# Levenshtein’s Algorithm **Introduction:** Levenshtein’s algorithm calculates the distance between two words. The distance between “kitten” and “mitten” is 1 because you can switch the ‘k’ and ‘m’ to get the other word. The distance increases with each instance required to add, delete or exchange a letter. Similarly, “Sunday” and “Saturday” have a distance of 3 to add ‘a’, ‘t’ and swap ‘n’ for ‘r’. The purpose of using Levenshtein’s algorithm with Punchy is to retrieve closely related puns when querying the database. Should the Levenshtein distance be small enough, say 1 or 2 units away, then the words could be similar lexically. If the user types “cat” we can retrieve “cat” related puns and “car” related puns.

# **Resources for Understanding Levenshtein’s Algorithm:**

* [A Beginner’s Guide to the Levenshtein Distance Algorithm (Part 1) | by Lisa Primeaux-Redmond | JavaScript in Plain English](https://javascript.plainenglish.io/a-beginners-guide-to-the-levenshtein-distance-algorithm-part-1-d581fef7588f#:~:text=The%20Levenshtein%20distance%20determines%20the%20lowest%20possible%20number,can%20become%20%E2%80%9CSundays%E2%80%9D%20in%20more%20than%20one%20way%3A) – brief introduction to better understand
* [Levenshtein distance - Wikipedia](https://en.wikipedia.org/wiki/Levenshtein_distance#Iterative_with_full_matrix) – walks through each step of the algorithm
* [Levenshtein Distance (pitt.edu)](https://people.cs.pitt.edu/~kirk/cs1501/Pruhs/Spring2006/assignments/editdistance/Levenshtein%20Distance.htm) – for implementing recursively

# **How it Works:**

# The algorithm is contained in the method GetDistance(string firstWord, string secondWord).

# Using a 2D array we can build out the words’ differences. The first word populates the first row. The second word populates the first column. I concatenate a space to the front of each string so the index of the first letter will be 1.

# The second step is to populate the rest of the array where each index represents the distance from first word letters 1 to i and second word letters 1 to j. The table below labels the letters of each index but keep in mind they are NOT part of the 2D array. It is a 5x5 array including the 0th index. For example, in the table below [2,1] will represent the distance between letters “G, U” in gumbo and “G” in gambol which would be 1.

# To populate the matrix, I use a nested for loop (this can be optimized using tail recursion) and depending if it was a deletion, addition, swap or nothing, the distance is accounted for using the minimum. See the links and code for the actual math.

Table

Description automatically generated 🡪 Text

Description automatically generated

# **Conditions to Return a “Qualifying” Pun:** In order for Levenshtein’s distance to be effective with Punchy, we must add boundaries like the maximum distance we should allow a word to be queried for in the DB. 1. The first letter must be the same. 2. The length of the word is no more than 1 or more characters shorter or longer. 3. The Levenshtein distance is at most 1 (if distance is 0, the words are the same). Examples of this include keyword “cat” returning a “car” pun or keyword “sun” returning a “stun” pun.

Message Controller  
**Introduction:**The Message Controller within PunchyAPI handles the logic to get the keywords of the user’s input and get the stemmed words as well as the words that qualify for Levenshtein distance.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Namespace** | | PunchyBot.Controllers | | | | |
| **Class name** | | MessageController | | | | |
| **Property Name** | | | **Data Type** | | **Description** | |
|  | | |  | |  | |
| **Method** | **Signature** | | | **Return** | | **Description** |
| Get | Get([FromQuery] string userInput) | | | Task<PunSetupRespDto> | | Combines keywords from stemmer and Levenshtein’s distance |
| GetStemmedUserKeyWords | GetStemmedUserKeyWords(KeyPhraseCollection phrases) | | | List<string> | | Returns both the original keywords and the stemmed versions |
| GetLevenshteinKeyWords | GetLevenshteinKeyWords(List<string> keywords) | | | Task<List<string>> | | Returns keywords from the DB that start with the same letter as the user’s keywords and meet requirements for levenshtein |