Code Jam 2008 - Round 1C

Analysis: Ugly Numbers

As the problem clearly states, there are 3^{D-1} ways you need to consider. When **D** is no larger than 13, as it is in the small dataset, one may simply generate all the possible combinations and calculate each of them. However, with **D** as large as 40, brute force is clearly too slow.

As you might expect from a programming contest, the magic term is dynamic programming. This is the first problem in Google Code Jam 2008 that falls into the standard dynamic programming (DP) category; and one can be assured that there are more to come.

As in any DP problem, the first task is to structure the problem in a good way so there are not too many numbers to compute, and each number can be computed easily from previous ones. If you observe the solutions of the top scorers in this sub-round, you will see, although their algorithms vary to some degre, they all contain the following magic number

```
2 · 3 · 5 · 7 = 210.
```

Let us say we have two numbers, x and y, knowing the ugliness of x and y is not enough to decide whether x + y and x - y are ugly. On the other hand, we do not need the exact value of x and y. Knowing x % 210 and y % 210 is enough for us to decide if, say, x + y is ugly or not.

For those who prefer mathematical terms, we are using the Chinese remainder theorem. Our problem can be viewed as arithmetics on the cyclic group (Z_{210}, +).

So, let us outline the most central step of our dynamic programming solution. We want to compute

```
dyn[i][x] := number of ways we get an expression evaluating
             to x (mod 210) if we only consider the first i
             characters of the string. (*)
```

So, we have only 40·210 numbers to consider. For each dyn[i][x], we try all possible positions for inserting the last '+' or '-' sign. If the last sign was a '+' before position j (j<i), and the number formed by digits from position j to position i is d, then we want to know dyn[j-1][(x-d)%210]. On the other hand, if the sign inserted was a '-', then we want to look at dyn[j-1][(x+d)%210].

Here is a masterful implementation from our own Derek Kisman. His complete C++ solution follows the idea above, with a little twist and some nice programming tricks.

```
#define MOD (2*3*5*7)
string s;
long long dyn[41][MOD];

main() {
  int N, prob=1;
  for (cin >> N; N--;) {
    cin >> s;
    memset(dyn, 0, sizeof(dyn));
    dyn[0][0] = 1;
    for (int i = 0; i < s.size(); i++)
    for (int sgn = (i==0) ? 1 : -1; sgn <= 1; sgn += 2) {
      int cur = 0;
      for (int j = i; j < s.size(); j++) {
        cur = (cur*10 + s[j]-'0')%MOD;
        for (int x = 0; x < MOD; x++)
          dyn[j+1][(x+sgn*cur+MOD)%MOD] += dyn[i][x];
      }
    }
    long long ret = 0;
    for (int x = 0; x < MOD; x++)
      if (x%2 == 0 || x%3 == 0 || x%5 == 0 || x%7 == 0)
        ret += dyn[s.size()][x];
    cout << "Case #" << prob++ << ": " << ret << endl;
  }
}
```

More information:

Dynamic programming - Chinese remainder theorem