Leetcode

Leetcode Solution 1:-

```
class Solution {
    public boolean isAdditiveNumber(String num) {
        int n = num.length();
        for (int i = 1; i < Math.min(n - 1, 19); ++i) {
            for (int j = i + 1; j < Math.min(n, i + 19); ++j) {
                if (i > 1 && num.charAt(0) == '0') {
                    break;
                }
                if (j - i > 1 && num.charAt(i) == '0') {
                    continue;
                }
                long a = Long.parseLong(num.substring(0, i));
                long b = Long.parseLong(num.substring(i, j));
                if (dfs(a, b, num.substring(j))) {
                    return true;
                }
            }
        }
        return false;
    }

    private boolean dfs(long a, long b, String num) {
        if ("".equals(num)) {
            return true;
        }
        if (a + b > 0 && num.charAt(0) == '0') {
            return false;
        }
        for (int i = 1; i < Math.min(num.length() + 1, 19); ++i) {
            if (a + b == Long.parseLong(num.substring(0, i))) {
                if (dfs(b, a + b, num.substring(i))) {
                    return true;
                }
            }
        }
        return false;
    }
}
```

Leetcode Solution 2:-

```
class Solution {
    public boolean isPowerOfThree(int n) {
        return n > 0 && 1162261467 % n == 0;
    }
}
```

Leetcode Solution 3:-

```
class Solution {
    public int strongPasswordChecker(String password) {
        int types = countTypes(password);
        int n = password.length();
        if (n < 6) {
            return Math.max(6 - n, 3 - types);
        }
        char[] chars = password.toCharArray();
        if (n <= 20) {
            int replace = 0;
            int cnt = 0;
            char prev = '~';
            for (char curr : chars) {
```

```
        if (curr == prev) {
            ++cnt;
        } else {
            replace += cnt / 3;
            cnt = 1;
            prev = curr;
        }
    }
    replace += cnt / 3;
    return Math.max(replace, 3 - types);
}
int replace = 0, remove = n - 20;
int remove2 = 0;
int cnt = 0;
char prev = '~';
for (char curr : chars) {
    if (curr == prev) {
        ++cnt;
    } else {
        if (remove > 0 && cnt >= 3) {
            if (cnt % 3 == 0) {
                --remove;
                --replace;
            } else if (cnt % 3 == 1) {
                ++remove2;
            }
        }
        replace += cnt / 3;
        cnt = 1;
        prev = curr;
    }
}
if (remove > 0 && cnt >= 3) {
    if (cnt % 3 == 0) {
        --remove;
        --replace;
    } else if (cnt % 3 == 1) {
        ++remove2;
    }
}
replace += cnt / 3;
```

```
        int use2 = Math.min(Math.min(replace, remove2), remove / 2);
        replace -= use2;
        remove -= use2 * 2;

        int use3 = Math.min(replace, remove / 3);
        replace -= use3;
        remove -= use3 * 3;
        return (n - 20) + Math.max(replace, 3 - types);
    }

    private int countTypes(String s) {
        int a = 0, b = 0, c = 0;
        for (char ch : s.toCharArray()) {
            if (Character.isLowerCase(ch)) {
                a = 1;
            } else if (Character.isUpperCase(ch)) {
                b = 1;
            } else if (Character.isDigit(ch)) {
                c = 1;
            }
        }
        return a + b + c;
    }
}
```

Leetcode Solution 4:-

```
class Solution {
    public boolean checkPerfectNumber(int num) {
        if (num == 1) {
            return false;
        }
        int s = 1;
        for (int i = 2; i * i <= num; ++i) {
            if (num % i == 0) {
                s += i;
                if (i != num / i) {
                    s += num / i;
                }
            }
        }
        return s == num;
    }
}
```