**Code Jam 2015 - Round 1B**

# Analysis: Counter Culture

For the small dataset, it suffices to use a breadth-first search to find the minimum number of moves required to generate all numbers from 1 to $10^6$. You will never want to construct a number larger than $10^6$ and later flip it to get a smaller number, so you only need to consider $10^6$ states.

For the large dataset, the following solution works. The key ideas are:

- You should first build the numbers 10, 100, 1000, ... until you get a power of 10 with the same number of digits as N, and then build N.
- You should make at most one flip while the number is at a given number of digits.

Our algorithm works in two parts.

## Part 1: Get to the right number of digits as fast as possible.

To get from a 1 followed by X 0s to a 1 followed by X+1 0s: first count until the right half of the number is filled with 9s. (When the length is odd, make the left half shorter than the right half.) Then flip, then count until the number is all 9s. Then add 1. (To get from 1 to 10, obviously we don't need to flip.)

## Part 2: Either count directly to the answer, or do some counting, flip, and then do some more counting, whichever is faster.

Once we're at the right number of digits, we use a similar algorithm to produce N: count until the right half looks like the left half of N in reverse, then flip, then count up to the target. For example, to get from 100000 to 123456:

- count to 100321
- flip to get 123001
- count to 123456.

If the left half is just a 1 followed by 0s, we can skip the first two steps and just count to N.

When the right half of N is all zeroes the above method doesn't work, because after we flip, the right half ends with 1. Instead, we make the right half look like the left half of N-1, flip, then count up to N. For example, to get from 100000 to 300000:

- count to 100992
- flip to get 299001
- count to 300000.

However, as before, if the left half of N-1 is a 1 followed by 0s, we can skip the first two steps. For example, to get from 100000 to 101000, it's best to count up directly.

Why do we need to do at most one flip? The goal of the flip is to allow us to reach the left half of the number as quickly as possible. Raising the right half to the desired value is better done by directly counting the number up. In other words, there is no added value in doing multiple flips.

Sample implementation in C++:

```
#include <cstdio>
#include <cstdlib>

#include <string>
#include <algorithm>

using namespace std;

long long p10[10];  // p10[i] == 10^i

bool is_1_followed_by_0s(string S) {
  reverse(S.begin(), S.end());
  return atoi(S.c_str()) == 1;
}

int solve(long long N) {
  if (N < 10) return N;  // Trivial case.

  char X[20]; sprintf(X, "%lld", N);
  string S = X;
  int M = S.length();  // Number of digits of N.

  // Starts from 1.
  int ans = 1;

  // Part 1: from 1, get to the M digits as fast as possible.
  for (int d = 1; d < M; d++) {
    // For digits = 7, it starts from 7 digits:    1000000
    ans += p10[(d + 1) / 2] - 1; // Count up 9999: 1009999
    if (d > 1) ans++;            // Flip once:     9999001
    ans += p10[d / 2] - 1;       // Count up 999: 10000000
  }


  // Part 2:

  // Split N into two halves. For example N = "1234567"
  string L = S.substr(0, M / 2);  // L = "123"
  string R = S.substr(M / 2);     // R = "4567"

  // Handles the case where the right half is all zeroes.
  if (atoi(R.c_str()) == 0) return solve(N - 1) + 1;

  // Special case: Count directly to the answer.
  if (is_1_followed_by_0s(L))
    return ans + atoi(R.c_str());

  // Count until the right half looks like the left half of N
  // in reverse. In this case, count from 1000000 to 1000321.
  reverse(L.begin(), L.end());
  ans += atoi(L.c_str());

  // Flip 1000321 to 1230001.
  ans++;

  // Count up 4566 to the target from 1230001 to 1234567.
  ans += atoi(R.c_str()) - 1;
```

```
  return ans;
}

int main() {
  p10[0] = 1;
  for (int i = 1; i < 10; i++)
    p10[i] = p10[i - 1] * 10;

  long long T, N;
  scanf("%lld", &T);
  for (int TC = 1; TC <= T; TC++) {
    scanf("%lld", &N);
    printf("Case #%d: %d\n", TC, solve(N));
  }
}
```