Final Project Report

June 9, 2017

ECS122B Final Project

Team Members

Daniel Ruiz: Primary developer
Sebastian Grobelny: Consultant / Support developer
Melody Hsia: Support developer
Syed Akbar Mahmood: Support developer
Lac Phan: Support developer

Part1

Initial Repository

Part2

Part 2 is split in two folders, 1 for each application.
Circular String Linearization
Suffix Arrays

Team contributions

Collectively as a team we contributed in discussion to brainstorm the appropriate plan for solving the given problems as well as addressing bugs and roadblocks when encountered.

Daniel Ruiz

As the primary developer on this project Daniel performed the bulk of the implementations of the project.

Sebastian Grobelny

As a consultant Sebastian provided insight into bugs/problems faced when developing the programs.

Melody Hsia

As a support developer Melody assisted in the development of the Circular String Linearization and correcting discrepancies in the given implementation of Ukkonen's algorithm. Also, assisted in the initial setup and rearrangement of code.

Syed Akbar Mahmood

As a support developer Syed created the CMakelist and assisted in the writing of unit tests and sanity checks. Also, assisted in the initial setup and rearrangement of code and the creation of this document.

Lac Phan

Part 1

Modifications to Third Party Code

For Part 1 we needed to modify the given suffixTree code so that we could run and build a suffix tree for each test string generated. The following variables were taken out of the global run environment and declared only in the run(string text,string pattern) function.

```
std::unordered_map <long, Edge> edgeHash;
int inputLength;
string Input;
Node::noOfNodes = 1;
Node* nodeArray;
Input = text;
```

The Sanity Check generates 10,000 random texts of length 10 and patterns which are a randomly generated subset of the text. The third party search function should always return true. A screen shot of the last 10 iterations in cloud 9 are in the results section. The two unit tests check edge cases where search should return false.

Part 1 is can be executed using suffixTreeSanityCheck. The main function in suffixTree.cpp is changed to runPart1(string text,string pattern). These parameters are randomly generated with a custom function and passed in when performing the sanity check. The print edge implementation has been removed for simplicity sake. This part checks whether the tree construction and search function work with the sanity check and the following unit tests:

- **Unit test 1:** patternLargerThanText - makes sure search function returns false even if the first characters of the pattern match the text
- **Unit test 2:** patternUniqueCharacters - give pattern unique symbols (e.g. , ..) so search should return false. NOTE: third party search function has bugs but this test usually passes

Sanity test runs 10,000 cases to check Ukkonen's algorithm results (comparison between a randomly generated generated text T and pattern P).

?Two unit tests has also been established to present situations where if pattern P is larger than text T then return false and if there is a pattern P that does not contain characters of the alphabet then also return false. Both sanity and unit tests passed.?

Part 1 Results

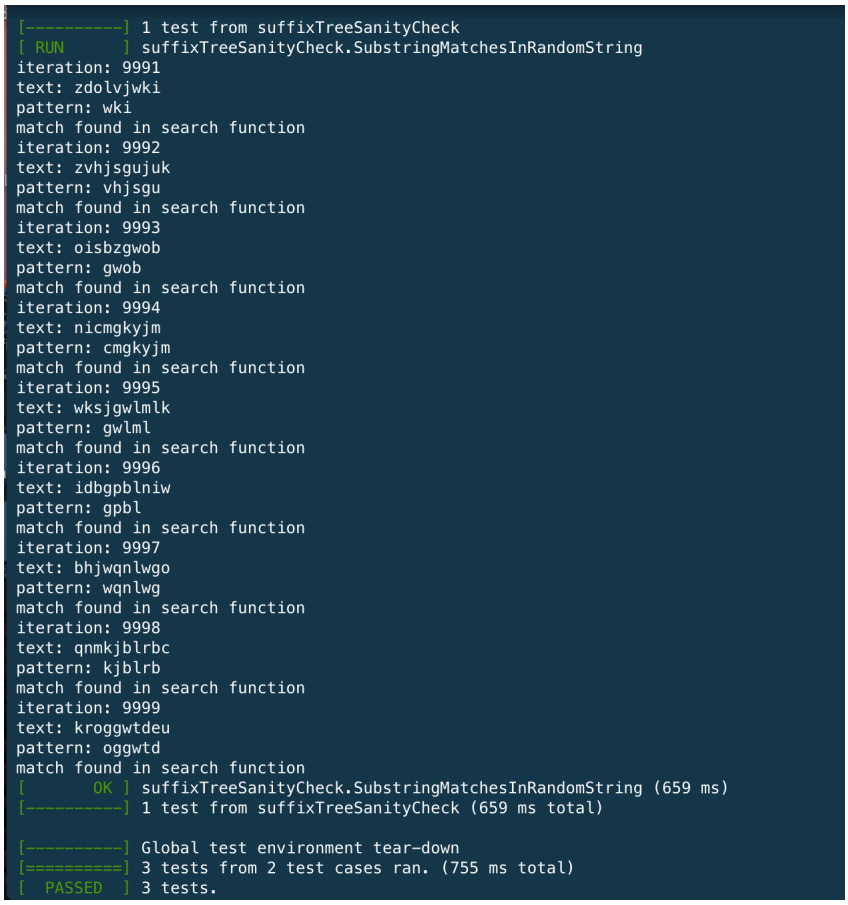


Figure 1: **Sanity Check** of pattern as a random substring of text so always results in a match between the pattern and text

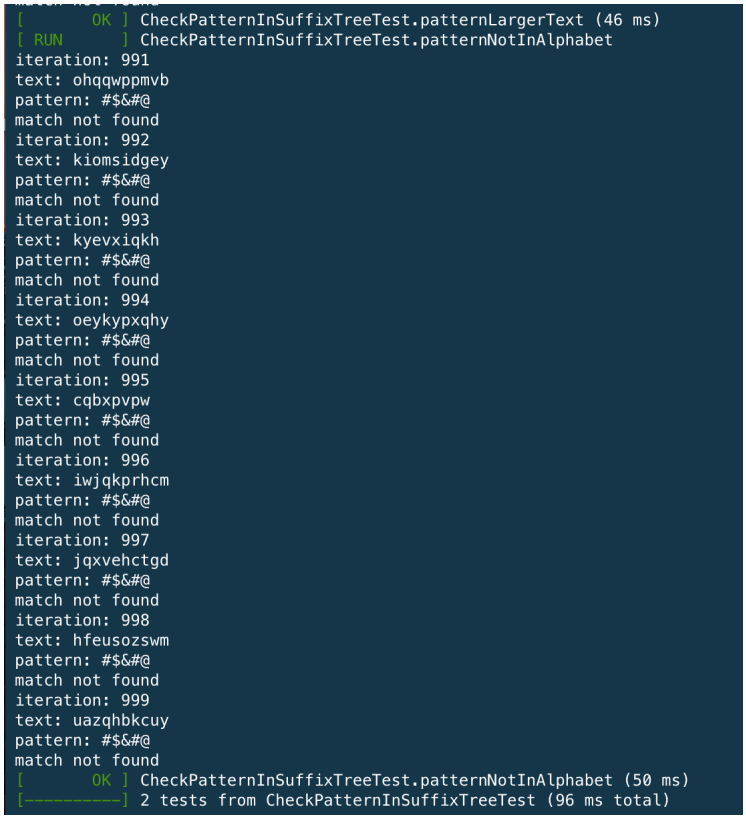


Figure 2: **Unit Test 1** where pattern is not in the alphabet

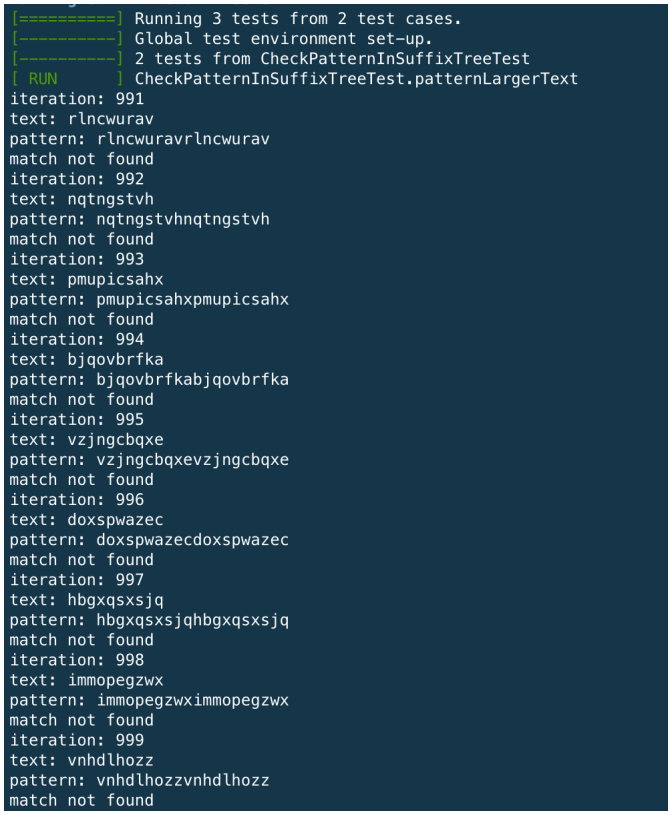


Figure 3: **Unit Test 2** where pattern is bigger than the given text

Part 2

Modifications to Third Party Code

Similar to Part 1, we modified the global variables to be declared locally in each of our separate run functions. There are two different run functions that compute our respective application implementations. We utilized the skeleton code from the printAllEdges function as a base for our own DFS functions.

The following functions were implemented:

- baseLexDFS
- recursiveLexDFS
- baseDFS
- recursiveDFS
- runArray(string text)

In addition, we implemented several recursive DFS functions such as BaseDFS and recursiveLexDFS that traverse through the suffix tree in a lexicographical order. Additionally, we added three separate run functions, runPart1, runLex,and runArray to assist in the sanity check of the application and runs each of it's respective implementations.

```
void recursiveLexDFS(suffixTree &tree, string Input, unordered_map <long, Edge>::iterator currNode,
    string *smallestString,unordered_map <long, Edge> &hash)
std::string baseLexDFS(suffixTree &tree,string Input, unordered_map <long,Edge> &hash)
void recursiveDFS(suffixTree &tree, string Input, unordered_map <long, Edge>::iterator currNode,
std::vector<int> *suffixArray,unordered_map <long, Edge> &hash, int numChars)
std::vector<int> baseDFS(suffixTree &tree,string Input, unordered_map <long, Edge> &hash)
bool runPart1(string text,string pattern)
std::string runLex(string text)
std::vector<int> runArray(string text)
```

The following unit tests for Suffix arrays were implemented.

- **Unit test 1:** lexicographicallySmallest- iterates through a text using a suffix array and returns the lexicographically smallest instance of that text.
- **Unit test 2:** generateRandomPermutatedString- generates 10,000 strings of varying lengths up to 10 and feeds each string into our runArray function.

We also have a unit test for lexDFS that uses the function on a string to check if the functions outputs the suffix tree correctly.

The following unit tests were implemented and run 1000 iterations with randomized strings.

For the suffixArray we ran into issue with the counter and consequently had 90% across 10000 iteration as the suffix tree generated by third party code differed than what was expected for certain edge cases.

An unforeseen edge case dipslayed in figure 7 below was with a string that had all the same characters, in this case our algorithm did not work as anticipated.

- **Unit test 1**: lexicographicallySmallest
- **Unit test 2**: generateRandomPermutatedString

Part 2 Results

```
ubuntu@syedakbarm-ecs122b-4697588:~/ECS122B/builds/122BFinalProject/122BFinalProject/Ukkonen-SuffixTree/LEXDFS$ ./lexDFSsanityCheck
[==========] Running 1 test from 1 test case.
[----------] Global test environment set-up.
[----------] 1 test from lexDFSSanityCheck
[ RUN      ] lexDFSSanityCheck.RandomStringIsLexicographicallyOrdered
iteration: 9991
smallestString: artseiclsm
checkerSmallestString: artseiclsm
iteration: 9992
smallestString: uvaythare
checkerSmallestString: uvaythare
iteration: 9993
smallestString: xsadjrgeo
checkerSmallestString: xsadjrgeo
iteration: 9994
smallestString: xxkdtlfvt
checkerSmallestString: xxkdtlfvt
iteration: 9995
smallestString: atyvyonwgp
checkerSmallestString: atyvyonwgp
iteration: 9996
smallestString: cnswvydxit
checkerSmallestString: cnswvydxit
iteration: 9997
smallestString: cecgxfgkve
checkerSmallestString: cecgxfgkve
iteration: 9998
smallestString: sazwmkraq
checkerSmallestString: sazwmkraq
iteration: 9999
smallestString: fflvzjtifs
checkerSmallestString: fflvzjtifs
[       OK ] lexDFSSanityCheck.RandomStringIsLexicographicallyOrdered (1037 ms)
[----------] 1 test from lexDFSSanityCheck (1037 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test case ran. (1037 ms total)
[  PASSED  ] 1 test.
```

Figure 4: **lexDFSSanity Check**

```
Running lexDFSUnitTest
[==========] Running 2 tests from 2 test cases.
[----------] Global test environment set-up.
[----------] 1 test from lexDFSTest
[ RUN      ] lexDFSTest.lexicographicallySmallest
checker: aaaa
smallest string: aaaa
[       OK ] lexDFSTest.lexicographicallySmallest (0 ms)
[----------] 1 test from lexDFSTest (0 ms total)

[----------] 1 test from lexDFSUnitTest
[ RUN      ] lexDFSUnitTest.generateRandomPermutatedString
iteration: 991
checker: acn
smallest string: acn
iteration: 992
checker: chhtil
smallest string: chhtil
iteration: 993
checker: hwol
smallest string: hwol
iteration: 994
checker: v
smallest string: v
iteration: 995
checker: fftjlylqwv
smallest string: fftjlylqwv
iteration: 996
checker: dzzvhnno
smallest string: dzzvhnno
iteration: 997
checker: essxlofg
smallest string: essxlofg
iteration: 998
checker: k
smallest string: k
iteration: 999
checker: fogi
smallest string: fogi
[       OK ] lexDFSUnitTest.generateRandomPermutatedString (64 ms)
[----------] 1 test from lexDFSUnitTest (64 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 2 test cases ran. (65 ms total)
[  PASSED  ] 2 tests.
[ 66%] Built target lexDFSUnitTest
[100%] Built target lexDFSsanityCheck
```

Figure 5: **suffixArray Sanity Check**

```
Running lexDFSUnitTest
[==========] Running 2 tests from 2 test cases.
[----------] Global test environment set-up.
[----------] 1 test from lexDFSTest
[ RUN      ] lexDFSTest.lexicographicallySmallest
checker: aaaa
smallest string: aaaa
[       OK ] lexDFSTest.lexicographicallySmallest (0 ms)
[----------] 1 test from lexDFSTest (0 ms total)

[----------] 1 test from lexDFSUnitTest
[ RUN      ] lexDFSUnitTest.generateRandomPermutatedString
iteration: 991
checker: acn
smallest string: acn
iteration: 992
checker: chhtil
smallest string: chhtil
iteration: 993
checker: hwol
smallest string: hwol
iteration: 994
checker: v
smallest string: v
iteration: 995
checker: fftjlylqwv
smallest string: fftjlylqwv
iteration: 996
checker: dzzvhnno
smallest string: dzzvhnno
iteration: 997
checker: essxlofg
smallest string: essxlofg
iteration: 998
checker: k
smallest string: k
iteration: 999
checker: fogi
smallest string: fogi
[       OK ] lexDFSUnitTest.generateRandomPermutatedString (64 ms)
[----------] 1 test from lexDFSUnitTest (64 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 2 test cases ran. (65 ms total)
[  PASSED  ] 2 tests.
[ 66%] Built target lexDFSUnitTest
[100%] Built target lexDFSsanityCheck
```

Figure 6: **lexDFS Unit Test**

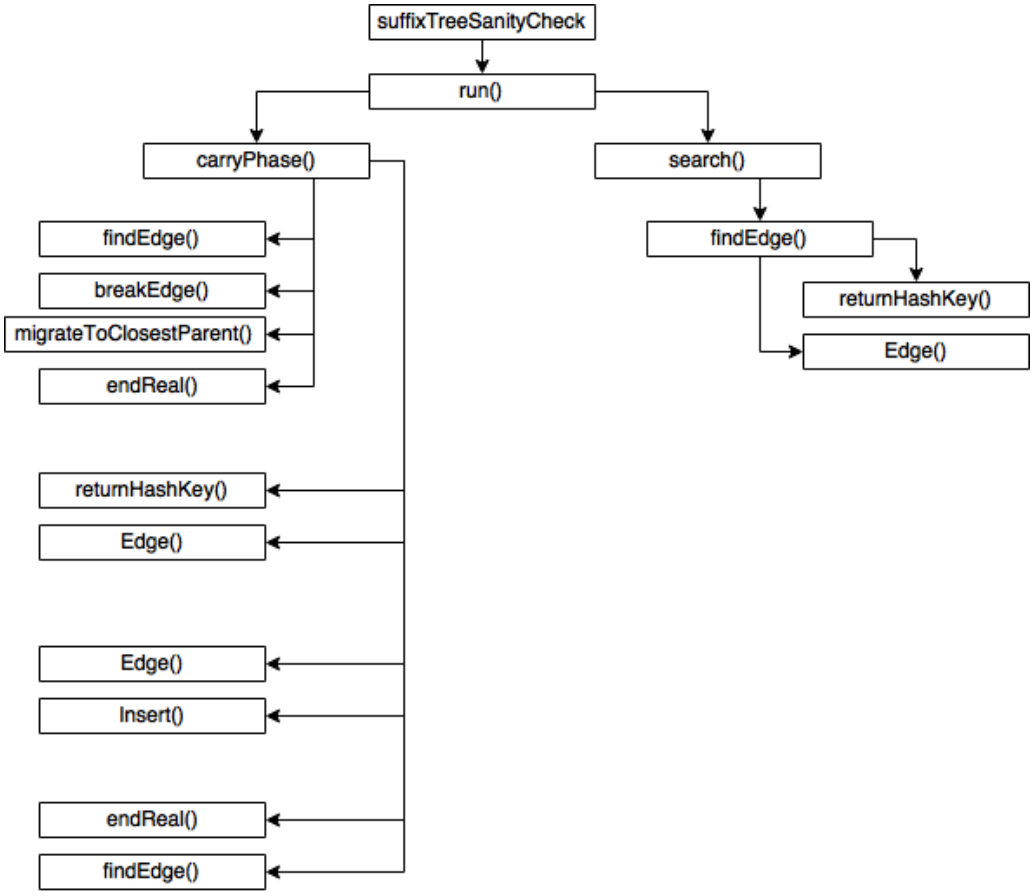
```
/home/ubuntu/ECS122B/builds/122BFinalProject/122BFinalProject/Ukkonen-SuffixTree/SUFFIXARRAY/suffixArrayUnitTest.cpp:32:
 Failure
      Expected: array[i]
      Which is: 1
To be equal to: returnSuffixArray[i]
      Which is: 10999
0
[  FAILED  ] suffixArrayUnitTest.lexicographicallySmallest (1 ms)
[ RUN      ] suffixArrayUnitTest.generateRandomPermutatedString
[       OK ] suffixArrayUnitTest.generateRandomPermutatedString (0 ms)
[----------] 2 tests from suffixArrayUnitTest (1 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 1 test case ran. (1 ms total)
[  PASSED  ] 1 test.
[  FAILED  ] 1 test, listed below:
[  FAILED  ] suffixArrayUnitTest.lexicographicallySmallest
```

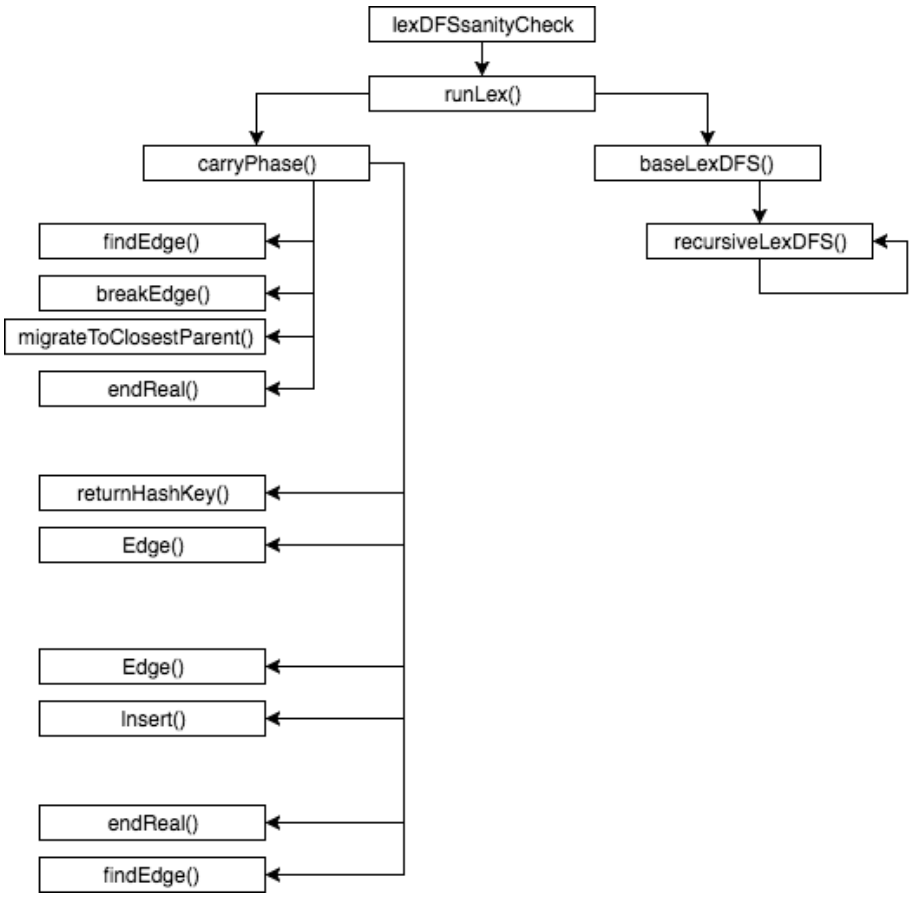
Figure 7: **Edge Case: suffixArray Unit Test**

Function hierarchy

suffixTreeSanityCheck



lexDFS


lexDFSsanityCheck
runLex()
carryPhase()
baseLexDFS()
findEdge()
breakEdge()
migrateToClosestParent()
endReal()
recursiveLexDFS()
returnHashKey()
Edge()
Edge()
Insert()
endReal()
findEdge()

Suffix Array