**CS31 Project4 Report**

**Xuanhe Zhang (706078980)**

**Question 1: A brief description of notable obstacles you overcame.**

1. It was challenging to implement the *rotateLeft()*, *flip()* and *split()* methods without creating any additional array. Especially *split()*, I use a sorting algorithm that first sorts the array. Then it iterates through the sorted array and returns the first element’s index that is equal or more than the *splitter*.

2. When I designed the *subsequence()* method, at first I used a double pointer to determine whether the elements in *a1* and *a2* were equal, and recorded the length of consecutive parts. However, I found that this method returns an incorrect result when there are duplicate elements in the array. I realized that such an algorithm was obscure.

Therefore, I used two variables *left* and *right* to trace the beginning and end of the equal part in *a1* with *a2*, and one variable *indexOfA2* to trace the current index in *a2*.

3. It was time-consuming to think of test cases corresponding to the various methods. Since there were many test cases needed, I used some auxiliary tester methods for testing.

4. For the case of n==0, the output is various for different methods and needs to be handled carefully. Especially in *subsequence()* method. When *n1* or *n2* equals 0, they return different results.

**Question 2: A list of the test data that could be used to thoroughly test your functions, along with the reason for each test.**

void testAppendToAll() {

string a[] = { "dianne", "fiona", "ed", "xavier", "greg" };

assert(appendToAll(a, 2, "!!!") == 2 && a[0] == "dianne!!!" && a[1] == "fiona!!!" && a[2] == "ed" && a[3] == "xavier" && a[4] == "greg");

//test for a valid input

assert(appendToAll(a, 4, "!!!") == 4 && a[0] == "dianne!!!!!!" && a[2] == "ed!!!" && a[3] == "xavier!!!" && a[4] == "greg");

//test for a more positions

assert(appendToAll(a, 5, "!!!") == 5 && a[4] == "greg!!!");

//test all positions

assert(appendToAll(a, -1, "!!!") == -1);

//test for an invalid index

string b[] = { "dianne", "fiona", "ed", "xavier", "greg", ""};

assert(appendToAll(b, 6, "") == 6);

//test for adding an empty string

assert(appendToAll(b, 6, " win!") == 6 && b[0] == "dianne win!" && b[5] == " win!");

//test for another string

}

void testLookup() {

string a[6] = {"A", "b", "c", "d", "E", ""};

assert(lookup(a, 6, "E") == 4);

//test for a valid input

assert(lookup(a, 6, "e") == -1);

//test for case sensitive

assert(lookup(a, 3, "e") == -1);

//test if the n variable works

assert(lookup(a, 6, "G") == -1);

//test if there is no such string

assert(lookup(a, 6, "") == 5);

//test for empty string

assert(lookup(a, -1, "e") == -1);

//test for a invalid position

}

void testPositionOfMax() {

string a[8] = { "Xander","bob", "Bob", "cabbage", "cabbage","Alice", "coco", ""};

assert(positionOfMax(a, 8) == 6);

assert(positionOfMax(a, 6) == 3);

assert(positionOfMax(a, 5) == 3);

assert(positionOfMax(a, 3) == 1);

assert(positionOfMax(a, 1) == 0); //test for only one element

assert(positionOfMax(a, 0) == -1); //test for no interesting elements

assert(positionOfMax(a, -1) == -1); //test for negative position

}

void testRotateLeft() {

string a[5] = { "kevin", "jack", "bob", "alice", "" };

assert(rotateLeft(a, 5, 1) == 1 && a[1] == "bob" && a[2] == "alice" && a[3] == "" && a[4] == "jack");

//test one valid input

assert(rotateLeft(a, 0, 3) == -1);

//test an invalid pos

assert(rotateLeft(a, 4, 4) == -1);

//test an edge case for an invalid pos

assert(rotateLeft(a, 0, 0) == -1);

//test both two variables are 0

assert(rotateLeft(a, -4, 1) == -1);

//test invalid n

assert(rotateLeft(a, 5, -4) == -1);

//test invalid pos

assert(rotateLeft(a, 5, 6) == -1);

//test the pos exceed n

string b[6] = { "This", "is", "a", "test", "case", "!" };

assert(rotateLeft(b, 6, 5) == 5 && b[5] == "!");

//no rotate

assert(rotateLeft(b, 4, 2) == 2 && b[2] == "test" && b[3] == "a");

//test for a partially rotate

};

void testCountRuns() {

string a[6] = {"A","A","a","a","A","A"};

assert(countRuns(a, 6) == 3);

//test valid input

assert(countRuns(a, 5) == 3);

//test for another n

assert(countRuns(a, 4) == 2);

//test for another n with different runs

assert(countRuns(a, 2) == 1);

//test for another n with different runs

assert(countRuns(a, 0) == 0);

//test 0 as input

assert(countRuns(a, -3) == -1);

//test bad input

string b[10] = { "A", "", "", "b", "b", "b", "b", "", "", "K" };

assert(countRuns(b, 10) == 5);

//test for empty string

assert(countRuns(b, 5) == 3);

//test for another n

}

void testFlip() {

string a[6] = {"alice", "bob", "coco", "dot", "eye", ""};

assert(flip(a, -3) == -1);

//test for invalid n

assert(flip(a, 0) == 0);

//test for 0 as n

assert(flip(a, 3) == 3 && a[0] == "coco" && a[1] == "bob" && a[2] == "alice" && a[3] == "dot");

//test a partial flip

assert(flip(a, 6) == 6 && a[0] == "" && a[1] == "eye" && a[2] == "dot" && a[3] == "alice" && a[4] == "bob" && a[5] == "coco");

//test a total flip

}

void testDiffer() {

string a[8] = { "a","b","c","d","e","f","g","h" };

string b[6] = { "a","b","c","d","e","f" };

string c[6] = { "a","","c","d","e","f" };

string copy[8] = { "a","b","c","d","e","f","g","h" };

assert(differ(a, -2, b, 3) == -1);

//test invalid n for first array

assert(differ(a, 8, b, -3) == -1);

//test invalid n for second array

assert(differ(a, 0, b, 3) == 0);

//test for 0 as n in first array

assert(differ(a, 3, b, 0) == 0);

//test for 0 as n in second array

assert(differ(a, 8, b, 6) == 6);

//test elements are equal, but a is longer than b

assert(differ(b, 6, c, 6) == 1);

//test two different array with same length

assert(differ(a, 8, c, 6) == 1);

//test two different array with different length

assert(differ(a, 8, copy, 8) == 8);

//test two exactlly same array

}

void testSubsequence() {

string a[10] = { "a", "b", "c", "d", "e", "e", "f", "g", "h", "" };

string copy[10] = { "a", "b", "c", "d", "e", "e", "f", "g", "h", "" };

string b[3] = { "a", "b", "c" };

string c[4] = { "c", "d", "e", "e" };

string d[3] = { "g", "h", "" };

string e[1] = { "e" };

string f[1] = { "" };

string g[2] = {"", ""};

string h[2] = { "e", "e" };

string i[3] = { "e", "f", "g" };

assert(subsequence(a, 10, copy, 10) == 0);

//test if two array totally equal

assert(subsequence(a, 10, copy, 3) == 0);

//test partial of copy

assert(subsequence(a, 10, b, 3) == 0);

//test a valid input in the begining

assert(subsequence(a, 5, copy, 10) == -1);

//test the second array is longer than first one

assert(subsequence(a, 10, c, 4) == 2);

//test a valid input in the middle

assert(subsequence(a, 10, d, 3) == 7);

//test a valid input in the end

assert(subsequence(a, 10, e, 1) == 4);

//test a valid input in the middle

assert(subsequence(a, 10, f, 1) == 9);

//test an empty input

assert(subsequence(a, 10, copy, 0) == 0);

//test a sequence of 0 elements

assert(subsequence(a, 10, g, 2) == -1);

//test an array that not in the first array

assert(subsequence(a, 10, h, 2) == 4);

//test a valid input

assert(subsequence(a, 6, h, 2) == 4);

//test a valid input with partial of first array

assert(subsequence(a, 5, h, 2) == -1);

//test an edge case with partial of first array

assert(subsequence(a, 5, h, 1) == 4);

//test a valid input with partial of first array

assert(subsequence(a, 10, i, 3) == 5);

//test two consective elements that only one in the second array

string j[10] = {"John","Kevin", "Kevin","Kevin","Xander","","Bob","Bob","Bob","Bob"};

string k[5] = { "Kevin","Kevin","Xander","","Bob" };

string l[5] = { "Kevin", "Xander","", "Bob", "Bob"};

string m[3] = { "Bob", "Bob", "Bob" };

string n[4] = { "Bob","Bob","Bob","Bob" };

//tests for repeated elements

assert(subsequence(j, 10, k, 5) == 2);

assert(subsequence(j, 10, l, 4) == 3);

assert(subsequence(j, 10, m, 3) == 6);

assert(subsequence(j, 10, n, 4) == 6);

assert(subsequence(a, 3, i, 0) == 0);

assert(subsequence(a, 0, i, 0) == 0);

//return 0 if two sequence are 0

assert(subsequence(a, 0, i, 1) == -1);

//return -1 if the first sequence is 0 but the second one is not 0

}

void testLookupAny() {

string a[10] = { "a", "b", "c", "d", "e", "e", "f", "g", "h", "" };

string b[3] = {"x", "y", "z"};

string c[3] = { "A", "B", "C" };

string d[3] = {"A", "", "B"};

string e[1] = { "e" };

string f[3] = {"b", "a", "c"};

assert(lookupAny(a, 10, b, 3) == -1);

//test an invalid input

assert(lookupAny(a, 10, c, 3) == -1);

//test case sensitive

assert(lookupAny(a, 10, d, 3) == 9);

//test lookup an empty string

assert(lookupAny(a, 10, e, 1) == 4);

//test multiple target string in first array

assert(lookupAny(a, 10, f, 3) == 0);

//test if multiple target string in second array

assert(lookupAny(a, 3, f, 3) == 0);

//test partial of first array

assert(lookupAny(a, 3, f, 2) == 0);

//test partial of second array which is valid

assert(lookupAny(a, 1, f, 1) == -1);

//test partial of second array which is not valid

assert(lookupAny(a, -2, f, 1) == -1);

//test invalid n1

assert(lookupAny(a, 10, f, -2) == -1);

//test invalid n2

assert(lookupAny(a, 10, f, 0) == -1);

//test if the second array contains 0 elements

assert(lookupAny(a, 0, f, 1) == -1);

//test if the first array contains 0 elements}

void testSplit() {

string a[10] = {"","Alice","BoB","xander","OK","magic","sun","possible", "Alice", "final"};

assert(split(a, 10, "") == 0);

//test for split by an empty string

assert(split(a, 10, "Alice") == 1);

//test for multiple targets

assert(split(a, 7, "Alice") == 1);

//test for partial of the array

assert(split(a, 3, "final") == 3);

//test for a string that not in the range of n

assert(split(a, 10, "BoBA") == 4);

//test for a nonexist string

assert(split(a, -3, "check") == -1);

//test a bad input

string b[3] = { "d","A","a" };

assert(split(b, 3, "e") == 3);

//test a string bigger than all strings

assert(split(b, 3, "c") == 2);

//test a string bigger than 2 strings

assert(split(b, 3, "E") == 1);

//test a string bigger than 1 string

assert(split(b, 3, "") == 0);

//test an empty input

assert(split(b, 3, "A") == 0);

//test the first string in order of alphabet

assert(split(b, 3, "a") == 1);

//test the second string in order of alphabet

assert(split(b, 3, "d") == 2);

//test the third string in order of alphabet

assert(split(b, 0, "a") == 0);

//test the case that n equals 0

}