**Data Structures and Algorithms II**

**Fall 2020**

Student name: Jacob Khalili

Student e-mail: jacob.khalili@cooper.edu

Assignment: Program #1

Grade: 94

Comments:

The program produces the correct output for five of my six test cases.

As indicated in the presubmission emails, however, it is slow when it comes to spellchecking large files, and one of my test cases in particular takes minutes to run, when it should take a second or two. Also, for that one test case that takes very long, your program reports two extra unknown words.

This shouldn't cause the bug, but looking at the findPos code, the main loop should only check for isOccupied. If a cell is not occupied, the isDeleted value shouldn't really have any meaning (deleted cells should still be occupied).

I tracked down the problem with the words. When rehash works correctly, it still returns false! So, the word that caused the rehash does not get inserted. -5 points

After this fix, the output for the final test case is correct (but it is still slow).

I am not sure what is causing the speed issue. I think it may be in the text processing. I know that regular expressions tend to be slow. I wouldn't think it causes this much of a slowdown, but I see nothing in the hash table code that should be slow, and the dictionary load speed is fine, so I don't think there any major errors with speed in the hash table function. Especially since your insert also calls contains which calls findPos, it seems like none of the major hash table member functions are slow. It's possible I'm wrong, but assuming I am correct, I am not taking off points for the speed, since that is not your fault.

Looking through the code, I think this line serves no purpose:

string line1 = line.c\_str();

It is converting line to an old-fashioned C-string, then converting right back (and assigning it) to a string! (This would not cause the major slowdown I am seeing, however, it only happens once per line.) -1 point

I verified that the speed problem was caused by the routine "resplit". I commented that out and replaced it with my own, old-fashioned (but reasonable) string processing code, just to see what would happen:

while (getline(file, line1)) {

//string line1 = line.c\_str();

//vector<string> words = resplit(line1);

vector<string> words;

string tmp = "";

for (int i = 0; i < line1.length(); i++) {

char c = line1[i];

if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z') ||

c == '-' || c == '\'')

tmp = tmp + c;

else {

if (tmp.length() > 0) {

words.push\_back(tmp);

}

tmp = "";

}

}

if (tmp.length() > 0)

words.push\_back(tmp);

I didn't rerun all the test cases, but I reran the slow one. The speed changes from over 280 seconds to under 2 seconds, and the output is still correct!

Again, I'm not taking off for this; it is not your fault that the use of regular expressions is slow.