Assignment 2 - Shortest Common Superstring Problem

Liverpool University

YUEFENG LIANG

201350099

24th, March

1. Problem

The problem is about giving a list of strings and outputting a single string, called SuperString, including all of them. We could do this by concatenating all strings simply, but the added requirement of this assignment is finding the shortest strings which contains them. A few different things need to be considered when solving this problem.
. Repeated strings
. Overlapping strings

2. Approaches to find overlap part

2.1. Naïve approach

We could see that all the strings are in the same length, because of this using a sliding method may easy, where each of the words is separated to prefix and suffix then compared with others. For example, to comparing the strings abc and bcd we would simply separate abc to prefix, [a,ab,abc] and suffix, [c,bc,abc] and bcd would be separated into prefix, [b,bc,bcd] and suffix, [d,cd,bcd]. Using this we could easily compare the suffix part of abc to the prefix part of bcd or the suffix part of bcd to the prefix part of abc.

Each string has m different child nodes, m representing the length of each string. We need to do n comparison and each comparison taking m time. Thus, the comparisons of all child nodes would be taking nm^2 time. Since this is only the cost of one node comparing to all other nodes, this process needs to be repeated n times, leading the time complexity of n^2m^2. Finally considering of the time generating these child nodes, the cost of this algorithm is $n^2m^2 + nm^2$.

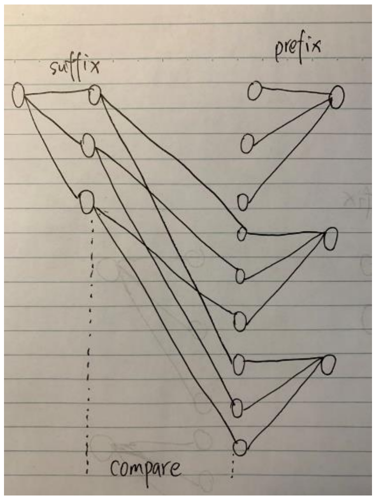
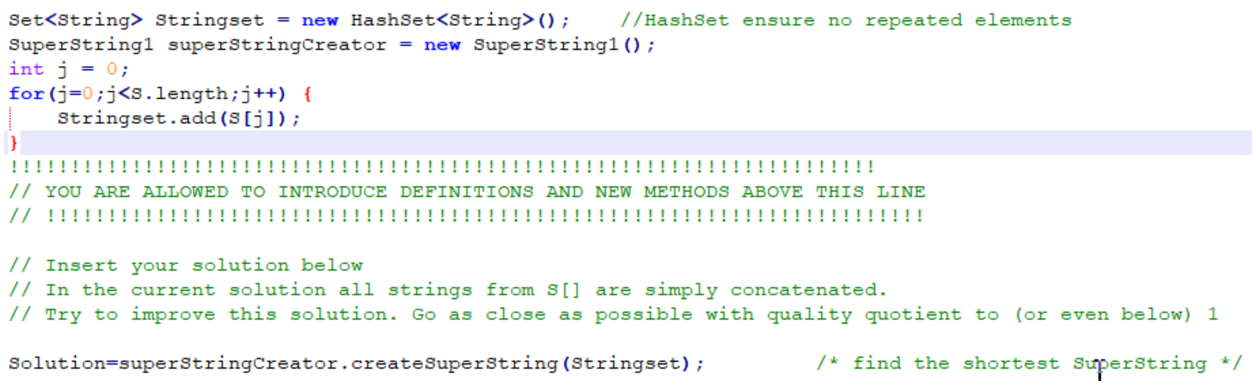


Figure1. example of algorithm

3. My implementations

Firstly, we would use HashSet to clean the total strings and then use superStringCreator.createSuperString to create all possible SuperStrings.



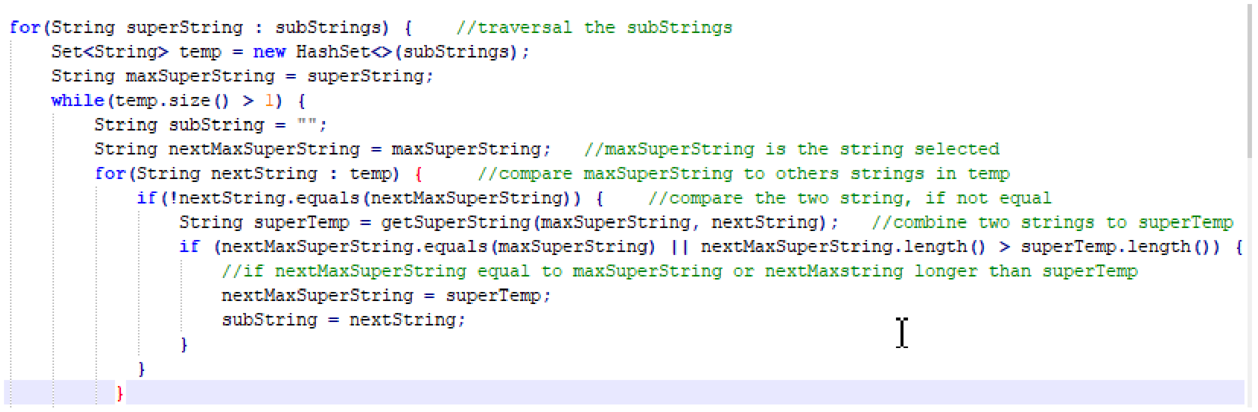
```
Set<String> Stringset = new HashSet<String>();    //HashSet ensure no repeated elements
SuperString1 superStringCreator = new SuperString1();
int j = 0;
for(j=0;j<S.length;j++) {
    Stringset.add(S[j]);
}
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
// YOU ARE ALLOWED TO INTRODUCE DEFINITIONS AND NEW METHODS ABOVE THIS LINE
// !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

// Insert your solution below
// In the current solution all strings from S[] are simply concatenated.
// Try to improve this solution. Go as close as possible with quality quotient to (or even below) 1

Solution=superStringCreator.createSuperString(Stringset);         /* find the shortest SuperString */
```

Figure2. Clean the data

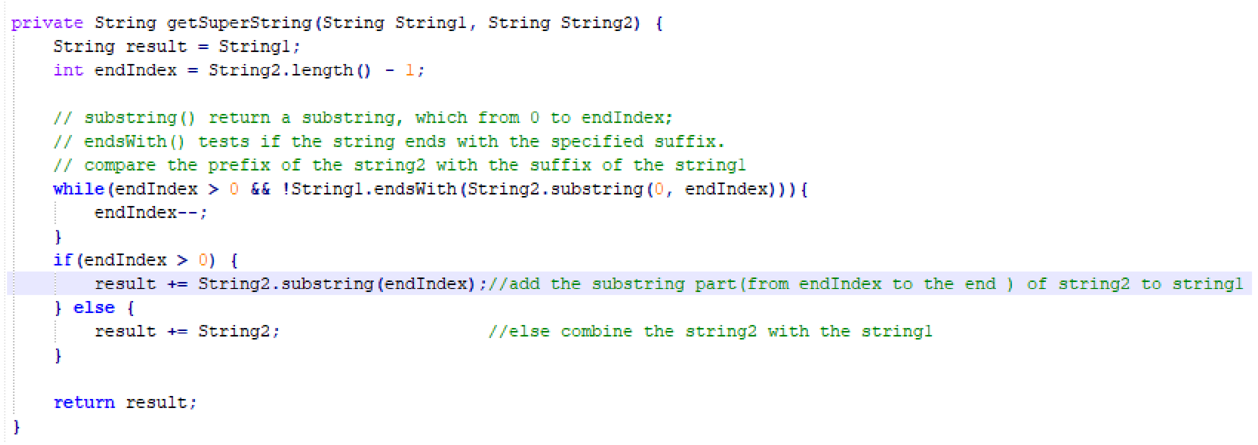
In the createSuperString method, use for loop to traversal all strings and compare each string with others. getSuperString input two strings and output a superstring contain two strings.



```
for(String superString : subStrings) {    //traversal the subStrings
    Set<String> temp = new HashSet<>(subStrings);
    String maxSuperString = superString;
    while(temp.size() > 1) {
        String subString = "";
        String nextMaxSuperString = maxSuperString;   //maxSuperString is the string selected
        for(String nextString : temp) {     //compare maxSuperString to others strings in temp
            if(!nextString.equals(nextMaxSuperString)) {     //compare the two string, if not equal
                String superTemp = getSuperString(maxSuperString, nextString);   //combine two strings to superTemp
                if (nextMaxSuperString.equals(maxSuperString) || nextMaxSuperString.length() > superTemp.length()) {
                    //if nextMaxSuperString equal to maxSuperString or nextMaxstring longer than superTemp
                    nextMaxSuperString = superTemp;
                    subString = nextString;
                }
            }
        }
```

Figure3. Select one string comparing with others

Then, in getSuperString method, we would compare the suffix part of the string1 with the prefix part of the string2 and cut the same part, then combine the remain part of string2 to string1.



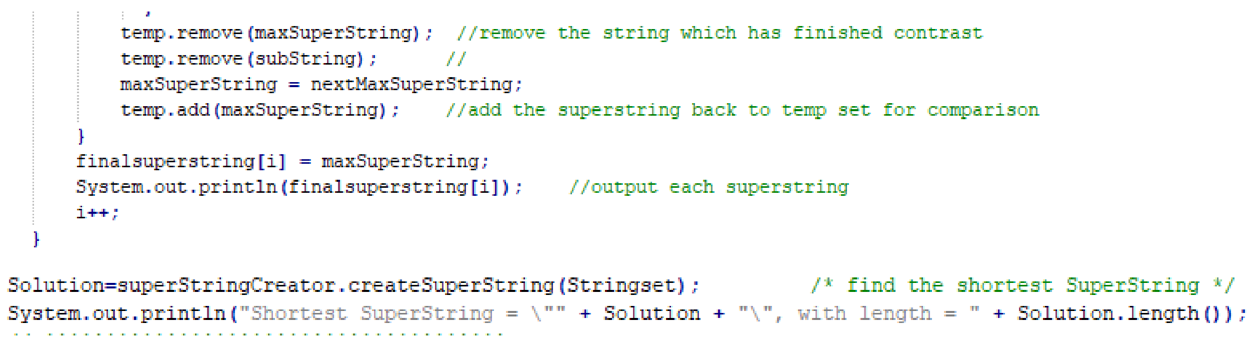
```java
private String getSuperString(String String1, String String2) {
    String result = String1;
    int endIndex = String2.length() - 1;

    // substring() return a substring, which from 0 to endIndex;
    // endsWith() tests if the string ends with the specified suffix.
    // compare the prefix of the string2 with the suffix of the string1
    while(endIndex > 0 && !String1.endsWith(String2.substring(0, endIndex))){
        endIndex--;
    }
    if(endIndex > 0) {
        result += String2.substring(endIndex);//add the substring part(from endIndex to the end ) of string2 to string1
    } else {
        result += String2;                    //else combine the string2 with the string1
    }

    return result;
}
```

Figure4. Compare two strings and return a superstring

Finally, print all possible superstrings and pick out the shortest superstring as the solution to output.



```java
            temp.remove(maxSuperString);  //remove the string which has finished contrast
            temp.remove(subString);      //
            maxSuperString = nextMaxSuperString;
            temp.add(maxSuperString);    //add the superstring back to temp set for comparison
        }
        finalsuperstring[i] = maxSuperString;
        System.out.println(finalsuperstring[i]);    //output each superstring
        i++;
    }

Solution=superStringCreator.createSuperString(Stringset);         /* find the shortest SuperString */
System.out.println("Shortest SuperString = \"" + Solution + "\", with length = " + Solution.length());
```

4. Result

SuperSring1

Solution is **“bbabbababbabbabababbababbab”**, with the length is 25;

The quality quotient= **1.0**;

SuperString2

Solution is **“acbdacbadcbacdbacabdcabcdabcadbca”**, with the length is 33;

The quality quotient= **1.0**

SuperString3

Solution is **“00000000110000010000000010000000000010000100010000000000000001000001100001000”**, the length is 77

The quality quotient= **0.77**

(Because each time the string provided randomly, the solution is also different, but the quality quotient is always less than 1.0)