Administrative:

- Team Name: That's So Scrabbulous
- Members: Jose Rivas Espinoza, Maya Harris, Noah Harris
- GitHub URL: https://github.com/mjoyharris/scrabble
- Link to Video:

Extended and Refined Proposal

Problem

You're playing Words With Friends against someone you desperately want to impress: your crush, your boss, your parents, your boss's parents, your parents' boss, or others. The only issue is your knowledge of English vocabulary is lacking. Worried that your opponent would correlate your Words With Friends ability directly to your intelligence, you look for any shortcut to improve your diction. And that's where we come in! With our code, you could type in any number of letters and find the top n words with the highest point total. As long as your opponent does not find out you're technically cheating, they're sure to be impressed!

Motivation

Scrabble or Words With Friends can be hard sometimes. Spending bushels of time diligently studying over the same groups of letters can feel tedious and painstaking. Our program allows you both to save your valuable time and fill your lexical desires.

Features

The user will be asked how many letters they want to use. Then they will be asked to supply those letters. Finally, they will be asked how many of the top scoring words they want to see. Following these questions, the n highest scoring words will be shown on screen along with their accompanying score.

Description of Data

The data is taken from Collins Scrabble Dictionary (2019) which can be found here: https://drive.google.com/file/d/1oGDf1wjWp5RF_X9C7HoedhIWMh5uJs8s/view. This text file consists of all 279,496 legal scrabble words. Since a Scrabble/Words With Friends board is 15x15, the longest possible word is only 15 letters, which is why you won't see anything like 'antidisestablishmentarianism' here. Using excel, we sorted this list by descending length of the word which can be found on our GitHub:
https://github.com/mjoyharris/scrabble/blob/main/Collins%20Scrabble%20Words%20(2019).txt

Tools/Languages/APIs/Libraries Used

For this code, we are only using C++ in our respective IDEs.

Graph Algorithms Implemented

We are using Breadth First Search and Depth First Search to iterate through our graph and find the suitable words.

Additional Data Structures/Algorithms Used

Along with our graph, we use a map to store both the point values of each word and an index specific to that word.

Distribution of Responsibility and Roles

Our group did the majority of our programming while on Microsoft Teams, talking and working through our problems together. On a more broad level though, Maya worked on reading in our text file, figuring out how to match the words we needed, and served as our messenger to the TA; Jose worked on getting our adjacency list up and running with some help from Noah and did the video; and Noah wrote the majority of the final report.

Analysis

Changes Made

Initially, our group misunderstood the assignment. Our plan was to store all of the words in our dictionary into a map. After the user submits their letters, we would loop through all the words and store all the words that are eligible to be created with the user's letters. We also created a graph whose 27 vertices consisted of every letter in the English alphabet and a null character. Each vertex (except null) had a weighted, directed edge that connected to every other vertex. The weight coming from the vertex would be the point value of its source vertex. As an example, for the word 'data', it would start at the 'd' and go to 'a' and give 2 points, then 't' to 'a' for 1 point and so on. Once the final letter was reached, it would go to the null vertex marking the end of the word. This process would be repeated for as many words as could be formed given the user's letters.

Unfortunately, as we misunderstood the assignment, this does not allow for us to properly use the required graph algorithms, and our graph did not have the required amount of vertices. As a result, we altered our project to create a graph consisting of all the words in the Scrabble dictionary. This now puts us well above the number of 100,000 required vertices and allows us to properly perform our DFS and BFS graph algorithms.

When we first started working on the project, we decided on a public dataset from the online site Kaggle. The dataset claimed to have consisted of the top 300,000 or so most frequent words in the English Corpus. However, once we started working with it, we noticed something a little odd about our dictionary. Strings such as “gooblle”, “acehtml”, “aesp”, and a host of others started cropping up. As fun as these pseudo-words may be, they are not quite what we were looking for. So, we decided to change to a text file we found on a public forum that holds all possible Scrabble words. This allows for a much cleaner experience both while testing and for the final product.

While working on the project, we started by forming an adjacency matrix with row sizes of 279,496 (the number of words in our dictionary). This matrix proved to be a little too large and caused our program to crash on startup. To account for this, we adjusted to using an adjacency list in the hopes that it would better hold our large amount of data.

Analysis

Adding the edges was the most intensive part of our program. Each word X has a vector that consists of words that can be formed using the letters of word X. We utilized a BFS and DFS to find the highest scoring words and the amount of words that fit our criteria. These are both O(n) since they hit every point in a certain graph cycle, but not through the entirety of the graph since not everything vertex was connected.

Reflection

Overall Experience

Honestly, our group struggled with this one. We misunderstood the prompt, had to tear down and restart our code more times than we would have wanted to, and contemplated our futures in computer science throughout it all. Nevertheless, working on a project with a friend group will always be a good time regardless of the hardships we went through.

Challenges

Our challenges were manifold. We misunderstood the prompt and had to restart our code and reshape our ideas. We got what we thought was a solid graph but it would have likely taken over a day to actually create it. During this last week we all were busy with end of the semester matters, so scheduling times to work on this was difficult.

What We Would Change To Project/Workflow

Soon after starting to work on our project, we realized that our problem was not one to be solved with a graph in any efficient manner. If we had to start again, we would do more research on a topic that better fits the guidelines of what's expected from us. Finding time to work was difficult, but largely unavoidable because of commitments we had for other classes as the semester wound to a close.

What We All Learned

Collectively we all learned to plan ahead better after coming up with a plan or even in forming a plan in general. We thought going into this project that it'd be much simpler than it turned out to be. Additionally, we spent a lot of time writing code for an impossible solution to the problem which could've been avoided by more careful forethought or testing while we code.

References

The only reference we have is our dictionary which can be found at: https://drive.google.com/file/d/1oGDf1wjWp5RF_X9C7HoedhIWMh5uJs8s/view.