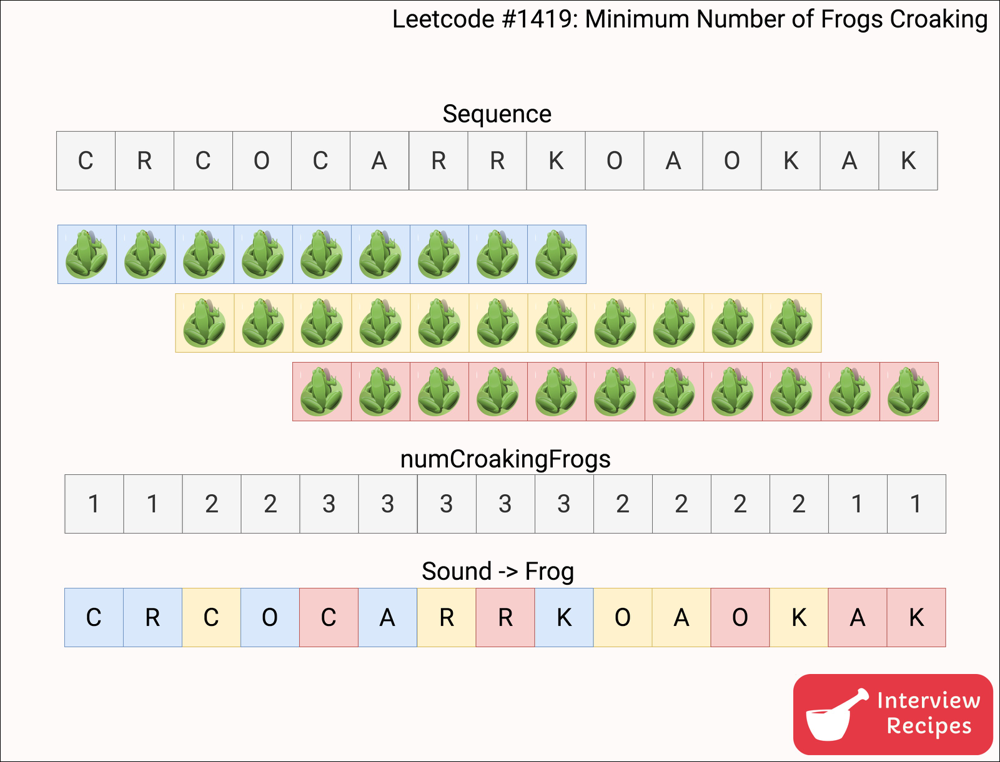
Style 1:

<https://leetcode.com/problems/minimum-number-of-frogs-croaking/discuss/586653/C%2B%2B-Python-Java-Lucid-code-with-documened-comments-%2B-Visualization>

Whenever a 'c' is encountered, you know that a frog has started to croak. This frog can be a new frog or one of the existing one.  
Whenever a 'k' is encountered, you know that one of the frogs have stopped croaking, hence decrement the count of frogs so that whenever a new 'c' is encountered, we can reuse the same frog.

This is the basic idea.  
For 'c', increment in use frogs by one.  
For 'k' decrement the in use count by one.  
The maximum value reached by the in use count is the answer.  


Please upvote if you liked the solution, it would be encouraging.

**C++: Variation 1**

class Solution {

unordered\_map<char, int> frequency; // Stores how many times each sound has occured. Sounds are c, r, o, a, k.

/\*

\* At any time, for a sequence to be valid, number of 'c' must not be less than 'r'.

\* Similarly, #'r' shouldn't less than #'o', and so on.

\*/

bool isStateValid() {

return (frequency['c'] >= frequency['r']) &&

(frequency['r'] >= frequency['o']) &&

(frequency['o'] >= frequency['a']) &&

(frequency['a'] >= frequency['k']);

}

public:

/\*

\* Minimum number of frogs that we need is maximum number of frogs that are croaking

\* simultaneously.

\* Each "croaking" is a sequence of c, r, o, a, and k.

\* Sound is a character in croakSequence.

\*/

int minNumberOfFrogs(string croakSequence) {

int numCroakingFrogs = 0; // Number of distinct frogs that are croaking at any given time.

int answer = 0; // Hold the final answer.

for (char &sound: croakSequence) { // for each sound i.e. character.

frequency[sound]++; // Note the sound.

if (!isStateValid()) { // Make sure we are still in valid state.

return -1;

}

if (sound == 'c') { // New "croaking" always begins at 'c'.

numCroakingFrogs++; // Addional frog for the new "croaking".

} else if (sound == 'k') { // A "croaking" ends at 'k'.

numCroakingFrogs--; // Some frog has stopped croaking now.

}

answer = max(answer, numCroakingFrogs); // Maximum number of frogs that are croaking

// simultaneously over a period.

}

return numCroakingFrogs == 0 ? answer : -1; // Make sure all frogs have completed the croaking.

}

};

**C++ : Variation 2**

int minNumberOfFrogs(string croak) {

int c = 0, r = 0, o = 0, a = 0, k = 0, in\_use = 0, answer = 0;

for (char d:croak) {

switch(d) {

case 'c':

c++;

in\_use++;

break;

case 'r':

r++;

break;

case 'o':

o++;

break;

case 'a':

a++;

break;

case 'k':

k++;

in\_use--;

break;

}

answer = max(answer, in\_use);

if ((c < r) || (r < o) || (o < a) || (a < k))

return -1;

}

if (in\_use == 0 && c == r && c == o && c == a && c == k)

return answer;

return -1;

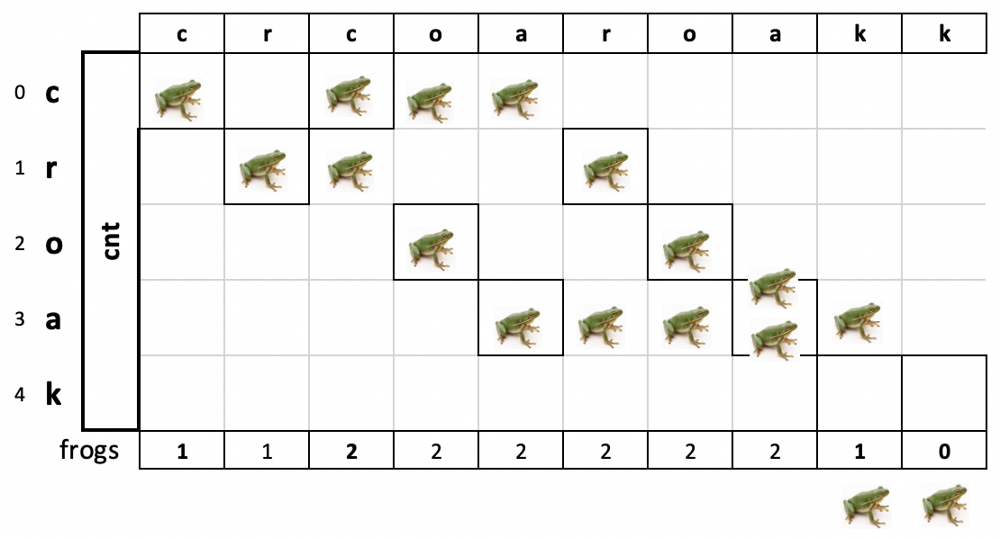
}

Style 2:

<https://leetcode.com/problems/minimum-number-of-frogs-croaking/discuss/586543/C%2B%2BJava-with-picture-simulation>

We can track how many frogs are 'singing' each letter in cnt:

* Increase number of frogs singing this letter, and decrease number singing previous letter.
* When a frog sings 'c', we increase the number of (simultaneous) frogs.
* When a frog sings 'k', we decrease the number of (simultaneous) frogs.
* If some frog is singing a letter, but no frog sang the previous letter, we return -1.



Track and return the maximum number of frogs ever signing together.

Catch: if some frog hasn't finished croaking, we need to return -1.

**C++**

int minNumberOfFrogs(string croakOfFrogs) {

int cnt[5] = {}, frogs = 0, max\_frogs = 0;

for (auto ch : croakOfFrogs) {

auto n = string("croak").find(ch);

++cnt[n];

if (n == 0)

max\_frogs = max(max\_frogs, ++frogs);

else if (--cnt[n - 1] < 0)

return -1;

else if (n == 4)

--frogs;

}

return frogs == 0 ? max\_frogs : -1;

}

**Java**

public int minNumberOfFrogs(String croakOfFrogs) {

int cnt[] = new int[5];

int frogs = 0, max\_frogs = 0;

for (var i = 0; i < croakOfFrogs.length(); ++i) {

var ch = croakOfFrogs.charAt(i);

var n = "croak".indexOf(ch);

++cnt[n];

if (n == 0)

max\_frogs = Math.max(max\_frogs, ++frogs);

else if (--cnt[n - 1] < 0)

return -1;

else if (n == 4)

--frogs;

}

return frogs == 0 ? max\_frogs : -1;

}

<https://leetcode.com/problems/minimum-number-of-frogs-croaking/discuss/586543/C++Java-with-picture-simulation/594851>

Worth noting that:

1. cnt[4] also tells how many frogs finished croaking at the end of the program.
2. middle else if portion else if (--cnt[n - 1] < 0) not only checks if it is sequential/out of order, but also performs a key operation.cnt[n-1-=1;

Well done sir!

<https://leetcode.com/problems/minimum-number-of-frogs-croaking/discuss/586543/C++Java-with-picture-simulation/594851>

Thanks for sharing, I do not quite follow this --cnt[n - 1] < 0  
I think we can just use else if(count[index - 1] < count[index]){ instead, means if current char is more than its previous, which is an invalid case/