Report

1. One main obstacle I ran into was solving the problem of having an index out of bounds exception in linux for the subsequence method. This was occuring because I did not have something that checked if the size of the sequence had been used up, only something that checked if the size of the subsequence. I fixed this by having an index checker, and putting that index checked before the two elements at the index were compared. Another error I had was figuring out how to deal with the edge case in which the position passed in was greater than the size of the array in the rotateLeft method. To solve for this, I checked if the size of the position index was greater or equal to the size of array (n), and then based on that returned a negative 1. One more obstacle was changing the differ method to fix an index out of bounds exception. I realized that before I started to go through both arrays, I needed to have a control flow statement that determined which array was smaller. Based on that, the for loop end condition would be determined.

string zero[0] = {};

***// append to all***

string nflTeams[4] = {"niners","raiders","patriots","chiefs"};

*//general case to check if method works*

assert(appendToAll(nflTeams, 4, "!!") == 4 && nflTeams[0] == "niners!!" && nflTeams[1] == "raiders!!" && nflTeams[2] == "patriots!!" && nflTeams[3] == "chiefs!!");

*//edge cases*

assert(appendToAll(zero, 0, "!!") == 0); *//zero case*

assert(appendToAll(nflTeams, -1, "!!") == -1); *//less than zero case*

***// lookup***

string n[6] = {"hari", "ravi" , "harsh", "sahen", "hari", "hArI"};

*//general cases to check if method works*

assert(lookup(n, 6, "sahen") == 3);

assert(lookup(n, 6, "hari") == 0);

*//edge cases*

assert(lookup(zero, 0, "hari") == -1); *//zero case*

assert(lookup(n, 6, "HaRi") == -1); *//test case for case sensitive string*

assert(lookup(n, -1, "harsh") == -1); *//less than zero case*

***//position of max***

string x[4] = {"green", "yellow", "red", "blue"};

string x1[6] = {"green", "blue", "red", "yellow", "yellow", "orange"};

*//general cases to check if method works*

assert(positionOfMax(x, 4) == 1);

assert(positionOfMax(x1, 6) == 3);

*//edge cases*

assert(positionOfMax(zero, 0) == -1); *//zero case*

assert(positionOfMax(x1, -3) == -1); *//less than zero case*

***// rotate to left***

string teams[4] = {"niners","raiders","patriots","chiefs"};

*//general cases to check if method works*

assert(rotateLeft(teams, 4, 3) == 3 && teams[0] == "niners" && teams[1] == "raiders" && teams[2] == "patriots" && teams[3] == "chiefs");

assert(rotateLeft(teams, 4, 0) == 0 && teams[0] == "raiders" && teams[1] == "patriots" && teams[2] == "chiefs" && teams[3] == "niners");

*//edge case*

assert(rotateLeft(teams, 4, -1) == -1); *//less than zero case*

assert(rotateLeft(teams, -1, -3) == -1); *//less than zero case*

assert(rotateLeft(teams, 4, 6) == -1); *//position > size case*

assert(rotateLeft(zero, 0, 1) == -1); *//zero case*

***// countRuns***

string s[5] = {"ted", "ted", "bill", "bill", "bill"};

string s1[3] = {"ted", "ted" , "ted"};

string s2[6] = {"ted", "tEd", "teD", "TEd", "tED", "TED"};

*//general cases to check if method works*

assert(countRuns(s, 5) == 2);

assert(countRuns(s1, 3) == 1);

assert(countRuns(s2, 6) == 6);

*//edge case*

assert(countRuns(zero, 0) == 0); *//zero case*

assert(countRuns(s, -4) == -1); *//less than zero case*

***// flip***

string fl[4] = {"a", "b", "c", "d"};

string fli[4] = {"a", "b", "c", "d"};

*//general cases to check if method works*

assert(flip(fl, 3) == 3 && fl[0] == "c" && fl[1] == "b" && fl[2] == "a" && fl[3] == "d");

assert(flip(fli, 4) == 4 && fli[0] == "d" && fli[1] == "c" && fli[2] == "b" && fli[3] == "a");

*//edge case*

assert(flip(zero, 0) == 0); *//zero case*

assert(flip(fl, -3) == -1); *//less than zero case*

***// differ***

string nba[4] = {"lakers","celtics","warriors","rockets"};

string nTeams[2] = {"lakers", "jazz"};

string nbaTeams[7] = {"lakers","celtics","warriors","rockets", "jazz", "pelicans", "knicks"};

*//general cases to check if method works*

assert(differ(nba, 4, nTeams, 2) == 1);

assert(differ(nba, 4, nbaTeams, 7) == 4);

assert(differ(nbaTeams, 7, nba, 4) == 4);

*//edge case*

assert(differ(nba, 4, zero, 0) == 0); *//zero case*

assert(differ(zero, 0, nTeams, 4) == 0); *//zero case*

assert(differ(nba, -1, nba, 4) == -1); *//less than zero case*

assert(differ(nba, 4, nTeams, -2) == -1); *//less than zero case*

***// subsequence***

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

string arr1[7] = {"a","b", "c", "e", "b", "c", "e"};

string arr2[3] = {"b", "c", "d"};

string arr3[4] = {"d", "e", "f", "g"};

*//general cases to check if method works*

assert(subsequence(arr, 3, arr1, 3) == 0);

assert(subsequence(arr, 6, arr3, 3) == 3);

assert(subsequence(arr1, 7, arr2, 3) == -1);

*//edge case*

assert(subsequence(arr, 6, zero, 0) == 0); *//zero case*

assert(subsequence(zero, 0, arr2, 3) == -1); *//zero case for sequence*

assert(subsequence(arr, 6, arr2, 7) == -1); *//n2 > n1 case*

assert(subsequence(arr, -1, arr2, 7) == -1); *//less than zero case*

assert(subsequence(arr2, 3, arr1, -2) == -1);*//less than zero subsequence case*

assert(subsequence(arr, 6, arr3, 4) == -1); *//sequence ends before subsequence found*

***// lookup any***

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

string k[3] = {"f", "g", "c"};

*//general cases to check if method works*

assert(lookupAny(l, 4, k, 3) == 1);

assert(lookupAny(k, 3 , l, 4) == 2);

assert(lookupAny(l, 4, k, 2) == -1);

*//edge case*

assert(lookupAny(zero, 0, k, 3) == -1); *//zero case*

assert(lookupAny(zero, 0, k, 0) == 0); *// zero case*

assert(lookupAny(l, -1, k, 3) == -1); *//less than zero case*

assert(lookupAny(l, 4, k, -2) == -1); *//less than zero case*

***// divide***

string u[5] = {"q", "w", "f", "h", "p"};

*//general cases to check if method works*

assert(divide(u, 5, "a") == 0);

assert(divide(u, 5, "z") == 5);

assert(divide(u, 5, "l") == 2);

assert(divide(u, 5, "r") == 4);

*//edge case*

assert(divide(zero, 0, "ted") == 0); *//zero case*

assert(divide(u, -1, "h") == -1); *//less than zero case*