Statement

↳ Given a string word and an abbreviation abbr, return TRUE if the abbreviation matches the given string. Otherwise, return FALSE.

↳ A certain word "calendar" can be abbreviated as follows:

- "cal3ar" ("cal end ar" skipping three characters "end" from the word "calendar" still matches the provided abbreviation)
- "c6r" ("c alenda r" skipping six characters "alenda" from the word "calendar" still matches the provided abbreviation)

↳ The following are not valid abbreviations

- "c06r" (leading zeroes)
- "cale0ndar" (replaces an empty string)
- "c24r" ("c al enda r" the replaced substrings are adjacent)

Approach

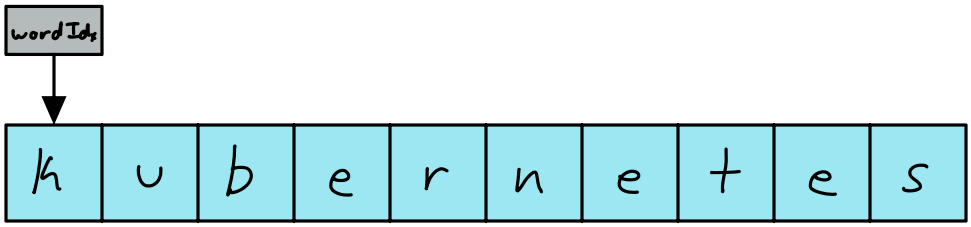
↳ Create two pointers: wordIndex and abbrIndex, both initialized to 0

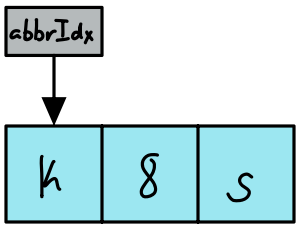
↳ Iterate through the abbreviation string while abbrIndex is less than the length of abbr

- If a digit is encountered at abbr[abbrIndex]:
  - Check if that digit is a leading zero. If it is, return FALSE
  - Calculate the number from abbr and skip that many characters in word
- If a the value at the index is a letter
  - Check for characters that match with word[wordIndex]. If they characters do not match in both strings, return FALSE
  - Increment both wordIndex and abbrIndex by 1.

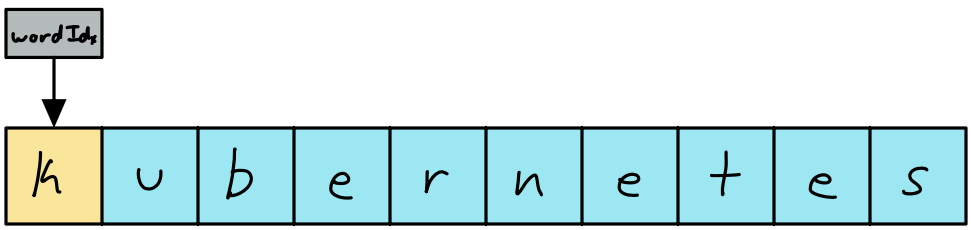
Visualization

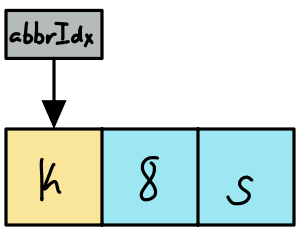
i) Initialize two pointers wordIndex and abbrIndex, both set to 0, for the word and its abbreviation.



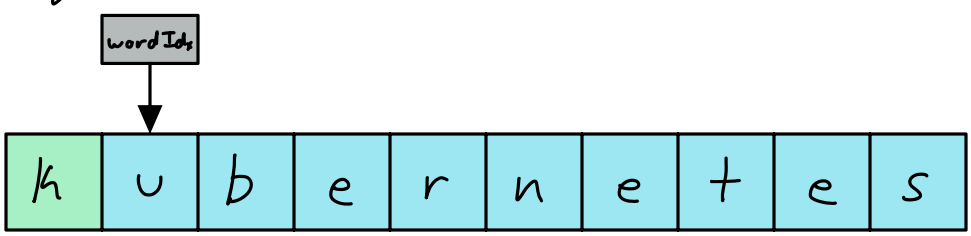


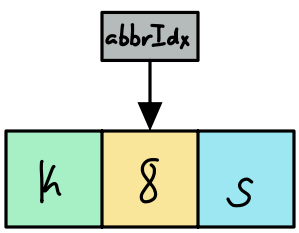
ii) The characters at wordIndex and abbrIndex are the same, so increment both pointers by 1.



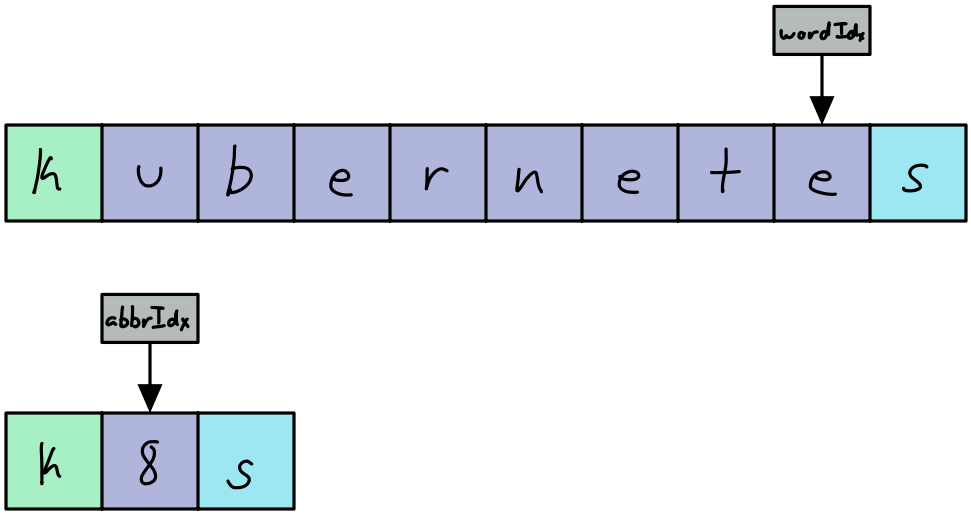


iii) At abbr[1], we come across an integer value. Store this value in num in order to skip that many characters in word.

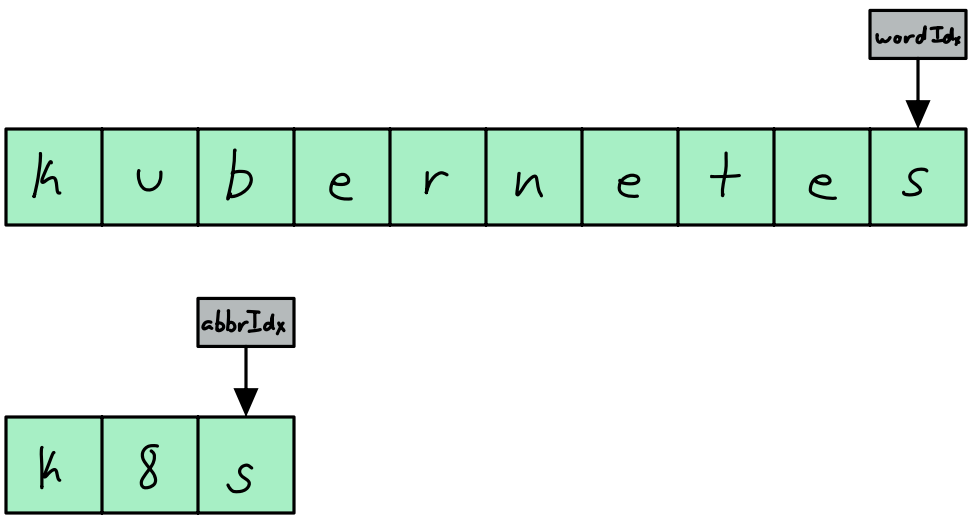




iv) Increment both pointers.



v) The characters at wordIndex and abbrIndex are the same. This is the last character in both strings, return TRUE.



Code

```
bool ValidWordAbbreviation(const string& word, const string& abbr) {
    int wordIdx = 0, abbrIdx = 0;

    while (abbrIndex < abbr.size()) {
        if (isdigit(abbr[abbrIdx])) {
            if (abbr[abbrIndex == '0') {
                return false;
            }
            int num = 0;

            while (abbrIdx < abbr.size() && isdigit(abbr[abbrIdx])) {
                num = num * 10 + (abbr[abbrIdx] - '0');
                abbrIdx++;
            }
            wordIdx += num;
        } else {
            if (wordIdx >= word.size() || word[wordIdx] != abbr[abbrIdx]) {
                return false;
            }
            wordIdx++;
            abbrIdx++;
        }
    }
    return wordIdx == word.size() && abbrIdx == abbr.size();
}
```

Time Complexity

↳ $O(n)$ where n is the length of the string abbr

Space Complexity

↳ $O(1)$