Next Letter (medium)

We'll cover the following

Problem Statement

Given an array of lowercase letters sorted in ascending order, find the **smallest letter** in the given array **greater than a given 'key'**.

Assume the given array is a **circular list**, which means that the last letter is assumed to be connected with the first letter. This also means that the smallest letter in the given array is greater than the last letter of the array and is also the first letter of the array.

Write a function to return the next letter of the given 'key'.

Example 1:

```
Input: ['a', 'c', 'f', 'h'], key = 'f'
Output: 'h'
Explanation: The smallest letter greater than 'f' is 'h' in the given array.
```

Example 2:

```
Input: ['a', 'c', 'f', 'h'], key = 'b'
Output: 'c'
Explanation: The smallest letter greater than 'b' is 'c'.
```

Example 3:

```
Input: ['a', 'c', 'f', 'h'], key = 'm'
Output: 'a'
Explanation: As the array is assumed to be circular, the smallest letter greater than 'm' i
s 'a'.
```

Example 4:

```
Input: ['a', 'c', 'f', 'h'], key = 'h'
Output: 'a'
Explanation: As the array is assumed to be circular, the smallest letter greater than 'h' i
s 'a'.
```

Try it yourself

Try solving this question here:

Java | Python3 | JS | C++

```java
class NextLetter {

  public static char searchNextLetter(char[] letters, char key) {
    // TODO: Write your code here
    return letters[0];
  }

  public static void main(String[] args) {
    System.out.println(NextLetter.searchNextLetter(new char[] { 'a', 'c', 'f', 'h' }, 'f'));
    System.out.println(NextLetter.searchNextLetter(new char[] { 'a', 'c', 'f', 'h' }, 'b'));
    System.out.println(NextLetter.searchNextLetter(new char[] { 'a', 'c', 'f', 'h' }, 'm'));
    System.out.println(NextLetter.searchNextLetter(new char[] { 'a', 'c', 'f', 'h' }, 'h'));
  }
}
```

RUN SAVE RESET

Solution

The problem follows the **Binary Search** pattern. Since **Binary Search** helps us find an element in a sorted array efficiently, we can use a modified version of it to find the next letter.

We can use a similar approach as discussed in Ceiling of a Number. There are a couple of differences though:

1. The array is considered circular, which means if the 'key' is bigger than the last letter of the array or if it

is smaller than the first letter of the array, the key's next letter will be the first letter of the array.

2. The other difference is that we have to find the next biggest letter which can't be equal to the 'key'. This means that we will ignore the case where `key == arr[middle]`. To handle this case, we can update our start range to `start = middle +1`.

In the end, instead of returning the element pointed out by `start`, we have to return the letter pointed out by `start % array_length`. This is needed because of point 2 discussed above. Imagine that the last letter of the array is equal to the 'key'. In that case, we have to return the first letter of the input array.

Code

Here is what our algorithm will look like; the most important changes are in the highlighted lines:

Java | Python3 | C++ | JS

```java
class NextLetter {

  public static char searchNextLetter(char[] letters, char key) {
    int n = letters.length;
    if (key < letters[0] || key > letters[n - 1])
      return letters[0];

    int start = 0, end = n - 1;
    while (start <= end) {
      int mid = start + (end - start) / 2;
      if (key < letters[mid]) {
        end = mid - 1;
      } else { //if (key >= letters[mid]) {
        start = mid + 1;
      }
    }
    // since the loop is running until 'start <= end', so at the end of the while loop, 'start == end+1'
    return letters[start % n];
  }

  public static void main(String[] args) {
    System.out.println(NextLetter.searchNextLetter(new char[] { 'a', 'c', 'f', 'h' }, 'f'));
    System.out.println(NextLetter.searchNextLetter(new char[] { 'a', 'c', 'f', 'h' }, 'b'));
    System.out.println(NextLetter.searchNextLetter(new char[] { 'a', 'c', 'f', 'h' }, 'm'));
    System.out.println(NextLetter.searchNextLetter(new char[] { 'a', 'c', 'f', 'h' }, 'h'));
  }
}
```

RUN | SAVE | RESET

Close

Output 2.295s

```
h
c
a
a
```

Time complexity

Since, we are reducing the search range by half at every step, this means that the time complexity of our algorithm will be $O(logN)$ where 'N' is the total elements in the given array.

Space complexity

The algorithm runs in constant space $O(1)$.

Back
Ceiling of a Number (medium)

Next
Number Range (medium)

Mark as Completed

Report an Issue | Ask a Question