Counting Numbers

Your task is to count the number of integers between a and b where no two adjacent digits are the same.

Input

The only input line has two integers a and b.

Output

Print one integer: the answer to the problem.

Constraints

0 <= a <= b <= 10^18

Example

Input:

123 321

Output:

171

Problem Breakdown

Lets understand the problem first

Count numbers in [a,b] such that:

-> no two adjacent digits are equal.

Example:
112 → ❌
123 → ✔️
909 → ✔️ (9 ≠ 0, 0 ≠ 9)
100 → ❌ (0 == 0)

We want:

answer = count(b) − count(a − 1)

So we just need a function count(x) =
how many numbers in [0, x] have no equal adjacent digits?

We'll build that using digit DP.

Digit DP State

`dfs(pos, prevDigit, tight, leadingZero)`

Where:

- pos = current digit index (0 ... len-1)
- prevDigit = digit we placed just before this
  - range: 0..9
  - plus a sentinel "no previous digit" → we use 10
- tight =
  - 1 → must not exceed N's digit
  - 0 → free (can use 0..9)
- leadingZero =
  - 1 → we still haven't placed any real digit (all zeros so far)
  - 0 → number has started

Why leadingZero?

Because leading zeros should not count as digits, so:
00023 shouldn't check adjacency between the leading zeros.
When leadingZero = 1, we set prevDigit = 10 (meaning "no previous digit").

Transition Logic

At position pos, we try digits:

`digit = 0 .. (tight ? digits[pos] : 9)`

Then:

If leadingZero = 1 and digit == 0:

- still leadingZero
- prevDigit stays 10
- This is safe: first real digit hasn't started.

If not leadingZero:

We check adjacency rule:

```
if (digit == prevDigit)
    skip this choice
```

Determine next states:

```
nextTight = tight AND (digit == limit)
nextPrev = (digit if digit != 0 OR not leadingZero else 10)
nextLeadingZero = (leadingZero && digit == 0)
```

When pos == len, return 1 (valid number).

Code

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll = long long;

vector<int> digits;
ll dp[20][11][2][2];
// pos (0..18), prev(0..9 and 10=special), tight(0/1), leadingZero(0/1)

ll dfs(int pos, int prevDigit, int tight, int leadingZero) {
    if (pos == (int)digits.size()) {
        return 1; // One valid number built
    }

    ll &res = dp[pos][prevDigit][tight][leadingZero];
    if (res != -1) return res;
    res = 0;

    int limit = tight ? digits[pos] : 9;
    for (int d = 0; d <= limit; d++) {

        // If number has started, apply adjacency rule
        if (!leadingZero) {
            if (d == prevDigit) continue; // adjacent equal digits not allowed
        }

        int nextTight = tight && (d == limit);
        int nextLeadingZero = (leadingZero && d == 0);

        int nextPrev;
        if (nextLeadingZero)
            nextPrev = 10;   // still no previous digit
        else
            nextPrev = d;    // real previous digit

        res += dfs(pos + 1, nextPrev, nextTight, nextLeadingZero);
    }
    return res;
}

ll countUpTo(long long x) {
    if (x < 0) return 0;

    digits.clear();
    if (x == 0) digits.push_back(0);

    while (x > 0) { digits.push_back(x % 10); x /= 10; }
    reverse(digits.begin(), digits.end());

    memset(dp, -1, sizeof(dp));

    return dfs(0, 10, 1, 1); // start: no prevDigit (10), tight=1, leadingZero=1
}

int main() {
    long long a, b;
    cin >> a >> b;
    cout << countUpTo(b) - countUpTo(a - 1) << "\n";
}
```