Additive Number

Additive number is a string whose digits can form additive sequence.

A valid additive sequence should contain **at least** three numbers. Except for the first two numbers, each subsequent number in the sequence must be the sum of the preceding two.

For example:
`"112358"` is an additive number because the digits can form an additive sequence: `1, 1, 2, 3, 5, 8`.

```
1 + 1 = 2, 1 + 2 = 3, 2 + 3 = 5, 3 + 5 = 8
```

`"199100199"` is also an additive number, the additive sequence is: `1, 99, 100, 199`.

```
1 + 99 = 100, 99 + 100 = 199
```

Note: Numbers in the additive sequence **cannot** have leading zeros, so sequence `1, 2, 03` or `1, 02, 3` is invalid.

Given a string containing only digits `'0'-'9'`, write a function to determine if it's an additive number.

Follow up:
How would you handle overflow for very large input integers?

Credits:
Special thanks to @jeantimex for adding this problem and creating all test cases.

Solution 1

use a helper function to add two strings.

Choose first two number then recursively check.

Note that the length of first two numbers can't be longer than half of the initial string, so the two loops in the first function will end when *i>num.size()/2* and *j>(num.size()-i)/2*, this will actually save a lot of time.

Update the case of heading 0s e.g. "100010" should return false

```
class Solution {
public:
        bool isAdditiveNumber(string num) {
            for(int i=1; i<=num.size()/2; i++){
                for(int j=1; j<=(num.size()-i)/2; j++){
                    if(check(num.substr(0,i), num.substr(i,j), num.substr(i+j)))
return true;
                }
            }
            return false;
        }
        bool check(string num1, string num2, string num){
            if(num1.size()>1 && num1[0]=='0' || num2.size()>1 && num2[0]=='0') re
turn false;
            string sum=add(num1, num2);
            if(num==sum) return true;
            if(num.size()<=sum.size() || sum.compare(num.substr(0,sum.size()))!=0
) return false;
            else return check(num2, sum, num.substr(sum.size()));
        }
        string add(string n, string m){
            string res;
            int i=n.size()-1, j=m.size()-1, carry=0;
            while(i>=0 || j>=0){
                int sum=carry+(i>=0 ? (n[i--]-'0') : 0) + (j>=0?  (m[j--]-'0') :
0);
                res.push_back(sum%10+'0');
                carry=sum/10;
            }
            if(carry) res.push_back(carry+'0');
            reverse(res.begin(), res.end());
            return res;
        }
    };
```

written by zjh08177 original link here

Solution 2

The idea is quite straightforward:

1. Choose the first number `A`, it can be the leftmost `1` up to `i` digits. `i<=(L-1)/2` because the third number should be at least as long as the first number
2. Choose the second number `B`, it can be the leftmost `1` up to `j` digits excluding the first number. the limit for `j` is a little bit tricky, because we don't know whether `A` or `B` is longer. The remaining string (with length `L-j`) after excluding `A` and `B` should have a length of at least max(length `A`, length `B`), where length `A` = `i` and length `B` = `j-i`, thus `L-j >= max(j-i, i)`
3. Calls the recursive checker function and returns true if passes the checker function, or continue to the next choice of `B` (`A`) until there is no more choice for `B` or `A`, in which case returns a false.

Here is the code in Java:

```java
    public boolean isAdditiveNumber(String num) {
        int L = num.length();

        // choose the first number A
        for(int i=1; i<=(L-1)/2; i++) {
            // A cannot start with a 0 if its length is more than 1
            if(num.charAt(0) == '0' && i>=2) break; //previous code: continue;

            // choose the second number B
            for(int j=i+1; L-j>=j-i && L-j>=i; j++) {
                // B cannot start with a 0 if its length is more than 1
                if(num.charAt(i) == '0' && j-i>=2) break; // previous: continue;

                long num1 = Long.parseLong(num.substring(0, i)); // A
                long num2 = Long.parseLong(num.substring(i, j)); // B
                String substr = num.substring(j); // remaining string

                if(isAdditive(substr, num1, num2)) return true; // return true if
passes isAdditive test
                // else continue; // continue for loop if does not pass isAdditiv
e test
            }
        }
        return false; // does not pass isAdditive test, thus is not additive
    }

    // Recursively checks if a string is additive
    public boolean isAdditive(String str, long num1, long num2) {
        if(str.equals("")) return true; // reaches the end of string means a yes

        long sum = num1+num2;
        String s = ((Long)sum).toString();
        if(!str.startsWith(s)) return false; // if string does not start with sum
of num1 and num2, returns false

        return isAdditive(str.substring(s.length()), num2, sum); // recursively c
hecks the remaining string
    }
```

written by GWTW original link here

Solution 3

The idea is quite straight forward. Generate the first and second of the sequence, check if the rest of the string match the sum recursively. `i` and `j` are length of the first and second number. `i` should in the range of `[0, n/2]`. The length of their sum should `>= max(i,j)`

Java Recursive

```
import java.math.BigInteger;

public class Solution {
    public boolean isAdditiveNumber(String num) {
        int n = num.length();
        for (int i = 1; i <= n / 2; ++i) {
            if (num.charAt(0) == '0' && i > 1) return false;
            BigInteger x1 = new BigInteger(num.substring(0, i));
            for (int j = 1; Math.max(j, i) <= n - i - j; ++j) {
                if (num.charAt(i) == '0' && j > 1) break;
                BigInteger x2 = new BigInteger(num.substring(i, i + j));
                if (isValid(x1, x2, j + i, num)) return true;
            }
        }
        return false;
    }
    private boolean isValid(BigInteger x1, BigInteger x2, int start, String num)
{
        if (start == num.length()) return true;
        x2 = x2.add(x1);
        x1 = x2.subtract(x1);
        String sum = x2.toString();
        return num.startsWith(sum, start) && isValid(x1, x2, start + sum.length()
, num);
    }
}
// Runtime: 8ms
```

Since `isValid` is a tail recursion it is very easy to turn it into a loop.

Java Iterative

```java
public class Solution {
    public boolean isAdditiveNumber(String num) {
        int n = num.length();
        for (int i = 1; i <= n / 2; ++i)
            for (int j = 1; Math.max(j, i) <= n - i - j; ++j)
                if (isValid(i, j, num)) return true;
        return false;
    }
    private boolean isValid(int i, int j, String num) {
        if (num.charAt(0) == '0' && i > 1) return false;
        if (num.charAt(i) == '0' && j > 1) return false;
        String sum;
        BigInteger x1 = new BigInteger(num.substring(0, i));
        BigInteger x2 = new BigInteger(num.substring(i, i + j));
        for (int start = i + j; start != num.length(); start += sum.length()) {
            x2 = x2.add(x1);
            x1 = x2.subtract(x1);
            sum = x2.toString();
            if (!num.startsWith(sum, start)) return false;
        }
        return true;
    }
}
// Runtime: 9ms
```

If no overflow, instead of BigInteger we can consider to use Long which is a lot faster.

Java Iterative Using Long

```java
public class Solution {
    public boolean isAdditiveNumber(String num) {
        int n = num.length();
        for (int i = 1; i <= n / 2; ++i)
            for (int j = 1; Math.max(j, i) <= n - i - j; ++j)
                if (isValid(i, j, num)) return true;
        return false;
    }
    private boolean isValid(int i, int j, String num) {
        if (num.charAt(0) == '0' && i > 1) return false;
        if (num.charAt(i) == '0' && j > 1) return false;
        String sum;
        Long x1 = Long.parseLong(num.substring(0, i));
        Long x2 = Long.parseLong(num.substring(i, i + j));
        for (int start = i + j; start != num.length(); start += sum.length()) {
            x2 = x2 + x1;
            x1 = x2 - x1;
            sum = x2.toString();
            if (!num.startsWith(sum, start)) return false;
        }
        return true;
    }
}
// Runtime: 3ms
```

written by dietpepsi original link here

From Leetcoder.