Tuenti Programming Challenge 4 Solutionss

Ángel García Gómez

May 8, 2014

Contents

1 Challenge 01 - Anonymous Poll

In this problem, you were given a big file with a lot of entries in the following format:

```
Name,Gender,Age,Studies,Year
```

Then, you received some queries, which consisted in gender, age, studies and year, and you had to output all names with the given data, ordered lexicographically.

In my solution, I created a map from "Poll" (a struct with gender, age, studies and year) to a vector of strings, which was a list of all names with that poll. I fill this map at the beginning of the program and then iterate sorting all vectors.

Finally, for each query, the only thing to do is printing the list of names given by the map for the given poll.

2 Challenge 02 - F1 - Bird's-eye Circuit

This problem asked you to draw a 2D circuit following some rules. It can have a very simple implementation using some tricks.

One of them is making a pair of arrays with the unitary vectors for all four directions, to avoid writing a lot of "ifs". Then I chose the directions so that, whenever a slash or backslash appeared, it was easy to compute the next direction (a simple XOR operation).

Finally, I perform one lap to the circuit, keeping track of the maximum and minimum values of the coordinates. With this, it is easy to know the final width and height of the circuit and the starting position. The only thing left is performing a second lap writing the correct symbols in a matrix.

Code 1: Drawing the circuit

```
1  //Directions: right, down, up, left
   const int dx[] = {1, 0, 0, -1};
   const int dy[] = {0, 1, -1, 0};

   ...
6
   for (int i = start; i < start + length; ++i) {

       //Turn left or right
       if (track[i] == '/') d ^= 2;
11     if (track[i] == '\\') d ^= 1;

       //Draw the circuit
       if (track[i] == '-') {
         if (d == 0 or d == 3) map[y][x] = '-';
16       else map[y][x] = '|';
       }
       else map[y][x] = track[i];

       //Next position
21     x += dx[d];
       y += dy[d];
   }
```

3 Challenge 03 - The Gambler's Club - Monkey Island 2

This was the easiest problem of the contest. You were given a web site in which you could write two numbers and get their password, and then you were told to find out the algorithm.

After a few tries, I quickly found out that the algorithm returned the hypotenuse, given the two legs.

Code 2: Pythagorean theorem

```
//Number of cases
int N;
cin >> N;

for (int i = 0; i < N; ++i) {
  //Each case is solved here
  int X, Y;
  cin >> X >> Y;

  //Pythagorean theorem
  cout << round(100 * sqrt(double(X * X + Y * Y))) / 100 << endl;
}
```

4 Challenge 04 - Shape shifters

This problem consisted of finding the shortest path between two given strings. In each step, it was possible to change only one character of the current string, and not every string was possible: you were given a list with all the allowed middle states.

I created a graph, in which each node was one of the allowed strings, and computed its adjacency list. After that, the problem was solved with a breadth-first search.

Code 3: BFS

```
//Find the shortest path with a breadth-first search
vector<int> previo(n, -1);
queue<int> q;
q.push(istart);
previo[istart] = -2;
while (not q.empty()) {
  int u = q.front();
  q.pop();
  if (u == iend) break;
  int e = list[u].size();
  for (int i = 0; i < e; ++i) {
    int v = list[u][i];
    if (previo[v] == -1) {
      previo[v] = u;
      q.push(v);
    }
  }
}
```

5 Challenge 05 - Tribblemaker

Given a 8×8 grid representing a state of Conway's Game of Life, you were asked when did the cycle begin and how long it was (it was guaranteed that there was a cycle, and that it started before 100 steps).

The only possible thing to do was simulating and storing all seen states until finding a repeated one. Since it was an 8×8 grid and each cell could be either alive or dead, storing and searching could be done efficiently in a map using a 64 bits unsigned integer, even although I didn't do that.

Code 4: Computing the next generation

```
void next_generation(vector<vector<int> >& current_grid) {
  vector<vector<int> > next_grid(10, vector<int>(10, 0));
  for (int i = 1; i <= 8; ++i) {
    for (int j = 1; j <= 8; ++j) {
      int alive = 0;
      for(int d = 0; d < 8; ++d) alive += current_grid[i + di[d]][j + dj[d]];
      if (current_grid[i][j] == 0) {
        if (alive == 3) next_grid[i][j] = 1;
      }
      else if (alive == 2 or alive == 3) next_grid[i][j] = 1;
    }
  }
  current_grid = next_grid;
}
```

Code 5: Simulating

```
//Create a map which, given a grid, return its generation number
map<vector<vector<int> >, int> generation;

//Simulate generations until there is a repetition
int current_generation = 0;
do {
  generation[grid] = current_generation;
  ++current_generation;
  next_generation(grid);
} while (generation.count(grid) == 0);

//Print result
int starting_gen = generation[grid];
cout << starting_gen << ' ' << current_generation - starting_gen << endl;
```

6 Challenge 06 - Man in the middle

One of my favorite problems of the contest. You were given two *JavaScript* files, *server.js* and *client.js*, and a Man-in-the-Middle service, and were asked to obtain the answer to some keyphrase from the server.

The main idea of my solution was pretending to be the server to the client, while at the same time pretending to be the client to the server, and getting the secret message before any of them discovers it. The easiest way to explain how I did this is showing the code, but the basic idea was to replace each message for its equivalent generated by my own code.

For example, when the client sends its key, I compute the answer to that key, but send my key instead. When the server responses, that response is for my key, so I replace it with the one computed by myself, which is for the client's key.

7 Challenge 07 - Yes we scan

In this problem, you were given a file with pairs of numbers (person IDs). Let's suppose that each person is an independent node, and that each pair given in the file means adding one edge between the two given nodes. Then, given two IDs, you were asked in which line those two nodes started to be in the same connected component.

Using a disjoint-set data structure, I added the edges one by one, merging connected components efficiently, until the two given nodes (the terrorists) were in the same connected components.

Code 6: Add edges to the forest

```
//Check the edges until A is in the same tree than B
int p = 0;
while (p < l and Root(A, root) != Root(B, root)) {
  int x = edge[p].first, y = edge[p].second;
  int rx = Root(x, root), ry = Root(y, root);

  //Join two trees
  root[rx] = ry;
  ++p;
}

//Print solution
if (Root(A, root) == Root(B, root)) cout << "Connected at " << p - 1 << endl;
else cout << "Not connected" << endl;
```

8 Challenge 08 - Tuenti Restructuration

This problem asked you for the minimum time needed to transform one table (3×3 grid) into another, and the only allowed move was swapping two adjacent positions. First of all, I coded the starting table as vector of nine integers, and hardcoded it's adjacent positions.

Code 7: Table

```
//Positions in the table
//0 1 2
//3 4 5
//6 7 8

//Pairs of adjacent positions
int sw[][2] = {
  {0, 1}, {1, 2}, {3, 4}, {4, 5}, {6, 7}, {7, 8},
  {0, 3}, {1, 4}, {2, 5}, {3, 6}, {4, 7}, {5, 8}
};
```

Then, the ending table was a certain permutation of those nine integers. Hence, to find the shortest path, it could be done with a breadth-first search in the graph in which the nodes are all permutations of nine elements, and there is an edge between two permutations if one can be obtained from the other swapping two table-adjacent positions.

Furthermore, there is only need to perform one breadth-first search, and not one for each table, since the starting permutation is always the identity.

Code 8: BFS

```
//With a breadth-first search, compute the shortest path
//from identity to every other permutation
dist = vector<int>(nine_factorial, -1);
queue<int> q;
q.push(0);
dist[0] = 0;
while (not q.empty()) {
  int u = q.front();
  q.pop();
  for (int i = 0; i < 12; ++i) {
    vector<int> p = permutation[u];
    swap(p[sw[i][0]], p[sw[i][1]]);
    int v = index[p];
    if (dist[v] == -1) {
      dist[v] = dist[u] + 1;
      q.push(v);
    }
  }
}
```

9 Challenge 09 - Bendito Caos

You are given a directed graph with two cities and are asked how many cars can get in an hour from one city to the other. At some point in the problem statement is the following observation: "When you begin the calculations the roads are already full". This observation makes this problem a maximum-flow problem. The capacity of each road is its maximum velocity, divided by the distance between the center of cars (5 m), multiplied by the number of lanes in that road.

Code 9: Maximum flow

```
//Computes the maximum flow from node 's' to node 't',
//in the graph defined by 'adj' adjacency list and
// 'E' as the list of edges.
ll maxflow(int s, int t, VVI& adj, vector<Edge>& E) {
  for (int i = 0; i < E.size(); ++i) E[i].flow = 0;
  ll flow = 0, bot;
  while (bfs(s, t, adj, E)) {
    act = VI(adj.size(), 0);
    while ((bot = dfs(s, t, INF, adj, E))) flow += bot;
  }
  return flow;
}
```

10 Challenge 10 - Random Password

You are given an IP address... and nothing else. Luckily, I remembered a hint from last year Tuenti Challenge: that some program was made using *Kate* Editor. So I put *index.php~* on the browser and I got this:

Code 10: Index.php~

```
<?php
srand(mktime(date("H"), date("i"), 0) * posix_getppid());
header("Date: ".gmdate("D, d M Y H:i:s")." GMT");
echo rand()==$_GET['password'] ?
json_decode(file_get_contents('../keys.json'),true)[$_GET['input']]:"wrong!";
?>
```

Then, the password was generated randomly but was valid in a whole minute. To be able to generate that password, I needed to know what was the value of the *PPID*. I made a *php* program that bruteforced it, and I got the value 1336.

Code 11: Bruteforcing the PPID

```
<?php
//Bruteforces until finding ppid
for ($i = 0; $i < 50000; $i++) {
  date_default_timezone_set("GMT");
  srand(mktime(date("H"), date("i"), 0) * $i);
  $url="http://random.contest.tuenti.net/?input=bfd84a3611&password=".rand();
  $result = file_get_contents($url);
  if ($result != "wrong!") {
    file_put_contents("ppid.txt", $i);
    break;
  }
}
?>
```

Finally, once the *PPID* was known, the problem was almost solved. I only had to made another *php* program that used that used the correct value to get the response to my input.

11 Challenge 11 - Pheasant

In my opinion, the hardest problem. You were given a lot of files encrypted with AES in ECB mode with a 32 bytes key. Given a file and 29 characters of that key, you had to be able to decrypt the file to get the answer to the problem.

Of course, there is a command line tool that makes the decryption, but it took too much time to try all possibilities for the three left characters. I had to use an open source library (*OpenSSL*) to program a function that decrypted the file given the key. This was much faster than the command line tool, but still too slow. Since I know that the first characters of the file are the ID corresponding to that file, if after decrypting the first block, its beginning is wrong, I stop the decryption and continue to the next three characters.

Code 12: Decrypting AES in ECB mode, 256 bits

```
//Decrypts a block encoded in aes-256-ecb (encrypted) with 'aes_key' and
//stores text in 'decrypted'
void myDecrypt(unsigned char * encrypted, unsigned char * decrypted,
               unsigned char * aes_key) {

  memset(decrypted, 0, sizeof(decrypted));
  for (int i = 0; i < inputslength; i += AES_BLOCK_SIZE) {
    AES_KEY dec_key;
    unsigned char dec_out[AES_BLOCK_SIZE];
    memset(dec_out, 0, sizeof(dec_out));
    AES_set_decrypt_key(aes_key, keylength, &dec_key);
    AES_ecb_encrypt(encrypted + i, dec_out, &dec_key, AES_DECRYPT);
    if (i == 0) {
      if (not isdigit(dec_out[0]) or not isdigit(dec_out[1]) or
          not isdigit(dec_out[2]) or not isdigit(dec_out[3])) {

        decrypted[0] = '\0';
        //If it is not a digit, the password is wrong
        return;
      }
    }
    for (int j = 0; j < AES_BLOCK_SIZE and i + j < inputslength; ++j)
      decrypted[i + j] = dec_out[j];
  }

  //Sets the end of string mark
  for (int i = 0; i < inputslength; ++i) {
    if (decrypted[i] != '\n' and decrypted[i] != ' ' and
        not isdigit(decrypted[i])) decrypted[i] = '\0';
  }
  decrypted[inputslength] = '\0';
}
```

Finally, one optimization to decrypt the minimum number of files, was to decrypt first those with later last event (which was known).

12 Challenge 12 - Taxi Driver

Another easy breadth-first search, where each node was a position with the direction from which that position was entered. From that node, you could get to the next node in that direction, or turn to the right (change direction) and go to that other position.

Code 13: Breadth-first search

```
//There are four starting nodes: the starting position
//with each of the four directions
for (int d = 0; d < 4; ++d) {
  dist[start[0]][start[1]][d] = 0;
  start[2] = d;
  q.push(start);
}

while (not q.empty()) {
  vector<int> u = q.front();
  q.pop();

  //Check destination
  if (map[u[0]][u[1]] == 'X') {
    cout << dist[u[0]][u[1]][u[2]] << endl;
    return;
  }

  //Keep on with the same direction
  int ni = u[0] + di[u[2]], nj = u[1] + dj[u[2]];
  if (ni >= 0 and nj >= 0 and ni < h and nj < w and map[ni][nj] != '#'
      and dist[ni][nj][u[2]] == -1) {
    vector<int> v(3);
    v[0] = ni;
    v[1] = nj;
    v[2] = u[2];
    dist[ni][nj][v[2]] = dist[u[0]][u[1]][u[2]] + 1;
    q.push(v);
  }

  //Turn to the right (directions are chosen so that turning
  //to the right is adding 1 to the current direction)
  int nd = (u[2] + 1) % 4;
  ni = u[0] + di[nd];
  nj = u[1] + dj[nd];
  if (ni >= 0 and nj >= 0 and ni < h and nj < w and map[ni][nj] != '#'
      and dist[ni][nj][nd] == -1) {
    vector<int> v(3);
    v[0] = ni;
    v[1] = nj;
    v[2] = nd;
    dist[ni][nj][nd] = dist[u[0]][u[1]][u[2]] + 1;
    q.push(v);
  }
}
```

13 Challenge 13 - Tuenti Timing Auth

Another of my favorites. You were given a web site with a textfield and a button. When using the right input, the button lead to another page telling you that your password was wrong.

Looking at the source code of the first page, you found a comment about some debug field. Adding this field in the browser made appear a comment in the second page, which told you how much time was spent checking your password.

Trying the 16 hexadecimal characters, I saw that there was one that took more time than the others (the correct one). So I made a *php* program that automatized this process: tried the 16 hexadecimal characters, chose the one that wasted more time and added it to the current password. It ended when arriving to the "success" page.

Code 14: Side channel attack

```
$result = file_get_contents($url, false, $context);

//Divide by lines
$result = explode("\n", $result);

//This happens when the whole password is correct
if ($result[139] == '</html>') {
  $best = 1000000;
  $ibest = $j;
}
else {
  //Divide this line by spaces
  $result = explode(" ", $result[27]);

  //Check the time spent by the server checking current password
  if ($result[2] > $best) {
    $best = $result[2];
    $ibest = $j;
  }
}
```

14 Challenge 14 - Train Empire

Given the description of a train system, you were asked what is the maximum number of points that could be achieved, under the restriction that each train had a limited amount of fuel.

My approach was a backtracking, storing the states already visited to avoid computing them again.

Code 15: Backtracking with memo

```
//Tries all possibilities with wagons in position 'pos_wagon',
//trains in position 'pos_train' and each train has a certain
//'fuel' left. Returns the maximum of points achieved
int backtracking(vector<int>& pos_wagon, vector<int>& pos_train,
                 vector<double>& fuel) {
  //If this state is already computed, return solution
  if (memo[pos_wagon][pos_train].count(fuel) != 0)
      return memo[pos_wagon][pos_train][fuel];

  int res = 0;
  //Compute current points
  for (int i = 0; i < s; ++i) if (pos_wagon[i] == dest[i]) res += value[i];

  //Lets try to move every train
  for (int k = 0; k < r; ++k) {
    //To every position
    for (int i = 0; i < s; ++i) {
      //If it is possible to go to that position
      if (dist[k][pos_train[k]][i] <= fuel[k] + eps) {
        int pos = pos_train[k];
        fuel[k] -= dist[k][pos][i];
        pos_train[k] = i;

        //Try the train going without carrying a wagon
        res = max(res, backtracking(pos_wagon, pos_train, fuel));

        for (int j = 0; j < s; ++j) {
          if (pos_wagon[j] == pos) {
            //For every wagon which is in position 'pos',
            //try to get it carried by the train to 'i'
            pos_wagon[j] = i;
            res = max(res, backtracking(pos_wagon, pos_train, fuel));

            //Take it back
            pos_wagon[j] = pos;
          }
        }

        //Let everything like it was before starting
        pos_train[k] = pos;
        fuel[k] += dist[k][pos][i];
      }
    }
  }

  return memo[pos_wagon][pos_train][fuel] = res;
}
```

15 Challenge 15 - Take a corner

This problem described you a game, and asked you to find a move that allowed one of the players to get a corner, whatever the other player's defense is.

I solved this problem with a min-max trying all possibilities (also a backtracking, but much prettier than the previous one), stopping when the attacking player finds a winning move, or when the defending player finds a losing (for the attacking player) move.

Code 16: Trying a move

```
//Check if this position is a valid move
vector<pair<int, int>> dirs;
for (int d = 0; d < 8; ++d) {
  int c = valid(i, j, d, who);
  if (c > 0) {
    //Found valid direction
    dirs.push_back(make_pair(d, c));

    //Flips enemy's tiles
    flip(i, j, d, c);
  }
}
//Puts its tile
board[i][j] = who;
bool winner;

if (dirs.size() > 0) {
  //There is a valid movement
  play = true;
  //Try possibilities after this move, and store if it is a winning move
  winner = backtracking(k + 1, 3 - who);
}
//Undo the movement
board[i][j] = 0;
for (int id = 0; id < int(dirs.size()); ++id) {
  flip(i, j, dirs[id].first, dirs[id].second);
}
//If there was a movement, check if there is need to try more
if (dirs.size() > 0) {
  //If it is winning move
  if (winner) {
    //And it is 1's turn, there is no need to try more. He wins.
    if (k % 2 == 0) {
      //If it is the first move, it is stored to print as solution
      if (k == 0) {
        ri = i;
        rj = j;
      }
      return true;
    }
  }
  //If it a losing move, and it is 2's turn, he wins.
  else if (k % 2 == 1) return false;
}
```

16 Challenge 16 - ÑAPA

In this problem, you were given a file with a list of circles (coordinates and radius). You also knew the constraints that the coordinates were in the interval $[0, 100000)$, and that the radius was in the interval $[1, 500]$.

I divided the plane in boxes with $500m$ side, that is, a 200×200 grid with 40000 boxes. Since the total number of circles is 3000000, supposing that they are uniformly distributed, the mean is that there are 75 circles per box.

Now, it's obvious that two circles cannot intersect if their boxes are at a distance greater than 2 (Manhattan distance). Thus, for each circle, I only have to try the circles in the 25 boxes at distance lower than or equal to two. That is a mean of 1875 circles. Doing this for each circle, I can get the result in an affordable computation time.

Code 17: Count intersections with a circle

```
//Count the number of circles that intersect with circle 'p'
int check(int p) {
  int count = 0;

  //There is only need to check circles in circle_box at distance
  //at most 2, because every box has side 500, and the maximum
  //radius is 500.

  //For all boxes at distance 2 or less
  for (int d = 0; d < 25; ++d) {
    int i = x[p] / 500 + di[d], j = y[p] / 500 + dj[d];

    //Check index out of bounds
    if (i >= 0 and j >= 0 and i < 200 and j < 200) {

      //Try every circle in that box
      int e = circle_box[i][j].size();
      for (int k = 0; k < e; ++k) {
        int q = circle_box[i][j][k];
        if (q == p) continue;
        if (intersection(p, q)) ++count;
      }
    }
  }

  //Result
  return count;
}
```

17 Challenge 17 - The Failsystem

In this problem, you were given a big file (over 5GB) in some "strange" file system. It was easy to discover that the file system was FAT32, but since I was told not mount the file, this followed with a lot of hours of research about how the FAT32 file system worked.

I won't explain its details here, but the general idea is that the files and directories are scattered in units called "clusters". There is also a table (File Allocation Table) indicating which cluster goes after the current one.

My criteria to tell if a file was corrupt was the following:

1. The file did not exist.
2. There was a *corrupt-file* mark in the File Allocation Table.
3. The cluster chain had a cycle and never reacher the *end-of-file* mark.

Also, the STD library didn't allow me to read to whole file, because it used 32 bits integers to store the current position and the file size was greater than 4GB. To solve this, I split the file in parts of 1GB at most.

18 Challenge 18 - Power Password

Another problem in the contest with no statement, only an IP direction. In this case, it was easy to find a compiled *python* program file, *index.pyc*. I decompiled this program with a tool called *decompyle2* and got the source code.

Code 18: Part of index.py

```
try:
    h, p = q.split(':')
except ValueError:
    error('Password missing')

try:
    i = json.load(open('../keys.json'))[h]
except:
    error('Invalid input')

print 'Right!' if p.isdigit() and len(p) < 15 and pow(i[0],int(p),i[1])==i[2]
      else 'Wrong!\n' + '\n'.join(map(str, i))
```

So, the true problem was to find the discrete logarithm of $i[2]$ with base $i[0]$ and modulo $i[1]$. Since it is guaranteed that the solution has length less than 15, it can be computed in $\mathcal{O}(10^7)$ with baby-step giant-step algorithm.

Code 19: Baby-step giant-step algorithm

```
#store m powers of i0 in a set
t = 1
for i in range(m):
    s.add(t)
    t = (t * i0) % i1


#compute powers of i0^(-m) multiplied by i2 and look for them in the set
sol = 0
t = i2
for j in range(m):
    if t in s:
        #A matching is found, so exists 'i' such that i2 * (i0^(-m))^j = i0^i
        sol = j
        break
    t = (t * i0_m) % i1

#look for that 'i'
goal = (i2 * pow(i0_m, sol, i1)) % i1
t = 1
for i in range(m):
    if t == goal:
        #A solution is found: i2 = i0^(j * m + i)
        #Print solution
        print(j * m + i)
        break
    t = (t * i0) % i1
```

19 Challenge 19 - Password recovery

This problem gives as input a *C* source code encoded in base64. The program contains an automaton that operates with the given password (64 bits) and tells you if it is the correct password.

After writing a readable program which does the same than this one, I saw that the program stored the sum of the eight bytes of the password and then performed fifty times a series of invertible operations. Then checked some final values for each of the bytes and, if all of them were correct, output "Right!", or "Wrong!" if any of them was wrong.

I wrote a function that performed fifty times the inverses of those operations starting with the final values, and hence ending up with the correct password: "4834636b7c643165".

Code 20: Reverse engineering

```
//Gets the password
void crack() {
  //Final values
  r[0] = 219;
  r[1] = 18;
  r[2] = 181;
  r[3] = 20;
  r[4] = 35;
  r[5] = 244;
  r[6] = 56;
  r[7] = 104;
  for (int i = 0; i < 50; ++i) {
    //Inverse operations
    swap(r[0], r[7]);
    swap(r[1], r[6]);
    swap(r[2], r[5]);
    swap(r[3], r[4]);
    if (r[4] > 128) r[1] -= 35;
    r[2] -= r[0];
    r[3] = ~r[3];
    r[0] ^= (((r[3] + r[7]) ^ r[1]) + r[4]);
    r[6] = -r[6];
    r[4] *= 197; //Inverse of 13 modulo 256
    r[7] = (((r[7] >> 4) & 15) | ((r[7] & 15) << 4));
    r[5] ^= 72;
  }
  //Print initial values: password
  for (int i = 0; i < 8; ++i)
    cout << hex << setw(2) << setfill('0') << int(r[i]);
  cout << endl;
}
```

20 Challenge 20 - forfor

In challenge 20, you were given a function and were asked to print the result of that function.

Code 21: forfor function

```
//Given function
long long calc(int n) {
  long long ret = 0,a,b,c,d,e,f,g,M = 3211123;

  for(a = 1; a <= n; a++)
    for(b = 1; b <= n; b++)
      for(c = 1; c <= n; c++)
        for(d = 1; d <= n; d++)
          for(e=1; e <= n; e++)
            for(f=1; f <= n; f++)
              for(g=1; g <= n; g++)
                if(a + b + c + d + e + f + g == n)
                  ret = (ret + a*a + b*b + c*c + d*d + e*e + f*f + g*g) % M;
  return ret;
}
```

I supposed that this function was equivalent to some polynomial, so I printed the first 20 values (without the modulo) and interpolated them with *WolframAlpha*. Effectively, it was the following polynomial:

$$\frac{1}{2880}x^8 - \frac{1}{120}x^7 + \frac{119}{1440}x^6 - \frac{7}{16}x^5 + \frac{3829}{2880}x^4 - \frac{554}{240}x^3 + \frac{167}{80}x^2 - \frac{3}{4}x$$

So I wrote a function that computed the result of evaluating this polynomial modulo M, multiplying by the inverses of the denominators module M instead of dividing, to get the coefficients.

Code 22: Evaluating the polynomial

```
//Computes the result of evaluation the polynomial in x = n
long long Calc(int n) {
  long long r = 0, N = 1;
  for (int i = 0; i < 9; ++i) {
    r = (r + coef[i] * N) % m;
    N = (N * n) % m;
  }
  return r;
}
```