Shaan Mathur

Stahl

CS31

February 13, 2015

Project 4

Most of the functions were a simple array traversal, going through a for-loop and returning if something was noticed about the array. Thus, majority of the functions were not that difficult to implement, ranging to about 15 lines of code. One function that was slightly challenging was the unionWithNoDuplicates(…) function. It could be done an inefficient manner, but I wanted to find an optimal way to write the function. After some time thinking, I created some helper functions: addWithoutDuplicates(…) and contains(…). The former would add the contents individually, making sure that it had not been added yet. The latter would help the former by returning a boolean value telling whether or not the resulting array already contained the string that we want to add. Unfortunately, the number of operations done increases a lot as we increase the number of items in the arrays. It would follow the form:

Unfortunately I could not think of another way to go about it. Perhaps if I had more time I would be able to shorten its runtime. I knew for the purposes of this assignment, however, this would be apt.

Test data:

int locateMinimum(const string array[], int n);

Test:

When n <= 0 Ensure that it returns -1

locateMinimum({“a”, “b”, “c”}, 3);

locateMinimum({“b”, “c”, “a”}, 3);

locateMinimum({“a”, “A”, “c”, “dog”}, 4);

Test actual ability of function to find minimum

locateMinimum({“duplicate”, “duplicate”, “notDuplicate!”, “duplicate”}, 4};

Test ability to return lowest index of minimum if there

are duplicates.

int findLastOccurrence(const string array[], int n, string target);

Test:

findLastOccurrence({}, 0, “BOB”)

Should return -1 if fails

findLastOccurence({“NOM”, “NOM”, “NOM”}, 3, “NOM”);

findLastOccurrence({“meow”, “meow”, “bark”, “moooo”}, 4, “meow”);

findLastOccurrence({“boo”, “bah”, “bingbong”}, 2, “boo”);

Test actual ability of function to find last Occurrence

especially with duplicate values.

int flipAround(string array[], int n);

Test:

flipAround({“a”, “b”, “c”, “d”, “e”, “f”}, 6);

flipAround({“flip”, “me”, “around”}, 3); Test ability to flip

flipAround({}, 0);

flipAround({“1”}, 1); Should return 0

bool hasNoDuplicates(const string array[], int n);

Test:

hasNoDuplicates({“blah”}, -40); Test negative numbers, should be false.

hasNoDuplicates({“nom”, “nom”, “NOM”}, 3);

hasNoDuplicates({“willbeRead”, “willBERead”, “won’tbeRead”, “won’tbeRead”}, 2);

Test actual ability of function to detect duplicates.

void unionWithNoDuplicates(const string array1[], int n1, const string array2[], int n2, string resultingString[], int& resultingSize);

Test:

string data1[5]={ “A”, “B”, “C”, “D”, “D”};

string data2[5]={ “a”, “b”, “c”, “d”, “d”};

string result[10];

int resultSize;

unionWithNoDuplicates(data1, -5, data2, 5, result, resultSize);

unionWithNoDuplicates(data1, 0, data2, 5, result, resultSize);

unionWithNoDuplicates(data1, 5, data2, 0, result, resultSize);

Should set resultingSize to -1.

unionWithNoDuplicates(data1, 5, data2, 5, result, resultSize);

Test to see if result ends up as a union without

duplicates

unionWithNoDuplicates(data1, 3, data2, 2, result, resultSize);

Test to see if result is a partial union without duplicates.

int shiftRight(string array[], int n, int amount, string placeholderToFillEmpties);

Test:

string data1[5]={ “A”, “B”, “C”, “D”, “D”};

string data2[5]={ “a”, “b”, “c”, “d”, “d”};

shiftRight(data1, 5, 0, “foo”);

shiftRight(data1, 5, 6, “foo”); Should return -1.

shiftRight(data2, 5, 2, “NOM”);

shiftRight(data1, 5, 3, “foo”); Check actual ability to shift right.

shiftRight(data1, 3, 3, “foo”);

Check ability to shift partially the array

bool isInIncreasingOrder(const string array[], int n);

Test:

string data1[5]={ “A”, “B”, “C”, “D”, “E”};

string data2[5]={ “A”, “B”, “C”, “D”, “D”};

isInIncreasingOrder(data1, -1); Should return false

isInIncreasingOrder(data1, 5);

assert(!isInIncreasingOrder(data2, 5));

isInIncreasingOrder(data2, 4); Check ability to detect whether

array is in increasing order.