Obstacles:

* I initially had problems with the split function without making a new array. However, I realized I could just sort the array in ascending order then deal with the split later. I achieved this using the bubble sort.

Test Cases

string h[7] = { "bernie", "hillary", "donald", "jeb", "", "carly", "ben" };

string g[4] = { "bernie", "hillary", "jeb", "carly" };

string e[4] = { "donald", "jeb", "", "carly" };

string d[5] = { "hillary", "hillary", "hillary", "ben", "ben" };

string y[8] = { "bernie", "bernie", "bernie", "donald", "donald", "jeb", "carly", "carly" };

string f[3] = { "jeb", "donald", "marco" };

string z1[7] = { "bernie", "hillary", "george", "barack", "lincoln", "marco", "donald" };

string z2[7] = { "bernie", "hillary", "george", "barack", "lincoln", "marco", "donald" };

string z3[7] = { "bernie", "hillary", "george", "barack", "lincoln", "marco", "donald" };

|  |  |  |
| --- | --- | --- |
| **Purpose** | **Assertion** | **Result** |
| Lookup | | |
| Standard case | assert(lookup(h, 7, "carly") == 5); | Correct |
| Target does not exist | assert(lookup(h, 2, "donald") == -1); | Correct |
| N is 0 | assert(lookup(h, 0, "bernie") == -1); | Correct |
| N is negative | assert(lookup(h, -2, "bernie") == -1); | Correct |
| AppendToAll | | |
| Standard case | assert(appendToAll(g, 4, "?") == 4 && g[0] == "bernie?" && g[3] == "carly?"); | Correct |
| N is 0 | assert(appendToAll(g, 0, "?") == 0 && g[0] == "bernie?" && g[3] == "carly?"); | Correct |
| PositionOfMax | | |
| Standard case | assert(positionOfMax(h, 7) == 3); | Correct |
| N is 0 | assert(positionOfMax(h, 0) == -1); | Correct |
| RotateLeft | | |
| Standard case | assert(rotateLeft(g, 4, 1) == 1 && g[1] == "jeb?" && g[3] == "hillary?"); | Correct |
| Rotating partial | assert(rotateLeft(g, 3, 0) == 0 && g[0] == "jeb?" && g[3] == "hillary?" && g[2] == "bernie?"); | Correct |
| When N is 0 | assert(rotateLeft(g, 0, 0) == 0 && g[0] == "jeb?" && g[3] == "hillary?" && g[2] == "bernie?"); | Correct |
| CountRuns | | |
| Standard case | assert(countRuns(d, 5) == 2); | Correct |
| Standard case | assert(countRuns(y, 8) == 4); | Correct |
| N is 0 | assert(countRuns(y, 0) == 0); | Correct |
| Flip | | |
| Standard case | assert(flip(f, 3) == 3 && f[0] == "marco" && f[2] == "jeb"); | Correct |
| Flip partial | assert(flip(y, 6) == 6 && y[0] == "jeb" && y[2] == "donald" && y[7] == "carly"); | Correct |
| N is 0 | assert(flip(f, 0) == 0 && f[0] == "marco" && f[2] == "jeb"); | Correct |
| Differ | | |
| Standard case | assert(differ(h, 4, g, 4) == 2); | Correct |
| No difference | assert(differ(h, 4, e, 4) == 0); | Correct |
| N2 is 0 | assert(differ(h, 7, e, 0) == 0); | Correct |
| N1 and N2 is 0 | assert(differ(h, 0, e, 0) == 0); | Correct |
| Subsequence | | |
| Standard case | assert(subsequence(h, 7, e, 4) == 2); | Correct |
| Outside N range | assert(subsequence(h, 5, e, 4) == -1); | Correct |
| N2 is 0 | assert(subsequence(h, 7, e, 0) == 0); | Correct |
| N1 and N2 is 0 | assert(subsequence(h, 0, e, 0) == -1); | Correct |
| N2 > N1 | assert(subsequence(e, 4, h, 7) == -1); | Correct |
| LookUpAny | | |
| Standard case | assert(lookupAny(h, 7, f, 3) == 2); | Correct |
| N2 is 0 | assert(lookupAny(h, 7, f, 0) == -1); | Correct |
| N1 is 0 | assert(lookupAny(h, 0, f, 3) == -1); | Correct |
| N1 and N2 is 0 | assert(lookupAny(h, 0, f, 0) == -1); | Correct |
| Split | | |
| Standard case | assert(split(z1, 5, "hillary") == 3 && z1[3] == "hillary" && z1[5] == "marco"); | Correct |
| All strings smaller than split | assert(split(z2, 7, "zed") == 7 && z2[3] == "george" && z2[5] == "lincoln"); | Correct |
| All strings larger than split | assert(split(z2, 7, "aaa") == 0 && z2[3] == "george" && z2[5] == "lincoln"); | Correct |
| N is 0 | assert(split(z3, 0, "hillary") == 0 && z3[1] == "hillary" && z3[6] == "donald"); | Correct |