Harsh Chobisa

Prof. David Smallberg

CS 31

November 4, 2018

Project 4 Report

This programming project was pretty straightforward. The biggest issue I had was with the last library function, as I had absolutely no idea how to efficiently implement a solution. What I ended up doing works, but is not efficient, because what I do is completely sort the array, and then find where the divider would be placed, and then return that value. Other issues I had were smaller ones, where I had bugs in my code that I could not find. For example, when I was writing subsequence, I kept getting an error when I ran the assert tests. Eventually, I figured out that I has a < rather than a <=, and after making that change, my program worked.

Test cases:

This tests the appendToAll method. We have a test to make sure that the exclamation marks are being appended properly. We also check for errors in input.

assert(appendToAll(a, 2, "!!!") == 2 && a[0] == "dianne!!!" && a[1] == "fiona!!!");

assert(appendToAll(a, 4, "!!!") == 4);

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

This tests the lookup method. We check that case makes a difference when running the program. We also check that eleni and ed are in the correct positions. We also check for errors in input.

assert(lookup(a, 5, "greg") == 4);

assert(lookup(a, 5, "Greg") == -1);

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

string h[7] = { "greg", "gavin", "ed", "xavier", "", "eleni", "fiona" };

assert(lookup(h, 7, "eleni") == 5);

assert(lookup(h, 7, "ed") == 2);

assert(lookup(h, 2, "ed") == -1);

This tests the positionOfMax method. We check that we are returning the correct position of max. In most of the cases shown that position is the position of Xavier.

assert(positionOfMax(h, 7) == 3);

assert(positionOfMax(a, 4) == 3);

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

assert(positionOfMax(b, 5) == 4);

assert(positionOfMax(b, 4) == 3);

This tests the rotateLeft method. First, we check for input errors, which will return -1. The last test checks individually that each value in the array has been rotated to the correct position.

assert(rotateLeft(b, 5, -1)==-1);

assert(rotateLeft(b, 5, 5)==-1);

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

assert(rotateLeft(b, 5, 0) == 0 && b[4] == "dianne" && b[0] == "fiona" && b[1] == "ed"

&& b[2] == "xavier" && b[3] == "yellow");

This tests the countRuns method. We have a series of arrays, and in every single one, we make sure that countRuns gives the proper number of runs. We also test an edge case, that if n equals zero, we return zero.

string d[9] = {

"xavier", "betty", "john", "john", "ed", "ed", "ed", "john", "john"

};

assert(countRuns(d, 9) == 5);

string e[9] = {

"xavier", "betty", "john", "john", "ed", "ed", "ed", "john", "james"

};

assert(countRuns(e, 9) == 6);

string f[9] = {

"betty", "betty", "john", "john", "ed", "ed", "ed", "john", "john"

};

assert(countRuns(f, 9) == 4);

string g[9] = {

"a", "b", "c", "d", "ed", "f", "g", "h", "i"

};

assert(countRuns(g, 9) == 9);

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

This tests the flip method. We have an array and we make sure that every value of the array has been flipped into the proper position specified by the flip algorithm.

string folks[5] = { "betty", "john", "hello", "james", "wanda"};

assert(flip(folks, 5) == 5 && folks[0] == "wanda" && folks[1] == "james" && folks[2] == "hello"

&& folks[3] == "john" && folks[4] == "betty");

This tests the differ method. We test a few different cases, but all make sure that the function is returning the proper value for whatever input is provided.

string folks2[6] = { "betty", "john", "dianne", "", "xavier", "bill" };

string group[6] = { "betty", "john", "dianne", "", "xavier", "chill" };

assert(differ(folks2, 6, group, 5)==5);

assert(differ(folks2, 6, group, 6)==5);

assert(differ(folks2, 2, group, 1)==1);

This tests the subsequence method. We test many different cases, including when there is only one match of the first item in the second array, to when there are multiple matches, to when only half of array two matches. In all cases, we are making sure that the function behaves as specified.

string names[10] = { "eleni", "gavin", "kevin", "greg", "betty", "fiona" };

string names1[10] = { "gavin", "kevin", "greg" };

assert(subsequence(names, 6, names1, 3)==1); *// returns 1*

string names2[10] = { "eleni", "greg" };

assert(subsequence(names, 5, names2, 2)==-1);

string names3[10] = { "eleni", "gavin", "kevin", "greg", "betty", "fiona" };

assert(subsequence(names, 6, names3, 6)==0);

string e2[4] = { "ed", "xavier", "", "eleni" };

assert(subsequence(h, 7, e2, 4) == 2);

string nba[] = {"clippers", "celtics", "lakers", "twolves", "warriors", "celtics", "lakers", "rockets", "heat"};

string nba2[]= {"celtics", "lakers", "rockets"};

assert(subsequence(nba, 9, nba2, 3) == 5);

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

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

*// cout << subsequence(arr, 6, arr3, 4);*

assert(subsequence(arr, 6, arr3, 4) == -1);

This tests the lookup method. We test that gavin is the proper value returned and that we return Gavin’s position, which is one. We also test for proper input in the second test.

string namez[10] = { "eleni", "gavin", "kevin", "greg", "betty", "fiona" };

string set1[10] = { "dianne", "betty", "greg", "gavin" };

assert(lookupAny(namez, 6, set1, 4)==1);

string set2[10] = { "xavier", "ed", "fiona" };

assert(lookupAny(namez, 6, set2, 2)==-1);

These tests check that divide is working properly. We check that the value returned for eleni is three, as in this given array, there are three values that would be before eleni. We also check for the case when the divider string is part of the array, which is a tricky situation to handle, as shown by the last few tests for Fiona.

string candidate2[8] = { "dianne", "fiona", "gavin", "xavier", "ed", "betty", "james", "dianne" };

string candidate[6] = { "dianne", "fiona", "gavin", "xavier", "ed", "betty" };

*// printArray(candidate, 6);*

divide(candidate, 6, "eleni");

*// printArray(candidate, 6);*

assert(divide(candidate, 6, "eleni")==3); *// returns 3*

assert(divide(h, 7, "fiona") == 3);

assert(divide(h, 7, "a") == 1);

string ddd[7] = {"alex", "bryan", "fiona", "curt", "gavin", "george", "xavier"};

assert(divide(ddd, 7, "fiona") == 3);

assert(divide(h, 7, "fiona") == 3);

These tests are test for edge cases for all of the functions. We make sure that if there are any input errors, or if there are any arrays which are empty arrays and arrays that have length zero are all handled properly by the functions. This was the most difficult part of the assignment: making sure the functions worked properly for edge cases. In most cases, we are returning -1 or 0.

string zero[0] = {};

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

*// cout << differ(zero, 0, t, 3)<< endl;*

assert(differ(zero, 0, t, 3) == 0);

string finaltest[3] = {"rahul", "swapna", "kiran"};

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

assert(lookup(zero, 0, "james") == -1);

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

assert(countRuns(zero, -1) == -1);

assert(subsequence(finaltest, 3, zero, 0) == 0);

assert(subsequence(finaltest, 3, zero, -1) == -1);

assert(subsequence(finaltest, -1, zero, 0) == -1);

assert(subsequence(finaltest, 0, zero, 1) == -1);

assert(subsequence(zero, 0, finaltest, 0) == 0);

assert(divide(finaltest, -1, "") == -1);

assert(divide(finaltest, 0, "") == 0);

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

*// rotate to left*

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

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

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

assert(rotateLeft(squads, -1, -3) == -1);

assert(rotateLeft(squads, 4, 6) == -1);

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

*// differ*

string nwa[4] = {"lakers","celtics","warriora","rockets"};

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

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

assert(differ(nwa, 4, westernConf, 2) == 1);

assert(differ(nwa, 4, zero, 0) == 0);

assert(differ(nwa, -1, nwa, 4) == -1);

assert(differ(nwa, 4, westernConf, -2) == -1);

assert(differ(nwa, 4, bestTeams, 7) == 4);

assert(differ(bestTeams, 7, nwa, 4) == 4);

assert(differ(zero, 0, squads, 4) == 0);

*//append to all edge cases*

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

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

assert(appendToAll(zero, 0, "!!") == 0);

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

assert(positionOfMax(zero, 0) == -1);

string oneoneone[5] = { "gavin", "gavin", "gavin", "xavier", "xavier" };

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

string sevensevenseven[2] = { "GAVIN", "gavin"};

assert(positionOfMax(sevensevenseven, 2) == 1);

string somanycases[2] = { "gavin", "gavin"};

assert(positionOfMax(somanycases, 2) == 0);

string hhh[7] = { "greg", "gavin", "ed", "xavier", "", "eleni", "fiona" };

string testt[3] = { "fiona", "ed", "john" };

assert(flip(testt, 3) == 3 && testt[0] == "john" && testt[2] == "fiona");

string moreNames[6] = { "betty", "john", "", "xavier", "kevin", "dianne" };

assert(flip(moreNames, 4) == 4 && moreNames[0] == "xavier" && moreNames[2] == "john");

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

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

string swaggy[5] = {"lio", "jamie", "karim", "sammi", "jon"};

assert(flip(swaggy, 5) == 5 && swaggy[0] == "jon" && swaggy[1] == "sammi" && swaggy[2] == "karim" && swaggy[3] == "jamie" && swaggy[4] == "lio");

string kobe[10] = { "eleni", "gavin", "kevin", "greg", "betty", "fiona" };

string yote[10] = { "dianne", "betty", "greg", "gavin" };

assert(lookupAny(kobe, 6, yote, 4) == 1); *// returns 1 (a1 has "gavin" there)*

string mj[10] = { "xavier", "ed" };

assert(lookupAny(kobe, 6, mj, 2) == -1); *// returns -1 (a1 has none)*

assert(lookupAny(hhh, 7, test, 3) == 2);

assert(lookupAny(kobe, 0, yote, 0) == 0);

assert(lookupAny(kobe, -4, mj, 2) == -1);

assert(lookupAny(kobe, 8, mj, -2) == -1);