2D Arrays: String Manipulation- Part 1

Relevel
by Unacademy

List of Concepts Involved:

- Stones and Jewels
- Valid JSON Object
- Another Number System
- String Strength

Stones and Jewels [Easy]

- Let us say you work in a jewellery company where your task is to record all the jewellery items that came out from piles of stones.
- For example, from a given pile of stones, for the sake of convenience we say 40 units, consists of 33 units of other metals and 7 units of metals which can be used to make jewellery. So, you record 7 units of jewellery from 40 units of stones.
- Here you will be given string jewels representing the types of stones that are used to make jewellery, and second string as stones representing the stones we received. Each character in the given stones string will represent a type of stone either useful for making jewellery or other type of metal which are not useful to make jewellery. We want to know how many of the stones we have are kind of stones which can be used to make jewellery.
- For our convenience we have made alphabetical symbols to represent both stones and jewels, i.e., a-z and A-Z.
- Note: "a" is considered a different type of stone from "A".

Stones and Jewels [Easy]

Example 1:
Input: jewels = "bB", stones = "bbbbaAA"
Output: 4

Example 2:
Input: jewels = "v", stones = "VV"
Output: 0

Constraints:

- 1 <= length of jewels and stones string length <= 50
- Jewels' and stones' strings consist of only English letters.
- All the characters in jewels' string are unique.

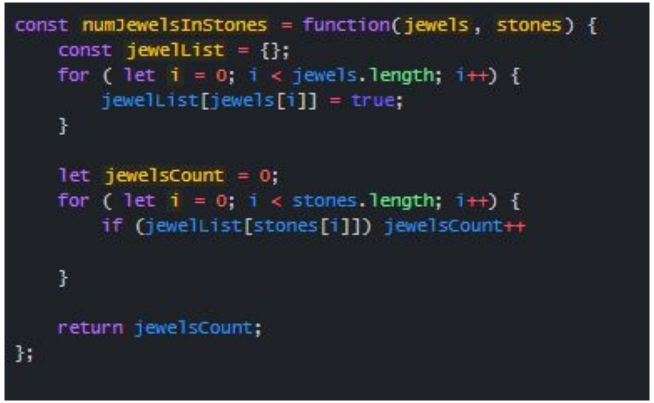
Stones and Jewels [Easy]

Approach:

1. While iterating over each stone from stones' string, we will be checking if the given character which represents a stone can be used to make jewellery or not.
2. We need to have a data structure here which can confirm us if the given stone is a jewel or not with searching time complexity of O(1) to reduce time complexity.
3. We will be using a JS object here, with keys as jewels' stone characters as they are unique with a value as true. So when we access any value with a stone character from this DS it will either return true or undefined i.e., considered as false.
4. For, valid condition we will increase the counter counting number of jewels.
5. Return the counter.

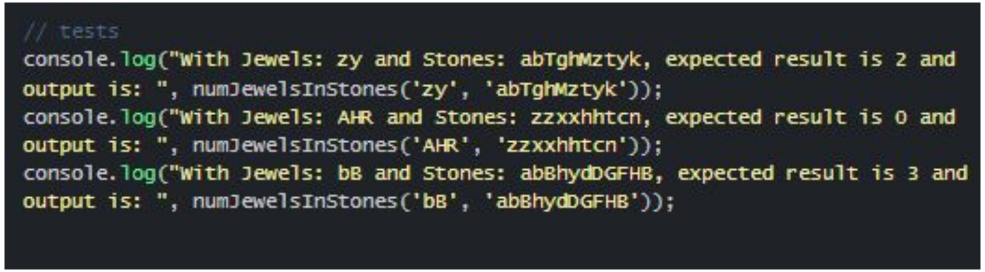
Stones and Jewels [Easy]

Code:



```
const numJewelsInStones = function(jewels, stones) {
    const jewelList = {};
    for ( let i = 0; i < jewels.length; i++) {
        jewelList[jewels[i]] = true;
    }

    let jewelsCount = 0;
    for ( let i = 0; i < stones.length; i++) {
        if (jewelList[stones[i]]) jewelsCount++

    }

    return jewelsCount;
};
```

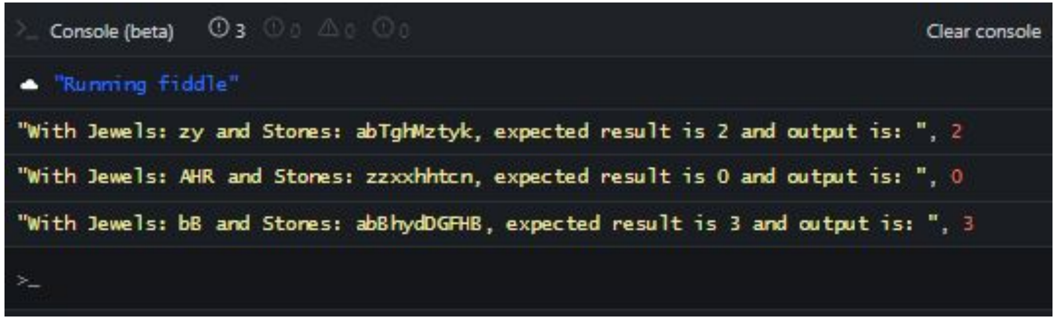
Tests:



```
// tests
console.log("With Jewels: zy and Stones: abTghMztyk, expected result is 2 and
output is: ", numJewelsInStones('zy', 'abTghMztyk'));
console.log("With Jewels: AHR and Stones: zzxxhhtcn, expected result is 0 and
output is: ", numJewelsInStones('AHR', 'zzxxhhtcn'));
console.log("With Jewels: bB and Stones: abBhydDGFHB, expected result is 3 and
output is: ", numJewelsInStones('bB', 'abBhydDGFHB'));
```

Stones and Jewels [Easy]

Output:



```
"Running fiddle"
"With Jewels: zy and Stones: abTghMztyk, expected result is 2 and output is: ", 2
"With Jewels: AHR and Stones: zzxxhhtcn, expected result is 0 and output is: ", 0
"With Jewels: bB and Stones: abBhydDGFHB, expected result is 3 and output is: ", 3
>_
```

Code link: https://jsfiddle.net/wj3aLq4t/

IdeOne link: https://ideone.com/ENdquQ

Time Complexity: O(n+ m), where n, m are lengths of jewels and stones length respectively.
Space Complexity: O(n), where n is size of jewels string.

Valid Parentheses Sequence [Easy]

- JSON object stands for JavaScript Object Notation, which is basically a JS object converted into a single string of data which is widely used for communication between REST web services and it's also easy to interpret by human as well as machine.
- JSON Objects always have valid parentheses sequence which is often necessary to read data from a json string.
- These are rules we must follow to decide a valid parentheses sequence such as:
  - Open brackets must be closed by the same type of brackets.
  - Open brackets must be closed in the correct order.
- Our goal is to identify if the given string is a valid JSON object, by verifying the above two conditions. For the sake of convenience all the text and other symbols are erased and the input string just contains the brackets like '(', ')', '{', '}', '[' and ']'. Write a function to determine if the input string is a valid JSON or not.

Valid Parentheses Sequence [Easy]

Example 1:
Input: s = "()()"
Output: true

Example 2:
Input: s = "[]{}()"
Output: true

Example 3:
Input: s = " {[}(])"
Output: false

Constraints:

- 1 <= length of input array <= 104
- Input string consists of parentheses only '()[]{}'.

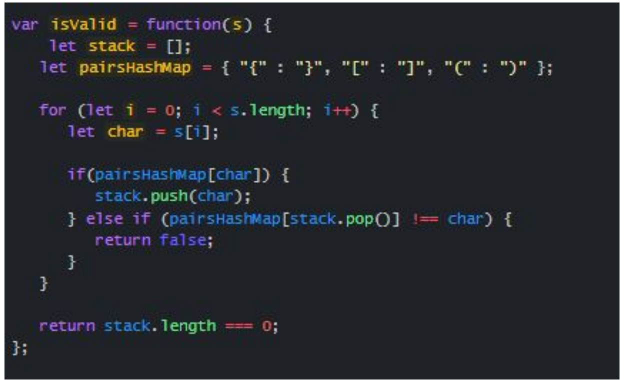
Valid Parentheses Sequence [Easy]

Approach:

1. First, we will be needing a data structure here to store brackets combinations and to reduce time complexity we will use a JS object with O(1) search complexity.
2. We will be verifying a valid string using a stack and its concept of push and pop.
3. When an open bracket is read we will push it in the stack and
4. When a close bracket is read we will pop.
5. Now to validate the orders the stack will itself help us as the last opened bracket should be closed first i.e., it should be at the top of the stack, if this does not happen we can confirm that the string is not a valid JSON object.
6. And finally if the stack is not empty we can also confirm that the string is not a valid JSON object.

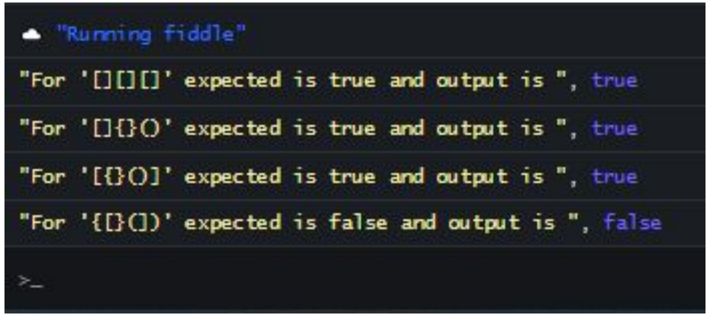
Valid Parentheses Sequence [Easy]

Code:



```
var isValid = function(s) {
    let stack = [];
   let pairsHashMap = { "{" : "}", "[" : "]", "(" : ")" };

   for (let i = 0; i < s.length; i++) {
      let char = s[i];

      if(pairsHashMap[char]) {
         stack.push(char);
      } else if (pairsHashMap[stack.pop()] !== char) {
         return false;
      }
   }

   return stack.length === 0;
};
```

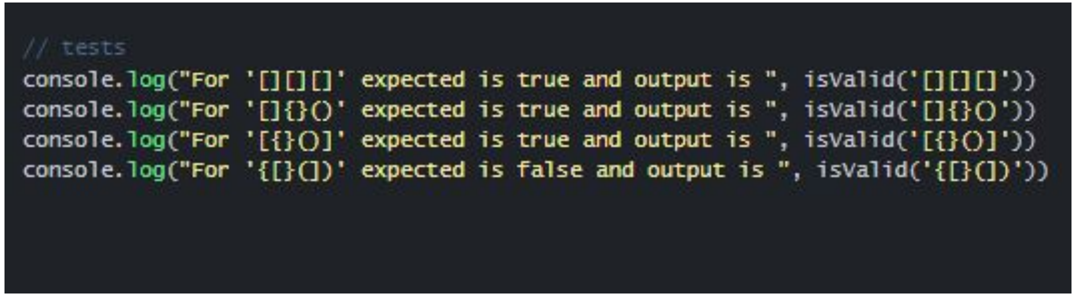
Tests:



```
"Running fiddle"
"For '[][][]' expected is true and output is ", true
"For '[]{}()' expected is true and output is ", true
"For '[{}()]' expected is true and output is ", true
"For '{[}()}' expected is false and output is ", false
>_
```

Valid Parentheses Sequence [Easy]

Output:



```
// tests
console.log("For '[][][]' expected is true and output is ", isValid('[][][]'))
console.log("For '[]{}()' expected is true and output is ", isValid('[]{}()'))
console.log("For '[{}()]' expected is true and output is ", isValid('[{}()]'))
console.log("For '{[}(])' expected is false and output is ", isValid('{[}(])'))
```

Code Link: https://jsfiddle.net/6xyjdu19/

IdeOne link: https://ideone.com/nE8bmz

Time Complexity: O(n), where n is the length of input string.
Space Complexity: Worst Case- O(n), when there are all unpaired brackets in the input string.

Another Number System [Medium]

We are introduced with a new number system where

A represents as 1

B represents as 2

C represents as 3

…

Z represents as 26

AA represents as 27

AB represents as 28

…

Let’s call this system the ALPHA numeric system. And our goal here is to convert the number given in ALPHA numeric form to the DECIMAL number system.

Another Number System [Medium]

Example 1:
Input: "AA"
Output: 27

Example 2:
Input: "ABC"
Output: 731

Example 3:
Input: "J"
Output: 10

Constraints:

- 1 <=input string length <= 7
- Input string consists only of uppercase English letters.
- Input string is in the range ["A", "FXSHRXW"].

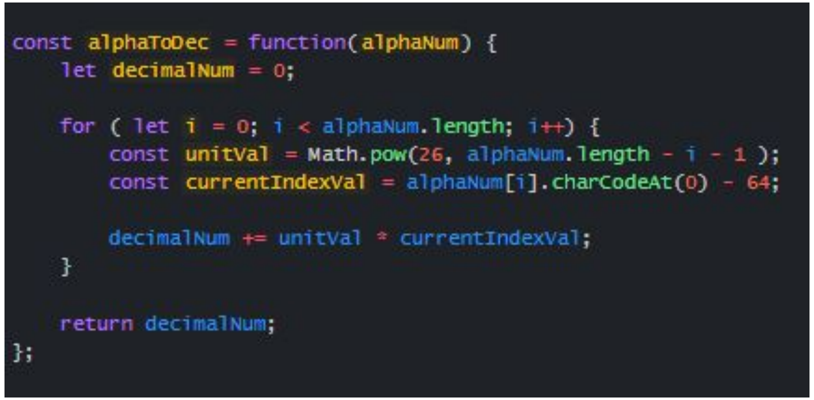
Another Number System [Medium]

Approach:

1. Like in DECIMAL number system when we see 143, we say that it is 1 hundred 40 tens and 3, i.e., 1 x 10*2 + 4 x 10*1 + 3 x 10*0 = 143.
2. We will be implementing the same logic here as well, all we have to do is replace 10 with 26, which is our new base of the number system here.
3. Calculate the unitVal: power of 26 at that position
4. And the currentIndexValue: value of the current letter, e.g. 11 for K
5. Apply the above formula
6. And return the decimal value.

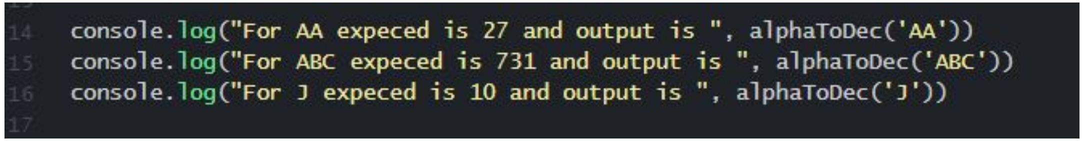
Another Number System [Medium]

Code:



```
const alphaToDec = function(alphaNum) {
    let decimalNum = 0;

    for ( let i = 0; i < alphaNum.length; i++) {
        const unitVal = Math.pow(26, alphaNum.length - i - 1 );
        const currentIndexVal = alphaNum[i].charCodeAt(0) - 64;

        decimalNum += unitVal * currentIndexVal;
    }

    return decimalNum;
};
```

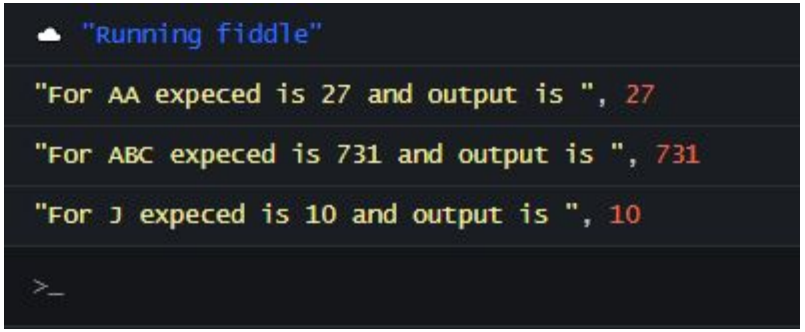
Tests:



```
console.log("For AA expeced is 27 and output is ", alphaToDec('AA'))
console.log("For ABC expeced is 731 and output is ", alphaToDec('ABC'))
console.log("For J expeced is 10 and output is ", alphaToDec('J'))
```

Another Number System [Medium]

Output:



```
"Running fiddle"
"For AA expeced is 27 and output is ", 27
"For ABC expeced is 731 and output is ", 731
"For J expeced is 10 and output is ", 10
>_
```

Code link: https://jsfiddle.net/qmr93jnf/

IdeOne link: https://ideone.com/bQ5Lvl

Time Complexity: O(n), where n is the length of input string.
Space Complexity: O(1).

String Strength [Hard]

- Let us assume there is a new logic defined to determine strength of a word, which is a longest possible substring of the word that only has unique characters in it.
- For example, the word ‘random’ has a strength of 100%, which is derived on the logic that it has all unique characters in it, so length of the longest substring with unique characters / length of the word (input string) in percentage will give us 100.
- We will be given with a string s, find the strength of the string by determining length of the longest substring without repeating characters.

Example 1:
Input: s = "abytffcde"
Output: 55.55
Explanation: The answer is "abytf", so the string strength here is 55.55%.

Example 2:
Input: s = "cccc"
Output: 25
Explanation: The answer is "c", so the string strength here is 25%.

Example 2:
Input: s = "pwwkew"
Output: 50
Explanation: The answer is "wke",so the string strength here is 50%.
Note: The answer must be a substring, "pwke" is a subsequence and not a substring.

Constraints:

- 0 <= input string length <= 5 * 104
- Input string consists of English letters, digits, symbols and spaces.

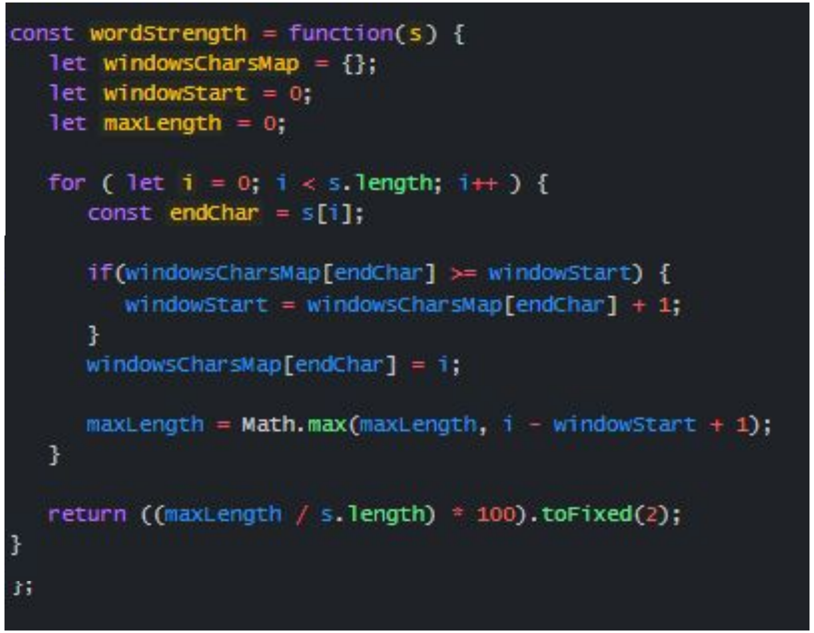
String Strength [Hard]

Approach:

1. We're going to use something called the sliding window approach.
    a. Our sliding window will represent the current substring of non-repeating characters that we are on.
    b. We will be working with a sliding window of variable size.
    c. Our window will grow or shrink in size as we iterate through the string.
2. We will create a for loop representing the end of the window.
3. We will create a WindowsCharsMap that will store characters and its index values as key value pairs.
4. At every iteration we will check for the windowStart that will track the window start position and also the size of that window.
5. When (windowsCharsMap[character at ith pos] >= windowStart)
    a. Then, windowStart = windowsCharsMap[endChar] + 1, which is the index of the character last appeared in the input string plus one to update window size.
6. We will be also updating the maxLength, which is storing the max length of substring with unique characters
7. maxLength = maximum of maxLength or index position of current char - windowStart + 1
8. Return the percentage of maxLength / input string length.

String Strength [Hard]

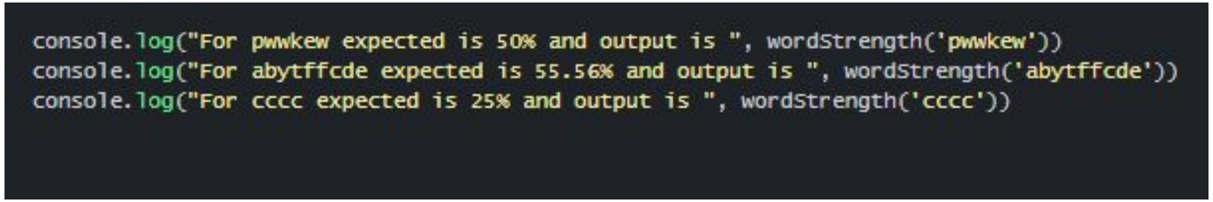
Code:



```
const wordStrength = function(s) {
   let windowsCharsMap = {};
   let windowStart = 0;
   let maxLength = 0;

   for ( let i = 0; i < s.length; i++ ) {
      const endChar = s[i];

      if(windowsCharsMap[endChar] >= windowStart) {
         windowStart = windowsCharsMap[endChar] + 1;
      }
      windowsCharsMap[endChar] = i;

      maxLength = Math.max(maxLength, i - windowStart + 1);
   }

   return ((maxLength / s.length) * 100).toFixed(2);
}

};
```

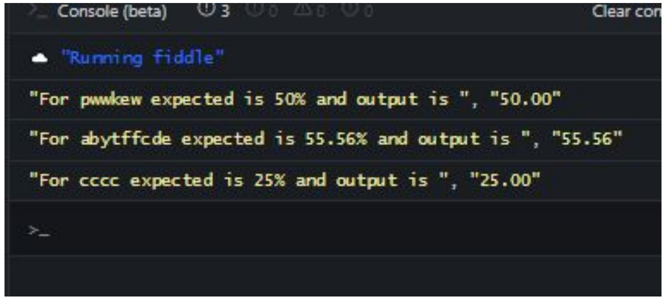
String Strength [Hard]

Tests:



```
console.log("For pwwkew expected is 50% and output is ", wordStrength('pwwkew'))
console.log("For abytffcde expected is 55.56% and output is ", wordStrength('abytffcde'))
console.log("For cccc expected is 25% and output is ", wordStrength('cccc'))
```

Output:



```
"Running fiddle"
"For pwwkew expected is 50% and output is ", "50.00"
"For abytffcde expected is 55.56% and output is ", "55.56"
"For cccc expected is 25% and output is ", "25.00"
>_
```

String Strength [Hard]

Code Link: https://jsfiddle.net/2zdnqjsr/1/

IdeOne link: https://ideone.com/mLBw58

Time Complexity: O(n), where n is the length of input string.
Space Complexity: O(min(m, n), The number of keys in Hash Map is bounded by the size of n and the size of character/alphabet m.

In the Upcoming Class:

- Prefix Matcher
- Last Word Length
- 1D Tetris
- Edit Distance

Thank You!