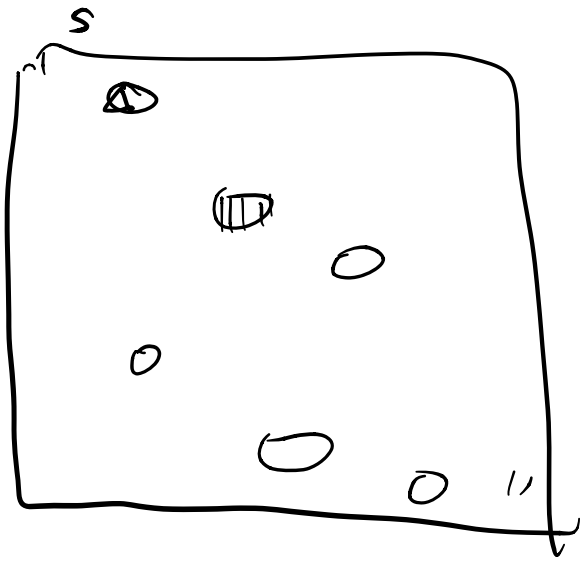
`testPmd` - 5 pts

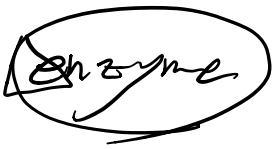


Str. startsWith (




1
enzyme
leftmargin
rightmargin





charAt(0) ~ charAt(enzymelen-1)

charAt(i) — △ replace 인지확인

replace.

~△ deposit. 맞아요.

charAt(i) — △ replace 인지확인

~△ deposit. 맞아요.

s.charAt(i) ≤ j ≤ s.charAt(i+enzymelen-1) :(j = 0; j < enzymelen; j++)

enzyme

char(i+j) == enzyme.char(j) count++;

< != — break;

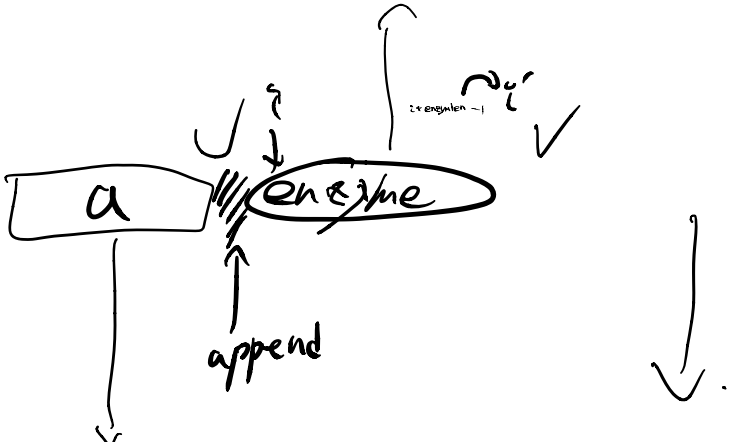
fullcount = enzymelen

```
int slen = s.length();
int enzymelen = enzyme.length();

for ( int i=0 ;  i < slen; i++ ) {
    if(  s.charAt(i) == enzyme.charAt(0) ) {
        for (int j=1 ; j<enzymelen; j++){
            if( s.char(i+j) == enzyme.char(j) ){ count++;}
            else{
                break; < temp.append (enzyme.substring( 0, j ));
            }
            if( count == enzymelen){
                // replace    l.append (temp);   temp = "";
                l.append ( splicee );

                i += (enzymelen -1);
            }
        }
    }
    else {
     temp += s.charAt(i) ;
    }
    if(  i ==slen - 1 ) {
            l . append (temp);

    }
}
```



```
}


@Override
public DnaStrand cutSplice(String enzyme, String splicee) {
        var l = new LinkedDnaStrand("");
        String s = first_node.dnaSequence;

        if (node_count == 1) {
                int slen = s.length();
                int enzymelen = enzyme.length();
                int count = 0;
                var temp = new StringBuilder("");
                for (int i = 0; i < slen; i++) {
                        if (s.charAt(i) == enzyme.charAt(0)) {
                                for (int j = 1; j < enzymelen && i + j < slen; j++) {
                                        if (s.charAt(i + j) == enzyme.charAt(j)) {
                                                count++;
                                        } else {
                                                count = 0;
                                                break;
                                        }
                                        if (count == enzymelen - 1) {
                                                count = 0;
                                                if (!temp.toString().equals("")) {
                                                        l.append(temp.toString());
                                                }
                                                temp = new StringBuilder("");
                                                l.append(splicee);
                                                i += enzymelen - 1;
                                        }
                                }
                        } else {
                                temp.append(s.charAt(i));
                        }
                        if (i == slen - 1 && !temp.toString().equals("")) {
                                l.append(temp.toString());
                        }
                }
                return l;
        } else {
                return null;
        }
}
```

ecoliPart="GCGGCTGGCGACGTATTCCAG"

enzyme GAATTC

GCGGCTGGCGAA|ACGTATTCCA
GCGGCTGGCGA|CGTATTCCAG
0 1 2 3 4 5 6 7 8 9 10 11