Strong Password Checker

Explanations

In order to do the checks necessary to solve this problem, I used regex : one to check if there is at least one uppercase letter (patternLowerCase) , one to check if there is at least one lowercase letter (patternUpperCase) , one to check if there is at least one digit (patternDigits) , and one to capture groups of repetitions of the same character more than three times (patternRepeat).

const std::regex patternRepeat("(.)\\1{2,}");

const std::regex patternLowercase ("(.\*[a-z].\*)");

const std::regex patternUppercase("(.\*[A-Z].\*)");

const std::regex patternDigits("(.\*[0-9].\*)");

Next, I needed to count the number of changes that must be done in order to have at least one uppercase letter, at least one uppercase letter and at least one digit. This number of changes is computed in the “changes” variable. In order to check, I used the regex\_match function, trying to match the password with the corresponding regex pattern.

if (!regex\_match(s, patternLowercase))

changes++;

if (!regex\_match(s, patternUppercase))

changes++;

if (!regex\_match(s, patternDigits))

changes++;

After that, with the function regex\_search and the regex patternRepeat, I extract all the substrings of three or more identical characters. From these substrings, I get the number of substrings of exactly three identical characters (in variable threeCharGroups, by dividing the length by 3, so I don’t consider “aaaa” as two, only one), and the number of substrings of length divisible by 3.

while (regex\_search(stringCopy, m, patternRepeat)){

if(m[0].length()%3 == 0)

idCharGroups++;

threeCharGroups += m[0].length()/3;

stringCopy = m.suffix().str( );

}

If the size of the password is less than 20, the minimum number of changes would be

the maximum between the number of characters that need to be added for the string to be of a size of at least 16, the number of characters that need to be inserted or replaced to have at leas one lowercase, one uppercase and one digit character, and the number of changes that need to be done in order to not have three repeating characters one after another.

That is because the changes for one case can also solve the other ones.

If the length of the string is less than 6, we need to insert character till we reach at least 6. But if we don’t have at least one uppercase, one lowercase or one digit, we could also do some insertion. For example, if the password has three characters, all lowercase, we would need to add three characters to reach the minimum length, and also add one uppercase and one lowercase (so to changes) , but this can be done from the insertions to reach the length, so the answer would be the maximum between 2 and 3, meaning 3.

The same is done if we have three repetitive characters one after another. We can insert between them/ replace one of them to one uppercase, one lowercase, one digit, or we can insert between them different characters to get the minimum desired length. This may not solve all the cases of three identical characters groups – in this case we do another changes. So that’s why in the end the minimum changes will be the maximum between those three values.

if (s.size() <= 20) {

return max3(6 - s.size(), changes, threeCharGroups);

}

If the size is greater than 20, the minimum number of changes is computed in a very

Similar way as before – the changes can overlap. We substract from the number of groups of identical characters with length greater than 3 the number of characters of length exactly 3. Then we make the maximum between the number obtained and the number of changes in order to obtain one lowercase, one uppercase and one digit (in variable minChange). The deletions needed to obtain length of at most 20 might not solve all the other cases in order to obtain a strong password, so we need to add to the number of characters that must be deleted the number of other changes computed before.

else {

int repetitions = threeCharGroups - idCharGroups;

int minChange = max(changes, repetitions);

return minChange + (s.size() - 20);

}