2014. Longest Subsequence Repeated k Times

Problem

Given a string `s` and an integer `k`, you need to find the longest subsequence in the string such that the subsequence is repeated `k` times. The characters in the subsequence are selected conservatively(function in program). If there are multiple such subsequences, you need to return the lexicographically largest one. If the answer is empty, return an empty string.

You can think of it as finding out the maximum-length string which could be a repeated subsequence.

Example

```
Input: s = "abcacb", k = 2
Output: "ab"
Explanation: The longest subsequence that occurs at least 2 times is "ab".
```

Approach

The solution for this problem involves using a BFS (Breadth-First Search) algorithm, along with a queue-based data structure. The solution uses a function `isSubsequence` that returns true if the given subsequence is a subsequence of the string for `k` times and false otherwise. The algorithm first keeps track of the count of each character in the string, a possible character set, and initials an empty queue with an empty string. Then, it dequeues a subsequence from the queue, and if its length times `k` is greater than the length of the string, it returns the answer. Otherwise, it iterates over the possible characters, forms new subsequences, and adds them to the queue if they are subsequences of the string. Finally, the answer is returned.

Example

Using the input example `s = "abcacb", k = 2`, the algorithm works as follows:

1. Count each character's frequency in the string: a:2, b:2, c:2
2. Create a list of possible characters: ['a', 'b', 'c']
3. Initialize the queue with an empty string: [""]
4. Iterate through the BFS queue, forming new subsequences ("a", "ab", "ac", "abc", "acb", "b", "ba", "bc", "bca", "ca", "cac", "cab", "cb", "c", "c").
5. While iterating over the new subsequences, add them to the BFS queue if they're subsequences of the string. For example, "ab" is a subsequence of the string, so it's added to the queue. ["", "a", "ab", "ac", "abc", "acb", "b", "ba", "bc", "bca", "ca", "cac", "cab", "cb", "c", "c"].
6. Return the largest lexicographically-valid subsequence, which is "ab" in this example.

1. Initialize a count array to store the frequency of each character in the string.
2. Initialize a possible Characters list that contains characters occurring `k` times in the string.
3. Initialize a BFS queue with an empty string.
4. While the queue is not empty:
    1. Dequeue one subsequence from the queue.
    2. If the length of the subsequence times `k` is greater than the length of the string, return the answer.
    3. Iterate through the possible characters list, form new subsequences, and add them to the queue if they're subsequences of the string.

Solution

Python

```
python
from collections import deque
from typing import List

class Solution:
    def longestSubsequenceRepeatedK(self, s: str, k: int) -> str:
        def is_subsequence(subseq: str, s: str, k: int) -> bool:
            i = 0
            for c in s:
                if c == subseq[i]:
                    i += 1
                    if i == len(subseq):
                        k -= 1
                        if k == 0:
                            return True
                        i = 0
            return False

        count = [0] * 26
        possible_chars = []
        bfs_queue = deque([""])

        for c in s:
            count[ord(c) - ord('a')] += 1

        for c in range(26):
            if count[c] >= k:
                possible_chars.append(chr(ord('a') + c))

        ans = ""
        while bfs_queue:
            curr_subseq = bfs_queue.popleft()
            if len(curr_subseq) * k > len(s):
                return ans
            for c in possible_chars:
                new_subseq = curr_subseq + c
                if is_subsequence(new_subseq, s, k):
                    bfs_queue.append(new_subseq)
                    ans = new_subseq

        return ans
```

Java

```
java
import java.util.*;

class Solution {
    public String longestSubsequenceRepeatedK(String s, int k) {
        int[] count = new int[26];
        List<Character> possibleChars = new ArrayList<>();
        Queue<String> q = new LinkedList<>(Collections.singletonList(""));

        for (char c : s.toCharArray()) {
            count[c - 'a']++;
        }

        for (char c = 'a'; c <= 'z'; c++) {
            if (count[c - 'a'] >= k) {
                possibleChars.add(c);
            }
        }

        String ans = "";
        while (!q.isEmpty()) {
            String currSubseq = q.poll();
            if (currSubseq.length() * k > s.length()) {
                return ans;
            }
            for (char c : possibleChars) {
                String newSubseq = currSubseq + c;
                if (isSubsequence(newSubseq, s, k)) {
                    q.offer(newSubseq);
                    ans = newSubseq;
                }
            }
        }

        return ans;
    }

    private boolean isSubsequence(String subseq, String s, int k) {
        int i = 0;
        for (char c : s.toCharArray()) {
            if (c == subseq.charAt(i)) {
                i++;
                if (i == subseq.length()) {
                    k--;
                    if (k == 0) {
                        return true;
                    }
                    i = 0;
                }
            }
        }
        return false;
    }
}
```

JavaScript

```
javascript
class Solution {
  longestSubsequenceRepeatedK(s, k) {
    const count = new Array(26).fill(0);
    const possibleChars = [];
    const q = [""];
    let ans = "";

    for (const c of s) {
      count[c.charCodeAt(0) - "a".charCodeAt(0)]++;
    }

    for (let i = 0; i < 26; i++) {
      if (count[i] >= k) {
        possibleChars.push(String.fromCharCode("a".charCodeAt(0) + i));
      }
    }

    while (q.length > 0) {
      const currSubseq = q.shift();
      if (currSubseq.length * k > s.length) {
        return ans;
      }
      for (const c of possibleChars) {
        const newSubseq = currSubseq + c;
        if (this.isSubsequence(newSubseq, s, k)) {
          q.push(newSubseq);
          ans = newSubseq;
        }
      }
    }

    return ans;
  }

  isSubsequence(subseq, s, k) {
    let i = 0;
    for (const c of s) {
      if (c === subseq[i]) {
        i++;
        if (i === subseq.length) {
          k--;
          if (k === 0) {
            return true;
          }
          i = 0;
        }
      }
    }
    return false;
  }
}
```

C++

```
cpp
#include <iostream>
#include <queue>
#include <string>
#include <vector>

class Solution {
public:
  std::string longestSubsequenceRepeatedK(const std::string& s, int k) {
    std::string ans;
    std::vector<int> count(26);
    std::vector<char> possibleChars;
    std::queue<std::string> q{{""}};

    for (const char c : s)
      ++count[c - 'a'];

    for (char c = 'a'; c <= 'z'; ++c)
      if (count[c - 'a'] >= k)
        possibleChars.push_back(c);

    while (!q.empty()) {
      const std::string currSubseq = q.front();
      q.pop();
      if (currSubseq.length() * k > s.length())
        return ans;
      for (const char c : possibleChars) {
        const std::string& newSubseq = currSubseq + c;
        if (IsSubsequence(newSubseq, s, k)) {
          q.push(newSubseq);
          ans = newSubseq;
        }
      }
    }

    return ans;
  }

private:
  bool IsSubsequence(const std::string& subseq, const std::string& s, int k) {
    int i = 0;
    for (const char c : s)
      if (c == subseq[i])
        if (++i == subseq.length()) {
          if (--k == 0)
            return true;
          i = 0;
        }
    return false;
  }
};
```

C#

```
csharp
using System;
using System.Collections.Generic;

public class Solution {
    public string LongestSubsequenceRepeatedK(string s, int k) {
        int[] count = new int[26];
        List<char> possibleChars = new List<char>();
        Queue<string> q = new Queue<string>();
        q.Enqueue("");
        string ans = "";

        foreach (char c in s) {
            count[c - 'a']++;
        }

        for (char c = 'a'; c <= 'z'; c++) {
            if (count[c - 'a'] >= k) {
                possibleChars.Add(c);
            }
        }

        while (q.Count > 0) {
            string currSubseq = q.Dequeue();
            if (currSubseq.Length * k > s.Length) {
                return ans;
            }
            foreach (char c in possibleChars) {
                string newSubseq = currSubseq + c;
                if (IsSubsequence(newSubseq, s, k)) {
                    q.Enqueue(newSubseq);
                    ans = newSubseq;
                }
            }
        }

        return ans;
    }

    private bool IsSubsequence(string subseq, string s, int k) {
        int i = 0;
        foreach (char c in s) {
            if (c == subseq[i]) {
                i++;
                if (i == subseq.Length) {
                    k--;
                    if (k == 0) {
                        return true;
                    }
                    i = 0;
                }
            }
        }
        return false;
    }
}
```

Conclusion