Minimum Number of Pushes to Type Word II

Description

You are given a string word containing lowercase English letters.

Telephone keypads have keys mapped with **distinct** collections of lowercase English letters, which can be used to form words by pushing them. For example, the key 2 is mapped with ["a","b","c"], we need to push the key one time to type "a", two times to type "b", and three times to type "c" .

It is allowed to remap the keys numbered 2 to 9 to **distinct** collections of letters. The keys can be remapped to **any** amount of letters, but each letter **must** be mapped to **exactly** one key. You need to find the **minimum** number of times the keys will be pushed to type the string word.

Return *the **minimum** number of pushes needed to type* word *after remapping the keys*.

An example mapping of letters to keys on a telephone keypad is given below. Note that 1, *, #, and 0 do **not** map to any letters.

Example 1:

Input: word = "abcde"

Output: 5

Explanation: The remapped keypad given in the image provides the minimum cost.

"a" -> one push on key 2

"b" -> one push on key 3

"c" -> one push on key 4

"d" -> one push on key 5

"e" -> one push on key 6

Total cost is 1 + 1 + 1 + 1 + 1 = 5.

It can be shown that no other mapping can provide a lower cost.

Algorithm

1.) Initialize and Count Frequency of Each Character:

- Create an array arr of size 26 to store the frequency of each character in the word. Each index represents a character from 'a' to 'z'.
- Iterate through the word string and update the frequency array arr for each character.

2.) Sort the Frequency Array in Descending Order:

- Sort the arr array in ascending order.
- Reverse the sorted arr to get the frequencies in descending order.

3.) Calculate Minimum Pushes:

- Initialize a variable res to store the result.
- Iterate through the sorted frequency array:
    - For the first 8 characters (highest frequencies), add their frequencies to res.
    - For the next 8 characters, add twice their frequencies to res.
    - For the next 8 characters, add thrice their frequencies to res.
    - For the remaining characters, add four times their frequencies to res.

4.) Return the Result:

- Return the calculated res as the result.

Pseudocode

```
function minimumPushes(word: String) -> int:
    n = length of word
    arr = array of 26 integers initialized to 0

    # Count frequency of each character
    for i from 0 to n - 1:
        ch = word.charAt(i)
        arr[ch - 'a']++

    # Sort the frequency array in ascending order
```

```
    sort(arr)
    # Reverse the sorted array to get descending order
    reverse(arr)

    res = 0

    # Calculate minimum pushes
    for i from 0 to 25:
        if i <= 7:
            res += arr[i]
        else if i <= 15:
            res += 2 * arr[i]
        else if i <= 23:
            res += 3 * arr[i]
        else:
            res += 4 * arr[i]

    return res
```

Code

```
class Solution {
    public int minimumPushes(String word) {
        int n = word.length();
        int[] arr = new int[26];

        for(int i = 0; i < n; i++){
            char ch = word.charAt(i);
            arr[ch - 'a']++;
        }
```

```
        Arrays.sort(arr);
        for(int i = 0; i < arr.length / 2; i++){
            int temp = arr[i];
            arr[i] = arr[arr.length - 1 - i];
            arr[arr.length - 1 - i] = temp;
        }
        int res = 0;
        for(int i = 0; i < 26; i++){
            if(i <= 7){
                res += arr[i];
            } else if(i <= 15){
                res += 2 * arr[i];
            } else if(i <= 23){
                res += 3 * arr[i];
            } else{
                res += 4 * arr[i];
            }
        }
        return res;
    }
}
```

Conclusion

The minimumPushes function calculates the minimum number of key presses needed to type a word on a simplified keyboard by counting the frequency of each character, sorting the frequencies in descending order, and then assigning key presses based on character frequency. This approach ensures an efficient and accurate calculation of the required key presses.