INTRODUCTION TO FOCUS AREAS IN BIOINFORMATICS - WS19/20

Project 14: Suffix Array applications

Raghavendra Tikare and Stanislav Klein

Full list of author information is available at the end of the article

Abstract

Goal of the project: This week's project was centered on using suffix arrays to solve problems from different programming competitions.

Main result(s) of the project: There are many different applications for suffix arrays, solving problems more efficiently than other approaches.

Personal key learnings: One needs to fully understand the functionality of suffix arrays to efficiently implement and apply them on a specific situation..

Estimation of the time: 16 hours

Project evaluation on a scale of 1-5: 3

Word count: 601

Keywords: suffix array; programming competition; C++

Problem 1 - Hidden Password

Description of the problem

Given a string of length n which is involved in the string rotations. The goal is to find the lexicographic rotations using the suffix arrays. Here in order to deal with any kind of problems related to the string rotations the best way to deal with is to concatenate the string with itself in order the reduce the burden of the problem and store in a temporary string. creating the array of strings and storing all its rotations and finally sorting the array to obtain the output i.e. determining the minimum lexicographic rotation. Suffix array is just all the suffixes of the string in sorted order.

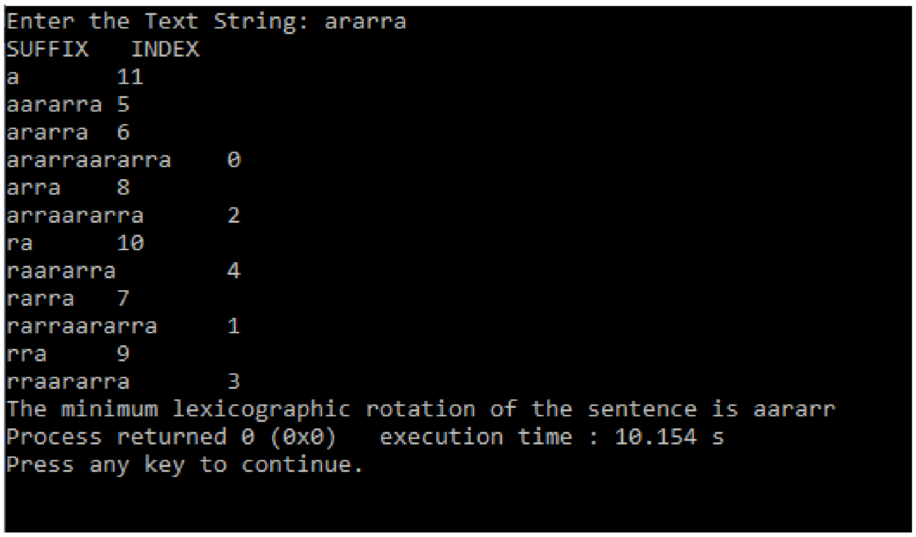
Demonstration using simple text string

Here in this example we first entered the text string "ararra" and executed it, then it initially takes the string and allocate the suffix index for all the letters in the string for the string rotations and sorts with the index for each rotation. The formed list of rotations will be in the lexicographic order and thereby gives the minimum possible lexicographic rotation which is shown in the figure 1.

Problem 7 - The longest palindrome (USACO training gate)

Description of the problem

We were given a string with a length of up to 20000 characters. The goal is to determine the longest substring in n that is also a palindrome. As opposed to the simple solution provided in the document, we surprisingly had a hard time implementing the code, or rather snippets, to mirror the procedure. This required adjusting the code in such a way that everything runs smoothly without altering



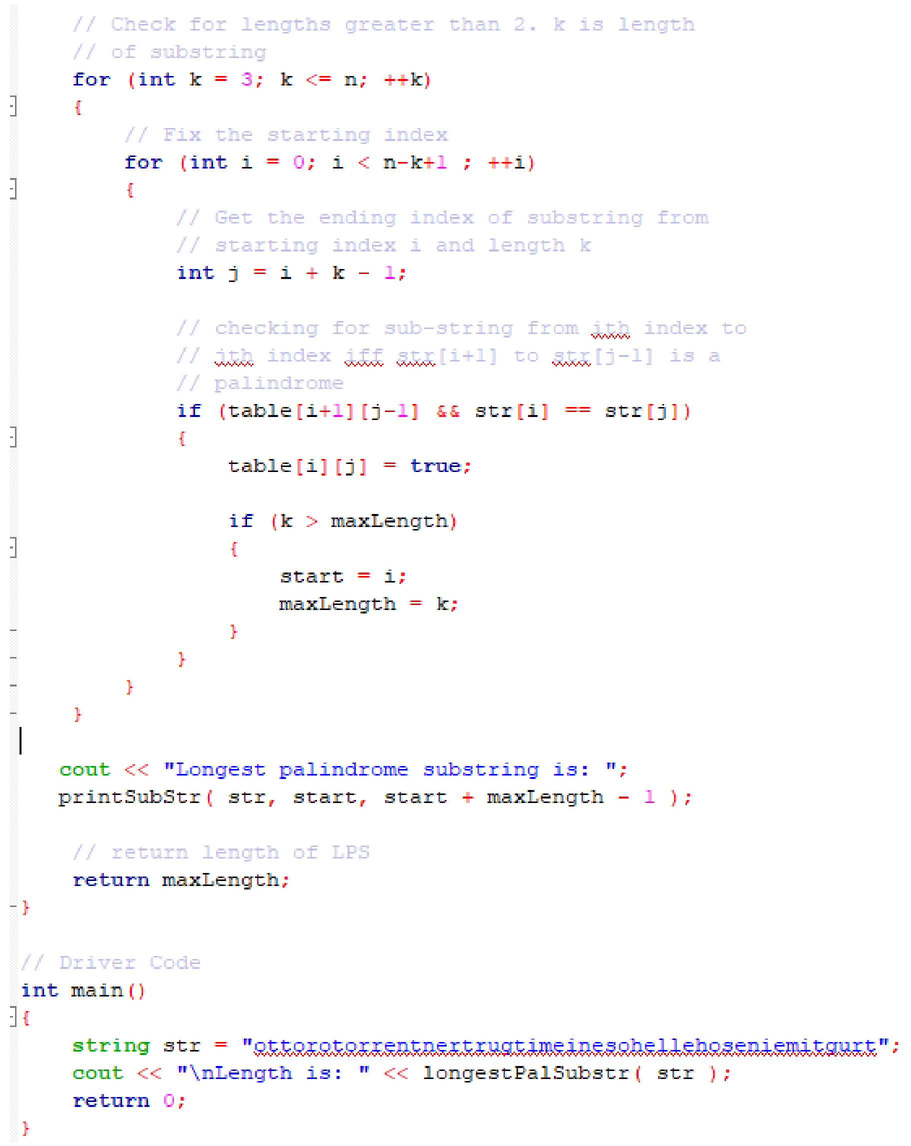
```
Enter the Text String: ararra
SUFFIX   INDEX
a       11
aararra 5
ararra  6
ararraararra    0
arra    8
arraararra      2
ra      10
raararra        4
rarra   7
rarraararra     1
rra     9
rraararra       3
The minimum lexicographic rotation of the sentence is aararr
Process returned 0 (0x0)   execution time : 10.154 s
Press any key to continue.
```

Figure 1 Hidden password with minimum lexicographic rotations

the actual code too much, which turned out harder than expected. As a result we decided to search for another possibility to solve the problem using suffixes and reversed strings in some kind. We came up with Suffix Tree Construction with provided code by a user named Ukkonen on https://www.geeksforgeeks.org/suffix-tree-application-6-longest-palindromic-substring/. This code does exactly what is required, except with a tree structure. However, assuming the provided code works fine, CodeBlocks does not fully recognize all of the syntax, so to not lose to much time on a more complex problem, we resorted to a solution without suffix arrays. On https://www.geeksforgeeks.org/longest-palindrome-substring-set-1/ we can find a way to solve the problem by using a boolean table, which we then took to demonstrate its functionality.

Demonstration using a arbitrarily chosen string

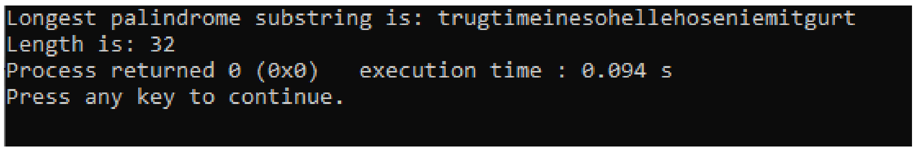
As seen in figure 2 we chose the string ”ottorotorrentnertrugtimeinesohellehose-niemitgurt”consisting of a concatenation of multiple different palindromes to test the algorithm which also yields acceptable results. By comparing the respective table entries we can determine matching characters with the help of indexes. The code consists of the parts, checking for sub strings of length 1 since every matching letter is a palindrome, length 2 or lengths of more than 2.



```
    // Check for lengths greater than 2. k is length
    // of substring
    for (int k = 3; k <= n; ++k)
    {
        // Fix the starting index
        for (int i = 0; i < n-k+1 ; ++i)
        {
            // Get the ending index of substring from
            // starting index i and length k
            int j = i + k - 1;

            // checking for sub-string from ith index to
            // jth index iff str[i+1] to str[j-1] is a
            // palindrome
            if (table[i+1][j-1] && str[i] == str[j])
            {
                table[i][j] = true;

                if (k > maxLength)
                {
                    start = i;
                    maxLength = k;
                }
            }
        }
    }

   cout << "Longest palindrome substring is: ";
   printSubStr( str, start, start + maxLength - 1 );

    // return length of LPS
    return maxLength;
}

// Driver Code
int main()
{
    string str = "ottorotorrentnertrugtimeinesohellehoseniemitgurt";
    cout << "\nLength is: " << longestPalSubstr( str );
    return 0;
}
```

Figure 2 Input of palindrome containing string and checking lenghts greater than 2

As for the output seen in figure 3, out of all the possible available concatenated palindromes, the algorithm yields the correct answer, being the substring ”trugtimeinesohellehoseniemitgurt” with a length of 32 out of 48 characters. This algorithm has quadratic runtime as opposed to the mentioned suffix tree approach, but it is easy to implement and still quite fast for small lengths.



```
Longest palindrome substring is: trugtimeinesohellehoseniemitgurt
Length is: 32
Process returned 0 (0x0)   execution time : 0.094 s
Press any key to continue.
```

Figure 3 Output of the code