**Tech Talk III Overview**

**Grant Saylor, Kyle Smith, Anthony Tran, Jiayi Xu**

**5/6/2021**

**Topic:**

***Total Runtime: 15-20 minutes***

* **Subtopics:**

**Presenters:**

* ***Anthony:***
* ***Grant:***
* ***Kyle:***
* ***Jiayi:***

**Transcript:**

***<Grant Saylor ->***

**1 min – 5 mins:**

**SLIDE 1: LOGOLESS TITLE**

**<SCROLL WHEEL TO NEXT SLIDE>**

**SLIDE 2: TITLE DROP**

**Hi everyone, this is Grant speaking. Our team is Libraworks and we are working on the app Virtual Library. Today, myself, Kyle Smith, Anthony Tran and Jiayi Xu are going to present to you the magic of Fuzzy Search.**

**<SCROLL WHEEL TO NEXT SLIDE>**

**SLIDE 3: WHAT IS FUZZY SEARCH:**

**You might be asking yourself, “what exactly *is* fuzzy search?” You probably would not be alone in asking that question as it can be a rather nebulous topic, but there are multiple implementations and solutions that more likely than not you have encountered multiple times per day.**

**To set the stage we should talk about a hard search vs a fuzzy search. In a hard search you supply a term, say “Bellevue College”, if you were misspelling college as “collage” then no results will appear. Additionally, the search engine using a hard search will not supply any hints to your query, it really is only going to use what you’ve supplied in the search box. However, as annoying as that may be, this is where fuzzy search becomes extremely handy.**

**<SCROLL TO NEXT SLIDE>**

**Why is it called Fuzzy Search? Well, fuzzy simply means kind of vague with unclear lines, which actually makes perfect sense for this type of searching.**

**Fuzzy search takes a search query and will take in account any typos, missing terms, and formatting. I won’t go too deep into the algorithm of fuzzy searching just yet, but to give you a surface level introduction you can think of it as a graph. If I searched the word “cook”, the related graph might contain “cooking” “cooked” “cooks” and return results related to that term. However, because I searched “cook”, anything that is just your searched word, instead of variations of it ,will appear higher on the list. This would be the case of expanding a search term to near matches based on relevance.**

**Because of this near-matching functionality, I could even misspell a word and the result I wanted would likely appear. For example, Imagine that you were in a rush using Google Search to look up a business and typed the wrong word. If you used a hard search you would be further delayed, but using fuzzy search the graph will help return the result you want, sending you on your way!**

**In our app, the combination of fuzzy searching, typo checking, format checking and more provides a solution that means the end user doesn’t have to worry about perfecting their query and typing exactly what is in the database.**

**<SCROLL TO NEXT SLIDE>**

**As you can probably infer, fuzzy searching is *everywhere*. Later in the presentation Jiayi will show you some more in-depth real-world examples of where you can find it and Kyle will demo how we use fuzzy search in our own application, Virtual Library.**

**While fuzzy search can be slow, or even provide false negatives, there are ways to combat these flaws using the Levenshtein distance formula. Generally, the speed of fuzzy search can be optimized, and its powerful matching capabilities outweigh the negatives. With that I will pass the mic over to Anthony to explain the algorithm powering Fuzzy Search.**

**<SCROLL TO NEXT SLIDE>**

***<Anthony Tran ->***

**10 mins – 15 mins:**

**Hi my name is Anthony and I’m going to talk about the Levenshtein distance algorithm. In computer science, fuzzy string searching is the technique of finding strings that match a pattern approximately rather than exactly. One of the main component of this logic is the Levenshtein distance formula. This algorithm is essential the backbone of most of the logic within most fuzzy searching. The levenshtein distance between two words is the minimum number of single character edits, insertions, deletions or substitutions, which i will get into later. It is also known as the edit distance. On the right side of the screen is the formula for the levenshtein distance.**

**<SCROLL TO NEXT SLIDE>**

**Here i’ll break down the formula for you. There are 4 case for this formula. All of these cases involve 2 strings, A and B.**

**Case 1 is if the length of string B is 0, which indicates an empty string, which then will returns the length of String A.**

**case 2 is the vice versa, where if string A is empty, then it returns the length of  String B.**

**before we talk about case 3,  let’s set string a as Apple, and string b as ample. case 3 is if the first character of A and B are the same, then you take the “tail” end of each string and put it through the levenshtein formula gain. The tail of a string is basically the string without the first character. As you can see here, this is somewhat similar to recursion.**

**Any other condition will cause case 4, which will return 1 plus the minimum of these three new cases, which is also recursive. Each of the three new cases corresponds with the 3 edit operations, deletion, insertion, and substitution.  In the end, case 4 will return a number, indicating how many edit operations it used.**

**<SCROLL TO NEXT SLIDE>**

**So again the three edit operations that the levenshtein distance uses is insertion, deletion and substitution. These three operations can be generalized as forms of substitutions by adding a null character star, wherever a character has been deleted or inserted. (explains the edit examples). Theres also the transposition operation, but its not part of the levenshtein distance.**

**<SCROLL TO NEXT SLIDE>**

**Here is an example of how to apply the levenshtein distance to two strings kitten and sitting. To get from kitten to sitting, there is a distance of no less than 3 because 3 is the least amount of edit operations you must use to get from kitten to sitting. Starting from the beginning of each string, the Levenshtein distance will compare character by character, until all characters have been evaluated with each other. So in this case, k and s are different so a substitution is done here. I t and t are similar on both sides so nothing happens. Then we do another substitution with e and I. I and N are similar. Even though we finished comparing all the characters of kitten, sitting still has another character G. so we need to do an insertion at the end. Finishing it off with 3 operations.**

**So in conclusion, the more edit operations there needs to be to compare two strings, there more unlikely the two strings are similar. Programmers and coders use this knowledge to cater to their own version of fuzzy search. Now im going to hand it over to Jiayi.**

***<Jiayi Xu ->***

**10 mins – 15 mins:**

**Hi, this is Jiayi Xu. I am going to introduce fuzzy search examples in the real world. Before seeing examples, I would like to talk a little bit about the Data quality of the Fuzzy Search.**

**Two categories of Data quality are standard data and less standardized data. The standard data like Country and state names, phone number format, email and URL, Capitalization on a name. The less standardized data like company name, job title, seniority level, part number, failure, and repair code.**

**Let see some examples of fuzzy search. When we use a website to search for something, we need to type words in the text box, if the website does not use fuzzy search, we need to type the exact word to find the result, however, most companies using the fuzzy search now, like Microsoft, Google, and other companies. If we type in "Goo" in the text box, we can see the Google list below. Other than websites, you also can think about the mobile application, like our group's project. We are using fuzzy search to do the search book jobs.**

**(Next Slide)**

**Another example I want to show is the Word Document, when we have typo mistakes, the word document will show the spelling options (Azure Cognitive Search).**

**(Next Slide)**

**As you can see, I listed the SQL here, the SQL also using the fuzzy search, to find the strings that match a pattern approximately rather than exactly. The SQL using the FREETEXT() function, so users do not need to use full-text search.**

**Now, let's take a look at a demonstration of fuzzy search work in our project. I will hand it over to Kyle. Thank you.**

***<Kyle Smith ->***

**5 mins – 10 mins:**

***Hi everyone, this is Kyle, and I’m going to be showing you our demo.  In our app we decided to use a package for our implementation of fuzzy search called Fuzzy Wuzzy.  So here you’ll see an emulator instance of our app on the right, and then a part of our IDE, Android Studio, called the ‘logcat’ on the bottom, this is showing printouts that occur within our code.  I’m going to head into the Search activity within our app, and as you’ll see in the logcat, we have some printouts.  Before I actually make searches within our app, I wanted to show you how our implementation of fuzzy search, ‘fuzzy wuzzy’, is working behind the scenes.  I made some print statements that show you a comparison of 2 strings and the comparison value that fuzzy wuzzy gives them as a pair.  The score has a maximum value of 100, with that score meaning a perfect match.  The first result between “Apple” and “Apple” is 100.  We can also see the results for “Apple” and “Ample” with a result of 80, “Apple” and “sapple” with a 91, and “Apple” and “elppA” with a score of 40.  These short comparisons show that the closer one word is to the other, the higher the matching score is.  So in our example the closest match to Apple would be “sapple”.  Even though the word “elppA” is the same length and same characters as the target word, Apple, it scores low because it prioritizes order.  This is a smart choice because when people do search for something and get it wrong, it’s usually a misspelling, not putting the word in backwards.  Below the results for the 1-word comparisons, I made a comparison for a book titled, “Barefoot in Paris”.  You can see that even with 2 separate spelling mistakes, it still finds a pretty high comparison score of 85.***

***Now let’s do some searches within our own book database.  If I search “barefoot”, we see that the search brings up 2 books within our DB, which is good, both books have that term within them.  The order of these search results is based on that comparison score from fuzzy wuzzy, so if I alter my search to say, “Barefoot Paris”, now we see that the book “Barefoot in Paris” jumps to the first position, since it now has a higher score than the other book.  We also use fuzzy wuzzy to look at all our authors as well, so if I flip the search and look for the author instead of the title, and use “Garten”, now we see all the books in our DB from Ina Garten, including a third book that doesn’t include Barefoot in the title.  Let’s try a book series, where each book has the same initial title, but has a different subtitle that follows.  Let’s look at the Artemis Fowl series.  If I search ‘Artemis’, we get 4 of the books in the series, even though some of the later books have much longer titles, such as “Artemis Fowl and the Opal Deception”.***

***Another way we use fuzzy wuzzy in our app is a little hidden behind the scenes, but we can use it to filter down our search results.  If you think about it, all string comparisons will give you a non-zero comparison value (assuming neither is an empty string), so if you make any search in our app, it would generate a comparison value for all the book titles and authors in our DB, and would display them here on the screen.  A way that we can get around this is to only accept a certain score and all values above it to make sure the search results are actually relevant.  If I searched for Artemis Fowl and still got a result for Barefoot in Paris, that wouldn’t really make any sense, so we set our threshold higher up so that only comparisons that are pretty close actually get shown to the user.***