Atij Mahesh

Professor Smallberg

Computer Science 31

November 2 2022

Project 4 Report

The only obstacle I encountered is with the exact specifics of every function, and their exact return value in all situations. I had to account for when n<0 for some functions, or when n<=0 for others. The main challenge I had was accounting for every possible value of n, including 0. In terms of indexing and loops, all I had to do was assure that there was never a search in an array where the index was greater than n-1.

Test Cases:

string h[7] = { "rishi", "margaret", "gordon", "tony", "", "john", "liz" };

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

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

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

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

The first three cases test the lookup method and include a case where the lookup method cannot find the string that the user inputs. The last case tests the positionOfMax in a usual-sized array of strings.

string h[7] = { "rishi", "margaret", "gordon", "tony", "", "john", "liz" };

assert(appendToAll(h, 4, "?") == 4 && h[0] == "rishi?" && h[3] == "tony?" && h[6] == "liz");

Useful for testing appendToAll, and making sure it appends the associated string to the end of the strings in the array. Also, checks if out-of-range strings remain unaffected.

string h[7] = { "rishi", "margaret", "gordon", "tony", "", "john", "liz" };

string g[4] = { "rishi", "margaret", "liz", "theresa" };

assert(differ(h, 4, g, 4) == 2);

assert(rotateLeft(g, 4, 1) == 1 && g[1] == "liz" && g[3] == "margaret" && g[2] == “theresa”);

string e[4] = { "gordon", "tony", "", "john" };

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

assert(subsequence(h,4,e,4) == -1);

Useful for testing the differ, rotateLeft methods, and subsequence methods, and the correctness of them within 2-usual sized arrays. Also, checks for when there is no instance of a2 in a1 in the subsequence method.

string d[5] = { "margaret", "margaret", "margaret", "tony", "tony" };

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

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

assert(countRuns(d, 1) == 1);

Useful for testing countRuns, with three different cases.

string h[7] = { "rishi", "margaret", "gordon", "tony", "", "john", "liz" };

string f[3] = { "liz", "gordon", "tony" };

assert(lookupAny(h, 7, f, 3) == 2);

assert(lookupAny(h, 7, f, 0) == -1);  
assert(flip(f, 0) == 0 && f[0] == "liz");

assert(flip(f, 3) == 3 && f[0] == "tony" && f[2] == "liz");

assert(split(h, 7, "liz") == 3);

Useful for testing lookup any, flip, and split methods, and a case where there is nothing to look up.

string a[2] = {“”, “”};

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

string f[3] = { "liz", "gordon", "tony" };

assert(lookup(a, 2, "john") == -1);

assert(lookup(a, 2, "") == 0);

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

assert(lookup(a, 0, "john") == -1);

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

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

assert(positionOfMax(a, -2) == -1);

assert(differ(a, 0, f, 0) == 0);

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

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

assert(appendToAll(a, -1, "bob") == -1 && a[0] == "");

assert(rotateLeft(a, 2, 4) == -1 && a[0] == "");

assert(rotateLeft(a, -1, 2) == -1 && a[0] == "");

assert(rotateLeft(a, 2, -1) == -1 && a[0] == "");

assert(subsequence(a,2,f,3) == -1);

assert(subsequence(a,1,f,3) == -1);

assert(subsequence(a,-1,f,3) == -1);

assert(subsequence(a,1,f,-3) == -1);

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

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

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

assert(lookupAny(a, -2, f, 3) == -1);

assert(lookupAny(a, 2, f, -3) == -1);

assert(lookupAny(h, 0, f, 3) == -1);

assert(lookupAny(a, 2, f, 0) == -1);

assert(flip(a, -1) == -1 && a[0] == "");

The above tests each test different cases when n is negative or 0, or when there are other strange user inputs. Some strange cases include when the user uses rotateLeft and the position being rotated about is greater than n, or when the user uses subsequence, and the second array is longer than the first. These cases mostly exist to test strange test cases or strings that the user may input.

string b[5] = {"d", "c", "b", "e", "a"};

assert(split(b, 5, "b") == 1);

assert(split(b, -1, "b") == -1);

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

A few more test cases to test split, and when n is a strange input in split (negative or zero).