# MATH 580A Assignment 2

## 1 Introduction

Wordle is a recent popular game that involves guessing a hidden 5 letter answer word. After each guess, the game will tell you whether each letter is in the answer and in the same location (green), in the answer and in a different location (yellow), or not in the answer (gray). The game typically gives you up to 6 guesses and has sparked some interesting analysis trying to come up with the best possible strategy for the game.

There are several ways to define an optimal strategy, but we will be asking for the **fewest guesses on average**, assuming every answer word has the same likelihood of appearing. This question has in fact been solved and there is even in fact a leaderboard for Wordle solvers. This assignment is based off of the techniques described by Alex Selby here, though we will only be able to scale our solver up to generate an optimal solution for a word list of about 1000 words.

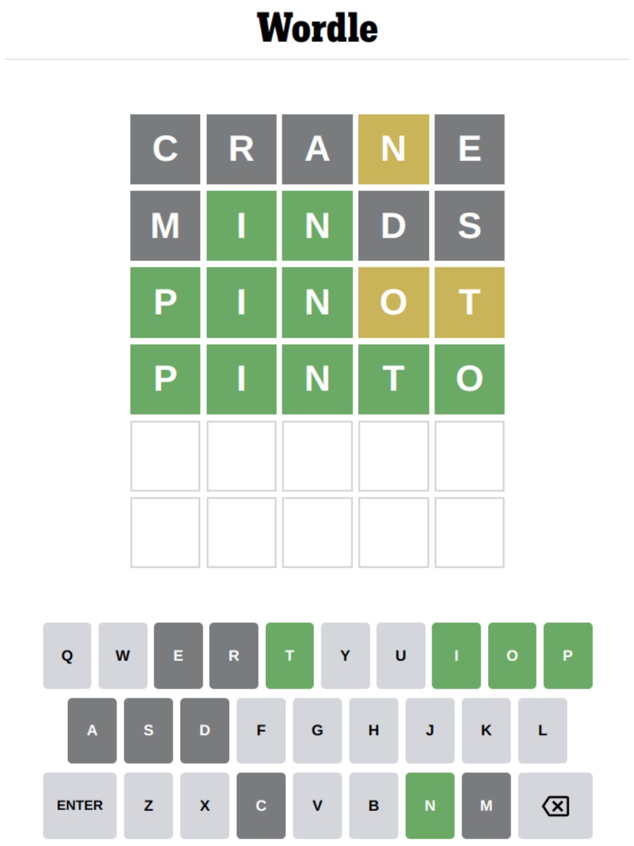

Figure 1: Sample Wordle Game

## 2 Formal Description

Let $\mathcal{A}$ denote the set of all possible answers and let $\mathcal{G}$ denote the set of allowed guesses, where necessarily $\mathcal{G} \subseteq \mathcal{A}$. Let $\mathcal{R} = \{X, Y, G\}^5$ denote the $3^5 = 243$ possible responses for a guess, where $X$ denotes a gray letter, $Y$ a yellow letter, and $G$ a green letter.

There is a function $\texttt{guess} : \mathcal{G} \times \mathcal{A} \to \mathcal{R}$ that returns a response $r$ for a given guess word and answer word pair $(g, a)$. The response $r$ is computed as follows:

1. If the $i$th letter in $g$ appears in the same position in $a$, then there is a $G$ in the $i$th position of $r$.
2. If the $i$th letter in $g$ does not appear in $a$, then there is an $X$ in the $i$th position of $r$.
3. From left to right, if the $i$th letter in $g$ appears in $a$ but not in the $i$th position *and* is not already accounted for, then there is a $Y$ in the $i$th position of $r$. Otherwise there is an $X$ in the $i$th position of $r$.

Here are two examples to clarify when $Y$'s appear in the response: $\texttt{guess(TRUST,LEAST)} = (X, X, X, G, G)$ and $\texttt{guess(OASIS,LEAST)} = (X, Y, Y, X, X)$.

A Wordle strategy $S$ is a decision tree, written as a directed rooted tree whose nodes are words in $\mathcal{G}$ and edges are labeled with a response in $\mathcal{R}$. The leaves of the tree are precisely the set of answers $\mathcal{A}$. For any path in the tree ending in a leaf $a$, the label of the outgoing edge from each intermediate node $g$ should equal $\texttt{guess}(g, a)$. For example, if the possible answers are `GORGE`, `DUSKY`, `SOOTH`, `COLON`, `WAFER`, `PAPER`, one possible strategy is the decision tree in fig. 2.

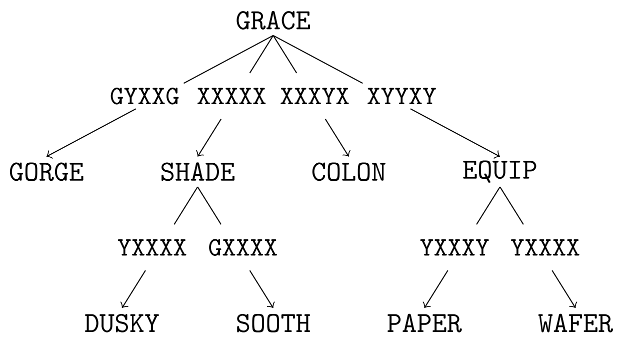

Figure 2: Example Wordle strategy

The average number of guesses needed in a strategy $S$, assuming a uniform distribution on $\mathcal{A}$ is given by

$$\frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} \ell(a),$$

where $\ell(a)$ denotes the length of the path from the root node to $a$ in $S$. Your objective will be to devise a strategy $S$ that minimizes the average number of guesses needed. Observe that for a fixed answer set, you will only need to minimize the total number of guesses $\sum \ell(a)$.

## 3 Solving Wordle Optimally

Let $T(\mathcal{A})$ denote the total number of guesses required in an optimal strategy where $\mathcal{A}$ is the set of possible answers left. Then $T(\mathcal{A})$ has a recursive structure as follows,

$$T(\mathcal{A}) = \begin{cases} 0 & \text{if } |\mathcal{A}| = 0, \\ 1 & \text{if } |\mathcal{A}| = 1, \\ |\mathcal{A}| + \min\{\sum_{r \in \mathcal{R}} T(\mathcal{A}_{g,r}) \ : \ g \in \mathcal{G}\} & \text{otherwise,} \end{cases}$$

where, $\mathcal{A}_{g,r} = \{a \in \mathcal{A} \ : \ \texttt{guess}(g, a) = r\}$. If there is only one possible answer left, then we need to guess the answer which is one guess. If there are multiple possibilities, each guess word $g$ will partition $\mathcal{A}$ into subsets $\mathcal{A}_{g,r}$ and the minimum total number of guesses required for each subset is $\sum T(\mathcal{A}_{g,r})$. In addition, the guess word $g$ needs to be included in the count, once per remaining answer left, giving a contribution of $|\mathcal{A}|$.

A recursive algorithm for finding an optimal strategy is then fairly simple to write. Starting with $\mathcal{A}$, we loop through all guesses $g \in \mathcal{G}$ and recursively compute the sum $\sum_{r \in \mathcal{R}} T(\mathcal{A}_{g,r})$. The first word in our guessing strategy should then be the word $g$ that minimizes this sum. Subsequent guessing words are computed recursively.

One implementation caveat is that we may recurse infinitely if a certain guess no longer gives any information. That is, if there is a guess word $g$ such that $\mathcal{A}_{g,r} = \mathcal{A}$ for some response $r$ and all the other subsets $\mathcal{A}_{g,r'}$ are empty. An explicit check is required to skip recursing in this case.

## 4 Solving Wordle Efficiently

The recursive algorithm we've considered so far is guaranteed to return an optimum strategy, but in practice is too slow to run to completion on the full Wordle word list when implemented naïvely. You may however wish to implement it first and test it on a smaller word list to convince yourself of its correctness first. In this section, we will cover some optimizations we can do while still guaranteeing we obtain an optimum strategy.

### Tree Pruning

The first major optimization we'll implement is tree pruning. In our recursive algorithm, each step has to compute a minimum across all guess words in $\mathcal{G}$ which is costly. The basic idea is that after we compute $\sum_{r \in \mathcal{R}} T(\mathcal{A}_{g,r})$ for some guess word $g$, we store this value and call it $\beta$. Then, for any other guess word $g'$, we obtain a lower bound on the sum without recursion. If this lower bound exceeds $\beta$, then we don't need to perform the costly recursive computation for $g'$. We will keep $\beta$ updated to be the best minimum we had computed so far. In the best-case scenario where we compute the true minimum on the very first guess word, this optimization could allow us to skip recursing on all the remaining guess words.

The crux of this optimization is in computing a good lower bound for $T(\mathcal{A}_{g,r})$ and ordering the words in $\mathcal{G}$ so that the first we try is close to the best one. Let us first tackle computing a good lower bound. Consider any subset $\mathcal{A}_{g,r}$ in the partition. If $\mathcal{A}_{g,r}$ is empty, then it contributes no guesses. If it is nonempty, then the best case scenario is that we guess one of the answers in $\mathcal{A}_{g,r}$ and if the response is not $(G, G, G, G, G)$, guess one of the remaining answers. This gives a total of $2|\mathcal{A}_{g,r}| - 1$ guesses. Thus an initial lower bound estimate we can use is

$$\texttt{estimate}(g) = \sum_{r \in \mathcal{R}} \max(2|\mathcal{A}_{g,r}| - 1, 0).$$

Recall that the true value of $g$ is $\sum_{r \in \mathcal{R}} T(\mathcal{A}_{g,r})$. As we evaluate $T(\mathcal{A}_{g,r})$ for some subset of responses in $\mathcal{R}$, we can improve our estimate, by replacing some of the lower bounds in the sum with the true value $T(\mathcal{A}_{g,r})$. It's possible that after evaluating some of the $T(\mathcal{A}_{g,r})$ recursively, we obtain an estimate that exceeds $\beta$ and we can skip the remaining recursive calls.

Try implementing tree pruning and see if you are able to compute optimal strategies for larger word lists.

## Entropy

For tree pruning to be efficient, we should ideally compute a $\beta$ value close to the true minimum on the very first guess word we try. If we knew exactly which guess word to use, then we could skip tree pruning and recursion entirely. However, we can compute an entropy value associated with each guess to obtain a quick approximation of which guess word is best.

For a fixed guess word $g$, we compute the *entropy* value associated with it via the following formula:

$$\texttt{entropy(g)} = -\sum_{r \in \mathcal{R}} \frac{|\mathcal{A}_{g,r}|}{|\mathcal{A}|} \log\left(\frac{|\mathcal{A}_{g,r}|}{|\mathcal{A}|}\right).$$

Entropy measures how much information we gain by making the guess $g$ and this is reflected in how the set of answers $\mathcal{A}$ gets partitioned into various possibilities. If $\mathcal{A}$ gets partitioned relatively uniformly into sets $\mathcal{A}_{g,r}$, then the entropy is high because each set $\mathcal{A}_{g,r}$ is now much smaller in size. In the worst case scenario, we gain no information and $\mathcal{A}_{g,r} = \mathcal{A}$ for one of the sets in the partition and all other subsets are size 0. You can verify that the entropy value is 0 in this case.

We can use entropy as a heuristic and first sort our list of guess words by the entropy they provide based on the remaining answer possibilities. This combined with tree pruning will speed up our solver significantly by reducing the number of recursive function calls we have to make, leading to fewer operations overall.

## 4.1 Memoization

In class, we've seen the technique of memoization to avoid repeating computations we've already done before. We can use this technique to speed up our Wordle solver as well. There are two primary areas where we can apply memoization. The first memoization we can do is for the `guess` function. We can memoize the response for a given $(g, a)$ guess word and answer word pair. Another function to memoize is our recursive function $T$.

It is quite possible that some sequence of guesses leads to the same set $\mathcal{A}$ of responses left as a different sequence of guesses. Because the recursive calls used to compute $T$ are so costly, we can speed up our solver significantly by memoizing our $T(\mathcal{A})$ computations for a given set $\mathcal{A}$.

## 5 Implementation Details and Requirements

Thus far, we have discussed abstractly how to solve Wordle. In this section we will discuss some of the implementation details for this assignment.

### Response Representation

A direct representation of guess responses might be to use a list of tuple of either `"X"`, `"Y"`, or `"G"`. However, in order to store and compute these guess responses more efficiently, we will represent them as numbers in base 3 instead. We will treat the letter $X$ as the digit 0, the letter $Y$ as the digit 1 and the letter $G$ as the digit 2. In addition, the base 3 representation will be read from left to right.

As an example, the guess response $(X, G, Y, X, G)$ would correspond to the base 3 number $20120_3 = 2 \cdot 3^4 + 0 \cdot 3^3 + 1 \cdot 3^2 + 2 \cdot 3^1 + 0 \cdot 3^0 = 177$. When implementing the function for computing guess responses, you will need to compute the base 3 representation as opposed to `"X"`, `"Y"`, `"G"` values.

### Solution Representation

To represent a Wordle strategy we will use nested tuples and dictionaries. Given an answer set $\mathcal{A}$, the Wordle strategy should be a tuple `(total_guesses, guess, substrategies)`. The variable `total_guesses` is the total number of guesses $T(\mathcal{A})$ summed across all answer words. The variable `guess` is the word to guess given this answer set $\mathcal{A}$. The variable `substrategies` is a dictionary whose keys are possible responses $r \in \mathcal{R}$ and whose values is a Wordle strategy for $\mathcal{A}_{g,r}$, that is another 3-tuple of the same format. If $|\mathcal{A}| = 1$, then the Wordle strategy should look like `(1, guess, None)`, where `guess` is the single answer word left in $\mathcal{A}$.

### Input, Output, and Interactivity

An efficient implementation in Python of the Wordle solver as described above will still unfortunately run several orders of magnitude too slowly to find an optimal solution for the official Wordle word lists. For this assignment, you will have several input word lists for the potential guess and answer words of varying sizes. In `wordle_runner.py` you will want to set `SIZE` to be `"small"`, `"medium"`, or `"large"`, corresponding to word lists of size 50, 200, and 500 respectively. For comparison, the official Wordle word list has 2315 possible answer words and 12972 possible guess words.

The Wordle runner can be executed in one of two modes, depending on whether `MODE` is set to `"file_output"` or `"interactive"`. In `"file_output"` mode, your solver code will be run and the final strategy will be outputted in a readable format in an output file placed under the `outputs` directory. In `"interactive"` mode, your solver code a clone of Wordle will pop up allowing you to play. At each step, your solver will be run on the remaining available answers and the optimal word to guess will be displayed along with the average number of guesses required.