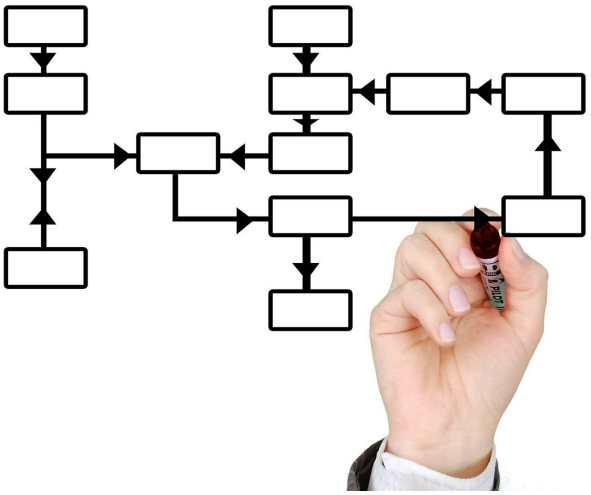
Solving Algorithms: A Structured Approach to Problem Solving

Whether you are delving into the job interview pool for a developer role or simply wanting to level up your problem solving skills. Then a question you have probably posed to yourself after the 100th codewars kata is “Is there a methodical approach I can use to conquer any coding problem?”.

The challenge pertaining to *any* problem makes it very difficult to create a specific method however there are general principles to be followed, that would help one progress to the next stage of solving the problem. Especially useful when you are completely perplexed by the task at hand (and we’ve all been there).

Step 1: Start with an example



The key here is finding the right balance. Pick a difficult example full of edge cases and you risk overwhelming yourself initially. On the other hand you can pick something far too basic and reach the stage of having code that compiles and asserts the expected outcome for a couple of scenarios but still fails on the majority of inputs. This can instill you with a false confidence but also much frustration upon trying to progress from here.

Problem: Given a string of many words, return a list of the top 3 most frequent words in descending order.

Example A: “Deer elk caribou”

Example B: “Meat salad potatoes meat bread salad meat tomatoes bread”

Example C: “I think i can but why can’t i think cant you see why i am confused”

Which example would you chose to work through the problem initially?

Let's consider each option in turn, A only considers 3 distinct words to determine a list of 3 most frequent words resulting in the simplistic edge case of a tiebreaker. Assuming tiebreakers can carry a random order, this would mean merely converting the string to a list of words would yield the correct output. No further logic needs to be implemented therefore this example should be avoided.

Option B contains 5 distinct words therefore observing the result, one should notice the absence of two words. Also there is no ambiguity over which word will come first in the list as 'meat' is the clear winner. There does exist two tie breakers between "salad" and "bread" & "potatoes" and "tomatoes". Due to the list having a length of 3, the spaces will be filled with the highest tie breaker therefore leaving "potatoes" and "tomatoes" absent. The example is relatively straight forward and can be calculated in your head within a matter of seconds yet counting logic is still needed to be able to arrive at the correct answer with a small amount of ambiguity. Best option so far…

First of all option C contains ELEVEN distinct words whilst this isn't a major hurdle to some individuals. One has to consider the interview scenario where you may be slightly nervous and panicked, the extra complexity here would be deemed unnecessary by wasting time in working through a long rudimentary counting problem. On top of this you have presented yourself with another edge case of dealing with special characters (leading to can't and cant being counted as distinct from each other). This can be examined later on allowing you to focus on getting to a general solution faster. You could even verbally note that you recognise this issue to the examiner but state that you're going to deal with it later.

Step 2: Brute force

You may have come across this term before when dealing with algorithms but not entirely understood it. Intuitively it suggests doing whatever it takes no matter what the cost to solve the problem. If you can't open a door with a key … why not bash it down with your foot?

But how does this translate into solving algos? Now the cost means inefficiency of the algorithm so brute force allows us to free ourselves from the constraints of efficiency in two ways. By not having to worry about...
i) memory.
ii) time taken

Remember this applies to challenging scenarios only, intuitively you may be able to jump straight into a partially optimised solution. If that's the case feel free to skip this step.

Consider the example from step 1

Example B: "Meat salad potatoes meat bread salad meat tomatoes bread"

1. Convert string of words to list.
2. Create an empty dictionary structure/ hashmap.
3. Iterate through list
    a. Search for list item in dictionary
    b. If exists increment value by 1 (value represents frequency)
    c. Otherwise initialize key as list word and set frequency to 1.

4. Now the dictionary is an unordered structure therefore it cannot be sorted easily. Yet we can iterate through the dictionary
    a. Compare every single key's value against the resulting list of top three most frequent words (initialised to empty list).
    b. If value is higher than current index Add the key:value pair as an object or nested array/tuple and copy the rest of the array to the next index onwards.
    c. Otherwise compare it to next index until reached final index where if lower push to the end of the array.

By the end we should have a list which can be shortened to length 3 giving us our answer. Each step logically makes sense but some gross inefficiencies can be highlighted and these will be addressed in the next step.

Step 3: Optimizing