Competitive programming

First session: the 5 easiest spoj problems

Prof. Rhadamés Carmona

Content

- What is competitive programming
- How to start?
- Training
- What to do in a real contest?
- First problem in spoj.com
- Problems for next class

What is competitive programming

- Mind Sport
- Main goal: solve problems in a short amount of time
- Two types: short term (1-5 hours) and long-term (days, months)
- We focus on short-term programming contests
- Most famous: International Collegiate Programming Contest (ICPC)
- Teams of 3 students
- Local contest, regional contest, world final contest

How to start?

- We have to select the preferred language
- I suggest to use c++
  - STL
  - The speed
  - Short codes
  - We can use macros to make the code even shorter
  - We can use some low level tricks (C part of C++) to make the code faster
- It is possible to use Java or Python. BigInteger of Java is very helpful. Java and Python are slower (T.L.E. = time limit exceeded…)

How to start?

- Select an online judge for training
  - http://Spoj.com (I will use it for training)
  - http://Codeforces.com
  - http://Codechef.com
  - https://uva.onlinejudge.org
  - http://topcoder.com
  - http://coderbyte.com
  - https://a2oj.com/ (I will use it for contests)
- I recommend to start with spoj.com
  - Create an account
  - Solve your first problem today ☺

How to start?

- Interesting list of algorithm and data structures we need to know

https://www.geeksforgeeks.org/top-algorithms-and-data-structures-for-competitive-programming/

- Classify the set of problems
  - Mathematics : Prime Number, Big Integer, Permutation, Factorial, Fibonacci…
  - Dynamic Programming : Longest Common Subsequence (LCS), Longest Increasing Subsequence, 0/1 Knapsack, Coin Change, Matrix Chain Multiplication, Max Interval Sum, …
  - Graph Traversal: Flood Fill, Floyd Warshal, MST, Max Bipartite Matching, Network Flow, Articulation Point, …
  - …

Training

- On every session
    - We will select a set of problems (from spoj) to be solved during the week
    - I will prepare slides for specific themes (e.g. dynamic programming, graphs, Fenwick tree, disjoint sets, convex hull…)
    - For each selected problem in the previous week, we discuss the solution you found; you can prepare some slides if necessary
    - You may solve some other related problems if you have free time. The more problems solved, the better!
    - Spoj.com keeps track of all your progress. You can show me your progress every week
- After 2 months, we may join into a live programming contest online (codeforce) every 2 or 3 weeks

What to do in a real contest?

- Read all the problems, rank the problems: easy to tough
- Start with the easiest one. Sketch the algorithms, complexity, conditions, data structures and tricky details. Try to find the easiest and fastest solution. Avoid brute force algorithms; take into account the maximum problem size. Do not code if you already know that it will give TLE
- Code it, as fast as possible, and it must be correct
- It is good to code in pairs… so, if the coder makes a mistake, a second “head” can help. Also, a second head can create test cases
- Try to break the algorithm – look out for boundary test cases
- Once you are sure that your solution is good, submit it

Sample macros in C/C++

```
/* input, output routines */
#define RI(X) scanf("%d", &(X))
#define RII(X, Y) scanf("%d%d", &(X), &(Y))
#define RIII(X, Y, Z) scanf("%d%d%d", &(X), &(Y), &(Z))
#define DRI(X) int (X); scanf("%d", &X)
#define DRII(X, Y) int X, Y; scanf("%d%d", &X, &Y)
#define DRIII(X, Y, Z) int X, Y, Z; scanf("%d%d%d", &X, &Y, &Z)
#define debv(x,n)     REP(i,n) printf("%d ",(x)[i] );  nl  //output int list

/* loops */
#define FOR(i,a,b) for (int (i)=(a); (i) < (b); (i)++)
#define REP(i,n) FOR(i,0,n)

/* strings */
#define rwc(c)  scanf("%c",&c)!=EOF && c!=10 && c!=32
#define rlc(c)  scanf("%c",&c)!=EOF && c!=10                    //read line by char
#define rac(c)  scanf("%c",&c)!=EOF                             //read all char by char
#define sz(x) (int((x).size()))
```

Sample macros in C/C++

```
 /* data types */
#define ull unsigned long long
#define ll long long

/* data structures */
#define pi pair <int,int>
typedef vector <int> vi;
typedef vector <vi> vvi;

/* Array */
#define initn(a,n,x) REP(i,n) a[i]=x
#define init(a,x) REP(i,sizeof(a)/4) a[i]=x;
#define ra(x) REP(i,sizeof(x)/4) scanf("%d",&x[i])
#define showa(x) REP(i,sizeof(x)/4) printf("%d ",x[i])
#define showan(x,n) REP(i,n) printf("%d ",x[i])
#define sa(x) do {\
                                        showa(x);\
                                        printf("\n");\
                                } while (0)
```

Sample macros in C/C++

```
 /* others */
#define up(v,x) (upper_bound((v).begin(), (v).end(),(x))-(v).begin())
#define lb(v,x) (lower_bound((v).begin(), (v).end(),(x))-(v).begin())
#define bs(v,x) binary_search((v).begin(), (v).end(),x)

#define flag printf("FLAG\n");
#define nl printf("\n");
#define pb push_back

/* constants */
#define inf (int)1e+9
#define VIS 1
#define UNVIS 0
#define null NULL
#define MAX (int)1e+6
#define MIN -(int)1e+6
```

First problem in spoj.com

- Problem #1: Life, the universe and everything (#basic #tutorial #ad-hoc-1)

Your program is to use the brute-force approach in order to find the Answer to Life, the Universe, and Everything. More precisely... rewrite small numbers from input to output. Stop processing input after reading in the number 42. All numbers at input are integers of one or two digits.

First problem in spoj.com

- Problem #1: Life, the universe and everything (#basic #tutorial #ad-hoc-1)

Example

```
Input:      Output:
1           1
2           2
88          88
42
99
```

First problem in spoj.com

```
#include <iostream>
using namespace std;
int main()
{
        int v;
        while (cin >> v)
        {
                if (v == 42) break;
                cout << v << endl;
        }
        return 0;
}
```

Problems to solve

42, 2, 24, 11

Try first yourself…

Problems for next meeting

- Problem #42: Adding reversed numbers (#simple-math #ad-hoc-1)
  - Let's start with this simple problem
  - Try to make it works with your first try
  - Just make a function to reverse any integer positive number
  - Solution is O(#digits(N)), it is $O(\log_{10}N)$

Problems for next meeting

- Problem #2: prime generator (#number-theory)
  - do not use brute force (TLE)
  - try Sieve of Eratosthenes (TLE and memory overflow)
  - precompute all primes, and use a big look-up table (more than 54 millions of primes… source code size overflow)
  - 10^9 can be represented in a single 32 bits integer
  - the dividers of x in 1 and 10^9 are the prime numbers in the range $1..\sqrt{10^9} = 1..31623$.
  - The dividers of x are all the primes between 1 and $\sqrt{x}$
  - Precompute all primes in 1.. 31623 before processing any range of numbers.
  - This solution takes about 1 second. There are other solutions which takes <0.1s
  - Boundary case: number 1 is not prime! I failed 2 times because of that!
  - May you obtain the complexity of your solution?

Problems for next meeting

- Problem #24: Small Factorials (#math #big-numbers)
  - Oh, Java has the BigInteger class. And it can compute the factorial for 100 easily. However, in c++ we have to do it by ourselves
  - To compute factorial, we need to multiply numbers every time. We cannot use long long, long double … they are not enough to represent 100!
  - Since there are only 100 possibilities, what about if we pre-compute all these factorials? Then, during the queries, we just access an array with the string containing the required factorial?. We have enough memory to store 100 strings. So, do not worry (by now) with the speed of your algorithm. Just get the values of the first 100 factorial numbers by brute force, and save them into an array. Then, you can " wire" these values into the code
  - The good news… during the factorial calculus, we are multiplying big numbers by small numbers…. So, we may simulate the product by a sequence of additions without impacting too much the speed (no TLE). Thus, we may not need to "wire" precomputed values. We can compute all 100 factorials before reading the input queries
  - Please, obtain the complexity of your solution

Problems for next meeting

- Problem #11: FCTRL – Factorial (#math)
  - We can print the numbers of problem #24
  - We can reuse some previous code from now ☺ for analysis
  - You can see the pattern.
  - You can do it!
  - Solution is O(log$_5$N)

```
  5   0
 10   00
 15   000
 20   0000
 25   000000
 30   0000000
 40   000000000                              9
 45   0000000000                            10
 50   000000000000                          12
 55   0000000000000                         13
 60   00000000000000                        14
 65   000000000000000
 70   0000000000000000
 75   000000000000000000
 90   000000000000000000000                 21
 94   000000000000000000000                 21
 95   0000000000000000000000                22
 99   0000000000000000000000                22
100   000000000000000000000000              24
124   0000000000000000000000000000          28
125   0000000000000000000000000000000       31
```

Problems for next meeting

Congratulations, you have completed your first 5 problems. Categories:

#number-theory #math #big-numbers #ad-hoc-1