Topic 6: Basic String Processing

Competitive Programming I (Fall 2019)

Ninghui Li (Purdue University)

Leet: Count Binary Substrings (Required)

Give a string s, count the number of non-empty (contiguous) substrings that have the same number of 0's and 1's, and all the 0's and all the 1's in these substrings are grouped consecutively.

Substrings that occur multiple times are counted the number of times they occur.

```
Input: "00110011"     Output: 6
0011, 01, 1100, 10, 0011, 01
```

How to compute this?

```
s.length will be between 1 and 50,000.
s will only consist of "0" or "1" characters.
```

Count Binary Substrings: Analysis

- Approach one:
  - Count sizes of blocks of consecutive 0’s and 1’s. Given A consecutive, followed by B consecutive. There are min(A,B) number of new such strings.
- Approach two:
  - For each index, either zero or one string ending at the index satisfies the condition.
  - Keep track of length of consecutive block ending at current index,
  - And the length of the previous block

Approach 1 in Java (Buggy version)

```
public int countBinarySubstrings(String s) {
   int sum = 0, a = 0, b = 1, i = 1, n=s.length();
   while (i<n) {
     while (i<n && s.charAt(i)==s.charAt(i-1)) {
       i++;  b++;
     }
     sum += Math.min(a,b);
     a = b;
     b = 1;
     i++;
   }
   return sum;
}
```

Where is the bug?

Execute the code on input “01”.

Undercount by 1 when string ends with one 0 or 1.

How to fix it?

Approach 1 (Ugly Bug Fix)

```
public int countBinarySubstrings(String s) {
  int sum = 0, a = 0, b = 1, i = 1;
  while (i<s.length()) {
    while (i<s.length() && s.charAt(i)==s.charAt(i-1)) {
      i++;  b++;
    }
    if (i >= s.length())          break;
    sum += Math.min(a,b);
    a = b;    b = 1;     i++;
  }
  sum += Math.min(a,b);
  return sum;
}
```

Approach 1 (Less Ugly Bug Fix)

```
public int countBinarySubstrings(String s) {
  int sum = 0, a = 0, b = 1, i = 1, n = s.length();
  while (i<=n) {
    while (i<n && s.charAt(i)==s.charAt(i-1)) {
      i++;  b++;
    }
    sum += Math.min(a,b);
    a = b;
    b = 1;
    i++;
  }
  return sum;
}
```

Correct, but unnatural logic.

Approach 1 (Less Ugly Bug Fix)

```
public int countBinarySubstrings(String s) {
  int sum = 0, a = 0, b = 1, i = 1, n = s.length();
  while (i<=n) {
    while (i<n && s.charAt(i)==s.charAt(i-1)) {
      i++;  b++;
    }
    sum += Math.min(a,b);
    a = b;
    b = 1;
    i++;
  }
  return sum;
}
```

These are duplicating logic, and duplicating logic is bad.

The code “b=1,i=1” is trying to take care of the first character before the loop starts. This should be avoided, because it is creating corner cases.

Approach 1 in Java (Better alternative)

```
public int countBinarySubstrings(String s) {
  int sum = 0, a = 0, b = 0, i = 0, n=s.length();
  while (i<n) {
    b=1;  i++;
    while (i<n && s.charAt(i)==s.charAt(i-1)) {
      b++;  i++;
    }
    sum += Math.min(a,b);
    a = b;
  }
  return sum;
}
```

Approach 2 in C++

```
// count number of valid substrings (0 or 1) ending at each c

  int countBinarySubstrings(string s) {
    char prev = '$';
    int num1 = 0, num2 = 0, cnt = 0;
    for (char c : s) {
        if (prev == c)     num2++;
        else               { num1 = num2;  num2 = 1; }
        if (num2 <= num1) cnt++;
        prev = c;
    }
    return cnt;
  }
```

Clever use of another value to handle starting char.

Lesson from Count Binary Substrings

Different insights lead to different ways to code.

Avoid duplicating logic.

Avoid handing the first element differently. Use sentinel value if needed.

LeetCode: Longest Common Prefix

Write a function to find the longest common prefix string amongst an array of strings. If there is no common prefix, return an empty string "".

```
Input: ["flower","flow","flight"]      Output: "fl"

Input: ["dog","racecar","car"]         Output: ""
```

All given inputs are in lowercase letters a-z.

Verion 1: Buggy Java Code

```
public String longestCommonPrefix(String[] strs) {
  int n = strs.length;
  if (n == 0)  return "";
  for (int i=0; i<strs[0].length; i++) {
    char ch = strs[0].charAt(i);
    for (int j=1; j<n; j++) {
      if (ch != strs[j].charAt(i))
        return strs[0].substring(0,i);
    }
  }
  return strs[0];
}
```

What is the bug?

Verion 2: Java Code (Somewhat Ugly Bug Fix)

```
public String longestCommonPrefix(String[] strs) {
  int n = strs.length, m = Integer.MAX_VALUE;
  if (n == 0)  return "";
  for (int j=0; j<n; j++)  m=Math.min(m,strs[j].length());
  for (int i=0; i<m; i++) {
    char ch = strs[0].charAt(i);
    for (int j=1; j<n; j++) {
      if (ch != strs[j].charAt(i))
        return strs[0].substring(0,i);
    }
  }
  return strs[0].substring(0,m);
}
```

Can we get rid of the red code?

Version 3: More Succinct

```
public String longestCommonPrefix(String[] strs) {
  if (strs.length == 0)  return "";
  for (int i=0;  ; i++) {
    for (int j=0; j<strs.length; j++) {
      if (i >= strs[j].length()
            || strs[j].charAt(i) != strs[0].charAt(i))
        return strs[0].substring(0,i);
    }
  }
}
```

Note that there is no special logic for strs[0].
Why is it okay even if we do not explicitly compute the minimum string length?

Java String Manipulation

Method 1: Using + for strings

- Each String object is immutable. String s = s +”a” would create a new String object.
- What is the complexity of the following Java code?
  - `String s=“a”;  for (i=0; i<N; i++)  s+=“a”;`

Method 2: Use StringBuilder

- Uses a variable length character array in implementation.
- What is the complexity of the following Java code?
  - `StringBuffer s = new StringBuffer();`
  - `for (i=0; i<N; i++)  s.append(“a”);`

Method 3: Use char[], and convert to String after processing.

LeetCode: Groups of Special-Equivalent Strings

Two strings S and T are *special-equivalent* if after any number of *moves*, S == T.
A *move* consists of choosing two indices i and j with i % 2 == j % 2, and swapping S[i] with S[j].

Given an array A of strings, a *group of special-equivalent strings from A* is a non-empty subset S of A such that any string not in S is not special-equivalent with any string in S. Return the number of groups of special-equivalent strings from A.

```
Input: ["abc","acb","bac","bca","cab","cba"]   Output: 3

Explanation: 3 groups ["abc","cba"], ["acb","bca"], ["bac","cab"]
```

Groups of Special-Equivalent Strings

How to check whether two strings are special-equivalent?

- The sub-sequences of all odd (and even) positions of two strings contain the same characters.

How to check that two strings contain the same characters?

- After sorting, they are the same. (Simple code.)
- Count numbers of each characters are the same. (Complicated)

How to compute how many different groups are there?

- Adding to a set and then get size. (Simple code)

Groups of Special-Equivalent Strings (Java)

```
public int numSpecialEquivGroups(String[] A) {
  HashSet<String> strs = new HashSet<String>();
  for (int i=0; i<A.length; i++) {
    int n = A[i].length(), n1 = (n+1)/2, n2 = n - n1;
    char[] s = new char[n];     // new string
    for (int j=0; j<n1; j++) s[j] = A[i].charAt(2*j);
    for (int j=0; j<n2; j++) s[n1+j] = A[i].charAt(2*j+1);
    Arrays.sort(s, 0, n1);
    Arrays.sort(s, n1, n);
    strs.add(new String(s));
  }
  return strs.size();
}
```

Groups of Special-Equivalent Strings (C++)

```
int numSpecialEquivGroups(vector<string>& A) {
    int n = A.size();
    string s[2];
    unordered_set<string> myset;
    for(int i = 0; i < n; i++) {
        s[0].clear();         s[1].clear();
        for(int j = 0; j < A[i].length(); j++) {
            s[j % 2] += A[i][j];
        }
        sort(s[0].begin(), s[0].end());
        sort(s[1].begin(), s[1].end());
        myset.insert(s[0] + s[1]);
    }
    return myset.size();
}
```

LeetCode: Greatest Common Divisor of Strings

For strings S and T, we say "T divides S" if and only if S = T + ... + T (T concatenated with itself 1 or more times)

Return the largest string X such that X divides str1 and X divides str2.

```
Input: str1 = "ABABAB", str2 = "ABAB"

Output: "AB"
```

Greatest Common Divisor of Strings

Approach one:

- Observe that the length of the gcd of the string must be the gcd of the lengths of string. Compute gcd d, and check whether both strings can be formed by repeating substring of length d.

Approach two:

- Recursively compute the gcd, similar to how to compute gcd of two numbers.

C++ Version (Using gcd)

```
string gcdOfStrings(string str1, string str2) {
    int ans = __gcd(str1.length(), str2.length());
    for (int i = 0; i < str1.size(); i++) {
        if (str1[i] != str1[i % ans]) {
            return "";
        }
    }
    for (int i = 0; i < str2.size(); i++) {
        if (str2[i] != str1[i % ans]) {
            return "";
        }
    }
    return str1.substr(0, ans);
}
```

Java Version One (Without using gcd of integers)

```
public String gcdOfStrings(String str1, String str2) {
  while (true) {
    if (str1.length() < str2.length()) {
      String str3 = str1;
      str1 = str2;     str2 = str3;
    }
    if (! str1.startsWith(str2))
      return "";
    if (str1.length() == str2.length())
      return str1;
    str1 = str1.substring(str2.length());
  }
}
```

What is the time complexity of this code?

Java Version Two (Using Char Array, no gcd, complex)

```
public String gcdOfStrings(String str1, String str2) {
  char[] c1 = str1.toCharArray(), c2 = str2.toCharArray();
  int b1=0, e1=c1.length, b2=0, e2=c2.length;
  while (true) {
    int nb1=b1, nb2=b2;
    while (nb1<e1 && nb2<e2 && c1[nb1]==c2[nb2]) {
      nb1++;   nb2++;
    }
    if (nb1<e1 && nb2<e2) return "";
    else if (nb1<e1)      b1 = nb1;
    else if (nb2<e2)      b2 = nb2;
    else                  return new String(c1, b1, e1-b1);
  }
}
```

What is the time complexity of this code?

Java version using gcd (Buggy version)

```
public int gcd(int a, int b) {  // only positive arguments
  if (b == 0) return a;
  return gcd(b, a%b);
}
public String gcdOfStrings(String str1, String str2) {
  int n1 = str1.length(), n2 = str2.length();
  int d = gcd(n1, n2);
  for (int i=d; i<n1; i++)
    if (str1.charAt(i) != str1.charAt(i%d))  return "";
  for (int i=d; i<n2; i++) {
    if (str2.charAt(i) != str2.charAt(i%d))  return "";
  return str1.substring(0,d);
}
```

Where are the bugs?

Java version using gcd (Correct version)

```
public int gcd(int a, int b) {  // only positive arguments
  if (b == 0) return a;
  return gcd(b, a%b);
}
public String gcdOfStrings(String str1, String str2) {
  int n1 = str1.length(), n2 = str2.length();
  int d = gcd(n1, n2);
  for (int i=d; i<n1; i++)
    if (str1.charAt(i) != str1.charAt(i%d))  return "";
  for (int i=0; i<n2; i++) {
    if (str2.charAt(i) != str1.charAt(i%d))  return "";
  return str1.substring(0,d);
}
```

LeetCode: String Compression

Given an array of characters, compress it in-place. The length after compression must always be smaller than or equal to the original array. Every element of the array should be a character (not int) of length 1. After you are done modifying the input array in-place, return the new length of the array. Use O(1) extra memory.

```
Input:    ["a","b","b","b","b","b","b","b","b","b","b","b","b"]

Result:   ["a","b","1","2"]
```

String Compression (Slightly tortured logic)

```
public int compress(char[] chars) {
  char ch = 0;
  int len = 0, count = 0, n = chars.length;
  for (int i=0; i<=n; i++) {
    if (i<n && chars[i] == ch) { count++; }
    else {  // new char
      if (count > 1) {
        String c = String.valueOf(count);
        for (int j=0; j<c.length(); j++)  chars[len++]=c.charAt(j);
      }
      if (i == n)    break;
      ch = chars[i];   chars[len++] = ch;      count = 1;
    }
  }
  return len;
}
```

String Compression (Better Logic: Look Ahead Loop)

```
public int compress(char[] chars) {
  int len = 0, n = chars.length;
  for (int i=0; i<n; ) {
    char ch = chars[i];
    chars[len] = ch;
    int count = 1;
    while (i+count < n && ch == chars[i+count])  count++;
    if (count > 1) {
      String c = String.valueOf(count);
      for (int j=0; j<c.length(); j++)  chars[len++] = c.charAt(j);
    }
    i += count;
  }
  return len;
}
```

USACO 2015 February, Bronze 1. Censoring (Bronze)

Given a string S of length at most 10^6 characters, and a string T of length <= 100 characters. Remove all occurences of T in S as follows. Finds the _first_ occurrence of T in S and deletes it. Then repeats the process again, deleting the first occurrence of T again, continuing until there are no more occurrences of T in S. Note that the deletion of one occurrence might create a new occurrence of T that didn't exist before.

Example input:

```
Whatthemomooofun
moo
```

Example output:

```
whatthefun
```

Simple Logic in Java

```
String s = nextStr(), t = nextStr();
int k;
while ((k=s.indexOf(t)) > 0) {
  String ns = s.substring(0,k);
  ns += s.substring(k+t.length(),s.length());
  s = ns;
}
out.println(s);
```

What is the time complexity? Will this pass with the specified parameters?

Censoring (Fast without new algorithms)

- No fancy algorithm needed since |T| <= 100 and |S|*|T| is acceptable.
- Construct the new string a character at time
- Check whether T appears at the end, removing T if it does.
- Need to make sure that removing takes constant time.
- In Java, can use char[] directly, or use StringBuider with delete at the endo.

Faster Logic for Censoring

```
String s = nextStr();    String t = nextStr();
char[] ss = s.toCharArray(), tt = t.toCharArray();
int n = ss.length, m=tt.length-1, cc = 0;
char[] ans = new char[n];
for (int i=0; i<n; i++) {
  ans[cc] = ss[i];
  int k = cc, g = m;        // checking for t backwards
  while (k>=0 && g>=0 && ans[k] == tt[g]) { k--;  g--; }
  if (g == -1)    cc = k;  // t appears at the end,
  cc++;
}
out.println(new String(ans, 0, cc));
```

Censoring using StringBuilder

```
String s = nextStr(), t = nextStr();
StringBuilder ans = new StringBuilder();
int ind = 0, m = s.length(), n = t.length();
for (int i=0; i<m; i++) {
  ans.append(s.charAt(i));
  if (ans.length()  >= ind+n) {
    if (ans.indexOf(t, ind) >= 0) {
      ans.delete(ind, ind + n);
      ind -= n-1;
    } else {  ind++;    }
  }
}
out.println(ans);
```