880. Decoded String at Index

Leetcode Link

Problem Explanation

The problem wants us to decode a string following specific rules. The encoded string consists of English letters and digits from 2 to 9. When we encounter a letter, we append it to the decoded string. When we encounter a digit, we repeat the entire string accumulated so far `d-1` more times where `d` is the digit. Now, given such an encoded string and an index `K`, we are asked to find and return the `K-th` letter of the decoded string (counting from 1).

Let's take an example to understand this.

Example: Consider the encoded string as "ab2c3". To decode this string, we start reading the characters one by one.

- First, we encountered 'a' which is a letter, so the decoded string becomes "a".
- Then, we encountered 'b', again a letter, so the decoded string becomes "ab".
- Next, we encountered '2' which is a digit. According to the rule, the entire decoded string becomes "abab".
- Then, we encountered 'c', so the decoded becomes "ababc".
- Finally, we encounter '3'. So, the final decoded string becomes "ababcababcababc".

Approach and Algorithm

We need to keep track of the size of the decoded string (let's call it `size`). This `size` will give us the total length of the decoded string till the present character while traversing the encoded string.

We first try building the decoded string in the forward direction till its size became equal to or larger than `K`.

However, doing so may result in extremely large strings as the input size increases, which might be inefficient due to space constraints.

So instead of actually constructing the string, we keep track of its length via `size`. If we come across a digit `D`, we multiply `size` with `D`; and for a letter increment `size`.

After we have the total length of the decoded string, we go backwards through our tape. Whenever `K` is a multiple of `size`, if the character at the `K-th` position in the string is a letter, that would be our answer. If not, we reduce the `size` accordingly and continue.

Python Solution

```python
python
class Solution(object):
    def decodeAtIndex(self, s, k):
        size = 0
        # Calculate size = length of decoded string
        for c in s:
            if c.isdigit():
                size *= int(c)
            else:
                size += 1

        for c in reversed(s):
            k %= size
            if k == 0 and c.isalpha():
                return c

            if c.isdigit():
                size /= int(c)
            else:
                size -= 1
```

Java Solution

```java
java
class Solution {
    public String decodeAtIndex(String S, int K) {
        long size = 0;
        int N = S.length();

        // Find size = length of decoded string
        for (int i = 0; i < N; ++i) {
            char c = S.charAt(i);
            if (Character.isDigit(c))
                size *= c - '0';
            else
                size++;
        }
        for (int i = N-1; i >= 0; --i) {
            char c = S.charAt(i);
            if (Character.isDigit(c)) {
                size /= c - '0';
                K %= size;
            } else if (K % size-- == 0)
                return String.valueOf(c);
        }

        throw null;
    }
}
```

JavaScript Solution

```javascript
var decodeAtIndex = function(s, K) {
    let size = 0;
    for (const ch of s) {
        if (ch >= '0' && ch <= '9') {
            size *= parseInt(ch);
        } else {
            size++;
        }
    }
    for (let i = s.length - 1; i >= 0; i--) {
        let ch = s.charAt(i);
        if (ch >= '0' && ch <= '9') {
            size = Math.floor(size / parseInt(ch));
            K %= size;
        } else if (K % size === 0) {
            return ch;
        } else {
            size--;
        }
    }
};
```

C++ Solution

```cpp
cpp
class Solution {
public:
    string decodeAtIndex(string s, int k) {
        long size = 0;
        for (auto ch : s)
            if (isdigit(ch)) size *= ch - '0';
            else ++size;
        for (int i = s.length() - 1; i >= 0; --i) {
            k %= size;
            if (k == 0 && isalpha(s[i])) return string(1, s[i]);
            if (isdigit(s[i])) size /= s[i] - '0';
            else --size;
        }
        return "";
    }
};
```

C# Solution

```csharp
csharp
public class Solution {
    public string DecodeAtIndex(string S, int K) {
        long size = 0;
        int N = S.Length;

        // Find size = length of decoded string
        for (int i = 0; i < N; ++i) {
            char c = S[i];
            if (char.IsDigit(c))
                size *= c - '0';
            else
                size++;
        }

        for (int i = N-1; i >= 0; --i) {
            char c = S[i];
            if (char.IsDigit(c)) {
                size /= c - '0';
                K %= (int)size;
            } else if (K % size-- == 0)
                return c.ToString();
        }

        throw null;
    }
}
```

Go Solution

```go
go
func decodeAtIndex(S string, K int) string {
    var n int64
    var repeat []int
    var idx []int64
    for _, c := range S {
        if '0' <= c && c <= '9' {
            n *= int64(c - '0')
        } else {
            n++
        }
        repeat = append(repeat, int(c-'0'))
        idx = append(idx, n)
    }
    for i := len(S) - 1; i >= 0; i-- {
        if 'a' <= S[i] && S[i] <= 'z' {
            if K == int(idx[i]) || K == 0 {
                return string(S[i])
            }
        } else {
            if K == int(idx[i]) {
                K--
            }
            K %= int(idx[i]) / repeat[i]
        }
    }
    return ""
}
```

Kotlin Solution

```kotlin
kotlin
fun decodeAtIndex(s: String, K: Int): String? {
    var length: Long = 0
    var i = 0
    while (i < s.length) {
        if (Character.isDigit(s[i])) {
            length *= (s[i] - '0').toLong()
        } else {
            length++
        }
        if (K <= length) break
        i++
    }
    while (i >= 0) {
        if (Character.isDigit(s[i])) {
            length /= (s[i] - '0').toLong()
            K %= length.toInt()
        } else if (length.toLong() == K.toLong() || K == 0) {
            return s[i].toString()
        } else {
            length--
        }
        i--
    }
    return null
}
```

It's important to note that the problem is a decoding issue at its core, and we solved it by capturing the number of times for repeating patterns with numbers and tracking the alphabet content until that point. We then used a backward run against the run-length to find the exact position of our solution character.

We have to handle every character of the to-be-decoded string, so it's important to note that the time complexity of these solutions is O(n), ignoring the possible multiplication operation on the big integer with a specific digit. The space complexity also turns out to be O(n) as we don't store a complete decoded string but only the string length.