Lab 1: Getting Started with Contests

Algorithmic Programming Contests 2023-2024

Wietze Koops

September 2, 2023

Although the course is called 'Algorithmic Programming Contests', for many problems in programming contests you do not need any algorithms at all, or only basic built-in algorithms such as sorting algorithms. In this first lab we discuss some problem-solving strategies and some points to pay attention to when implementing solutions.

1 Problem Solving Techniques

We start with a brief list of problem solving techniques. Many of these will be covered in much more detail later in the course; this list serves as an overview and as a reminder for those who have seen this in earlier courses.

- Math: combinatorics (e.g. factorial, binomial coefficients), number theory (e.g. primes, modular arithmetic, divisibility properties), calculus (mainly for optimizing functions), sum formulas for arithmetic/geometric series, graph theory (e.g. handshaking lemma, Euler's formula for planar graphs), probability theory
- Data structures: array, queue, stack, set, map/dictionary, priority queue
- Techniques: brute force, sorting, greedy algorithms, dynamic programming, binary search, divide & conquer, recursion, randomized algorithms, two pointers
- Algorithms: Euclid's algorithm for computing GCDs, graph algorithms (BFS, DFS, Floodfill, Dijkstra's, connected components, topological sort, minimum spanning tree), geometric algorithms, string algorithms

In this lab, we focus on the following problem types: mathematical problems, brute force, sorting, and implementation. In an implementation problem, the problem can be solved by carefully implementing the procedure explained in the problem statement.

2 Problem Solving Strategies

In this section we discuss some problem solving strategies. Note that you should also learn to think about problems with only pen and paper, i.e. without a computer, since at ICPC-style contests you only have one computer per team.

Check the limits to guess the complexity of the algorithm. In problems it is always indicated how large the maximal input n can be. This gives an indication of how efficient an algorithm has to be to solve the problem within the time limit. As a rough estimate, you can do between 10^7 and 10^8 operations per second. This means the following:

Upper bound	Required complexity
$n \leq 10^{18}$	$O(\log n)$
$n \leq 10^{12}$	$O(\sqrt{n})$
$n \leq 10^6$	$O(n)$
$n \leq 10^5$	$O(n \log n)$
$n \leq 2{,}000$	$O(n^2)$
$n \leq 200$	$O(n^3)$
$n \leq 20$	$O(2^n)$
$n \leq 10$	$O(n!)$

Note that the easiest problems, sometimes an $O(n^2)$ algorithm is accepted even though an $O(n)$ algorithm is possible (and sometimes even more natural).

Work out small cases. Finding a solution for small instances by hand can be very enlightening. It often helps to find a pattern.

- Try to do the small cases in a systematic way. If multiple solutions are possible, try to find all of them so that you do have the one that is part of a possible pattern.
- In case that there are multiple possible solutions, keep in mind that the solutions to the sample test cases are typically chosen in a way to not follow a possible pattern.

Keep considering both mathematical and algorithmic approaches. One challenge in contests is to identify the type of a problem. Some problems that may look like mathematical problems actually aren't mathematical problems, but need to be solved by e.g. dynamic programming or binary search. In such cases, trying to find a direct formula is doomed to fail.

The converse can also happen, e.g. every algorithmic approach seems to slow because you actually need a formula to compute the answer. (In such a case, it can be helpful to use the slow solution to compute the answer in many cases to help finding the pattern.)

3 Implementation Issues

Output modulo $10^9 + 7$. In some problems it is requested to output the answer modulo $10^9 + 7$ (or some other large prime). For such problems, keep in mind that $(a + b)\%m = (a\%m + b\%m)\%m$ and $(a * b)\%m = (a\%m * b\%m)\%m$; this means that you can do the modulo in all intermediate calculations (as long as they only involve addition, subtraction and multiplication). This avoids large numbers and thereby speeds up the implementation.

Be careful with floating point numbers. Especially in mathematical problems, it is easy to forget that floating point numbers are not real numbers in the mathematical sense: they are not exact. In case working with floating point numbers is unavoidable, there should be a statement in the Output section of the problem such as 'the answer should be correct up to an absolute or relative error of 10^{-6}'.

- If you want to compute $\lceil n/k \rceil$, do not use `math.ceil(n/k)` (in Python). This first computes the floating point number `n/k`, which may result in loss of precision and gives the wrong answer for e.g. $n = 10^{18}$, $k = 10^{17} - 1$.

  Instead, use `(n+k-1)//k` to compute $\lceil n/k \rceil$ using integer arithmetic.

- If a, b, c and d are positive integers, then to test whether $a/b \geq c/d$, do not use the test `a/b >= c/d`, but instead use `a*d >= b*c`.

- Do not create a set whose elements are floating point numbers.

- If it is given that floats in the input have at most (or exactly) r digits after the decimal point, then 10^r times the floats are ints. Use `int(10**r * x + 0.5)` to do the conversion. The `+0.5` is necessary, otherwise a number whose floating point representation ends in .99999... may be rounded downwards incorrectly.

- An alternative is testing `x == y` using `abs(x-y) < eps` where `eps` is a small number, e.g. $10^{-(r+1)}$ if input numbers have at most r digits after the decimal point.

- As an example, consider the following problem:

  Read in two floating point numbers a and b with at most three digits after the decimal point. Output the sum $a + b$. If $a + b = 1$, output 'One!' instead.

  Just testing `a+b == 1` will give Wrong Answer because it misses some cases where $a+b$ is .99999999... or 1.000000001... in floating point representation. So one of the two previous techniques should be used.

- If the problem states that 'changing inputs by up to 10^{-6} does not change the answer', you do not have to be careful. If the problem states that 'increasing inputs by up to 10^{-6} does not change the answer', then add e.g. 10^{-7} to all inputs. This makes sure that small rounding errors do not yield numbers that are smaller than the actual input values.

4 Reading from the book

A very good book on competitive programming, containing much more in-depth explanations on these topics, is the Competitive Programmer's Handbook by Antti Laaksonen. The sections corresponding to this lab are:

- Ch.1 (Introduction): Section 1.3 (Working with numbers) and Section 1.5 (Mathematics).
- Ch.2 (Time complexity): Entire chapter. For the problem in Section 2.4, it might be nice to try to find the $O(n^2)$ solution yourself before reading it.
- Ch.3 (Sorting): Section 3.1 (Sorting theory) and Section 3.3 (Binary search).

5 Lab Problems

There are two types of problems in this lab: mathematical problems and implementation problems. Mathematical problems are problems that can be solved using only a few lines of code, but you need to find the right formula to do this. With implementation problems, the goal is to carefully implement a procedure described in the problem.
All problems in the lab can be found on Kattis.

- (BAPC preliminaries 2019) Greetings! • Hint: 27
- (North American Regionals 2022) Overdraft • Hint: 33
- (North American Regionals 2022) Scaling Recipe • Hint: 23
- (North American Regionals 2022) Double Password • Hints: 26 · 49
- (North American Regionals 2022) Code Guessing • Hints: 25 · 5 · 42 · 44
- (BAPC preliminaries 2019) Architecture • Hints: 8 · 40 · 20 · 6
- (North American Regionals 2022) Stream Lag • Hints: 1 · 31 · 19
- (BAPC preliminaries 2019) Exits in Excess • Hints: 29 · 35 · 39
- (BAPC preliminaries 2019) Floor Plan • Hints: 12 · 46 · 2 · 13 · 37 · 24
- (BAPC preliminaries 2017) Disastrous Doubling • Hints: 3 · 28 · 22 · 38 · 45
- (BAPC 2021) Implementation Irregularities • Hints: 7 · 4 · 14 · 18 · 43
- (NWERC 2019) Expeditious Cubing • Hints: 36 · 30 · 11 · 16 · 17
- (NWERC 2019) Inverted Deck • Hints: 48 · 10 · 34 · 15 · 41 · 32
- (BAPC 2021) Lopsided Lineup • Hints: 47 · 21 · 50 · 9

6 Assignment

6.1 Assignment instruction

For each lab there will an assignment. Each assignment will be assessed on a pass/fail basis, and you need to pass at least 70% of the assignments to meet the 'active participation in course assignments' requirement. Note that it is not necessary to solve all the problems to pass the assignment. In particular, each assignment will include at least one more challenging problem, as this is the most effective way to train for the hard problems found in contests. However, if you did not solve a problem, it has to be clear that you put significant effort in it.

The lab assignment is due two weeks after the lab session. If you cannot meet this deadline due to exceptional circumstances, send an email before the deadline to `apc@rug.nl`.

Deliverable, to be sent to `apc@rug.nl` before the deadline:

- For each problem that you solved, your solution and a one or two sentence summary of your solution. In addition, you can send in any failed attempts if you are unsure why they failed to receive feedback on them.
- For each problem that you did not solve, your failed attempts. If you already know/suspect the reason why it doesn't work, also mention this.

  If you didn't write any code (because you had no idea where to start), write a few paragraphs on what you tried; in particular, did you figure out why the answer to the Sample Input is what it is? Did you think about the required complexity? What techniques/data structures/algorithms did you consider?

You may discuss solution strategies with other students. However, the solution that you submit via email (Deliverable) must be written completely by yourself.

6.2 Assignment problems

The assignment problems corresponding to this lab are:

- (CSES problem set) Missing Number
- (CSES problem set) Increasing Array
- (CSES problem set) Apartments

7 Hints

1. Suppose that packet i arrives at the beginning of second t_i, and can be played directly at that point. Give a formula for the lag at that point in time.
2. Suppose that n is even. Then either $m-k$ or $m+k$ is even, so both must be even (why?). What does this mean for n?
3. As a first attempt, just simulate the process, i.e. compute the number of bacteria in each step, check whether this is non-negative in each step, and if so, do the modulo 10^9+7 only in the final step. This gives WA in C++ and TLE in Python. Why?
4. Sort the solved problems by submission time. However, we cannot just sort the submission times, because we need to keep track of the index of the problem (or of the computer time required for the problem) as well. How can you do this?
5. A brute-force solution works, since there only a few possibilities. For each pair (c, d) of Bob's digits, determine whether the given string is consistent with the pair (c, d).
   Alternatively, for a mathematical solution: do a case distinction on the position of Alice's cards, and figure out (with pen and paper) the conditions on Alice's digits $p < q$ for the solution to be unique.
6. Make a grid where the two skylines intersect in the building with maximum height. What can you choose for the remaining building heights?
7. Each time a problem is solved, this creates a new possible bottleneck for the number of computers required. Hence, we want to look at the computer time required for solving the ℓ earliest submitted problems. To compute this, we first need to know what the ℓ earliest submitted problems are.
8. Start by figuring out why the second Sample Input is impossible. To do this, just draw a few 4×4 squares and try to fill them with numbers so that you get the required skylines. Which problem do you run into?
9. The difference in strengths is also equal to $\sum_{i \in S} \sum_{j=1}^{n} c_{i,j} - \sum_{i \in T} \sum_{j=1}^{n} c_{i,j}$. Hence, we want to maximize the difference between the rowsums corresponding to S and the rowsums corresponding to T.
10. Assume for simplicity that all numbers are different. At which point do we need to swap?
11. Suppose that in your final solve you get a time of 0. What is your final score? Hence, when is it impossible? Now suppose that you are exactly at the boundary. Do you actually need a score of 0?
12. Start by doing small cases. Why is $n = 10$ (second Sample Input) impossible? Are there other possible answers for Sample Input 3? How about the smallest cases: $n = 1, 2, 3, \ldots$?
13. Suppose that n is even. Then either $m - k$ or $m + k$ is even, so both must be even since their difference is $2k$ which is even. Hence, $n = m^2 - k^2 = (m-k)(m+k)$ is the product of two even numbers and hence divisible by 4. Hence, if n is even, but not divisible by 4, then it is impossible. It turns out that all other n are possible; find constructions using pen and paper (do small cases!).

14. Sort pairs (s_j, j) to sort problems by submission time while keeping track of the index.

15. Assume for simplicity that all numbers are different. If j is the smallest $j > i$ such that $v_j < v_{j+1}$, then card j is the last card of the part that needs to be swapped.

16. If the answer is not impossible nor infinite, the last time will count towards the score. Hence, the minimum and maximum of the current four times are the overall minimum and maximum and will be discarded. Write down an equation for the final score based on this, and solve it.

17. Implementation: to deal with rounding errors, either convert everything to integers using that the times have exactly two decimal places, or use ε-comparisons when comparing the best and worst three of the four to the target score.

18. Write down a formula for the number of computers required to submit the earliest submitted problem, i.e. if the first problem is submitted at time s_{p_1} and requires t_{p_1} computer time, write down a formula in s_{p_1} and t_{p_1} expressing the number of computers required.

19. Make sure that you always output a non-negative lag.

20. The highest building in the grid will show up in both skylines. Therefore, the maximum number in both skylines must be equal. It turns out that this is a sufficient condition, i.e. if the maximum is equal in both skylines, then it is possible. Why?

21. If S and T are subsets of $\{1, \ldots, n\}$ representing the teams, the difference in strengths is equal to $\sum_{i \in S} \sum_{j \in S} c_{i,j} - \sum_{i \in T} \sum_{j \in T} c_{i,j}$. It is given that the matrix is symmetric. This is essential to solve the problem; you can use the symmetry to find another way to express this difference.

22. When applying the modulo 10^9+7 at every intermediate step, one might mistakenly think that there are not enough bacteria while in fact there are enough bacteria. For example, if there are 10^9+8 bacteria, we have enough to use 42 of them for an experiment. However, when taking the modulo $10^9 + 7$, we get that there is only one bacterium which is not sufficient. Hence, we should only take the modulo for some point onwards. How many bacteria are needed to ensure that there are enough bacteria for all future experiments?

23. Mathematically, this problem is not very complicated: If a units of an ingredient are needed for x portions, then $\frac{ay}{x}$ units are needed for y portions. The tricky part of this problem is that with when working with floats, $a \cdot (y/x)$ is not guaranteed to be equal to $(ay)/x$. Only one of these two expressions can be computed without using floats. The other will give wrong answer due to rounding errors.

24. Suppose that n is divisible by 4. Then we can take $m = k + 2$.

25. Try to solve some testcases by hand. In particular, figure out why the answers to the Sample Inputs are what they are, but you can also write down some possible inputs yourself and try to figure out the answer with pen and paper.

26. A brute-force solution works, as there are only 10,000 possible passwords.
    Alternatively, for a mathematical solution: for each digit of the password there are 1 or 2 options. Since these options be made independently at each position, the total number of options is the product of the number of options for each digit.

27. In Python, one can make a string s consisting of n copies of character c using `s = n*"c"`. In C++, this can be done using the string constructor `std::string s(n, "c");`

28. When only applying the modulo $10^9 + 7$ at the end, we get a WA verdict in C++ due to overflow, and a TLE verdict in Python due to computations with huge numbers taking too long. Hence, we should apply the modulo 10^9+7 also at intermediate steps. However, if we always do this, we will get WA. Why?

29. Formally, the question asks to remove at most half of the arcs from a directed graph to make it acyclic. You can use that a directed graph is acyclic if and only if there exists some ordering of the nodes $v_1, v_2, \ldots, v_n$ (called a *topological sorting*) such that the following holds: if there is an arc from v_i to v_j, then $i < j$.

30. Suppose that in your final solve you get a time of 'infinite'. What is your final score?

31. If packet i arrives at the start of second t_i, the lag is at least $t_i - i$ (it can be larger, if it cannot be played when it arrives). Note that lagged time can never be recovered.

32. A tricky thing is to take ties into account. In principle, ties are fine in a non-decreasing sequence, so we do not need to reverse at that point already. However, it could be that there are ties directly before (i.e. with the same value as) v_i, where i is the smallest i such that $v_i > v_{i+1}$. Where do you need to start reversing then?

33. Suppose that the starting balance is 0. How can you compute the smallest (i.e. most negative) balance reached in the process? How does this relate to the answer?

34. Assume for simplicity that all numbers are different. If i is the smallest i such that $v_i > v_{i+1}$, then card $i + 1$ is the first card of the part that needs to be swapped.

35. You can make sure that either $1, 2, \ldots, n$ or $n, n-1, \ldots, 1$ is a topological sorting.

36. Write down some values for the first four times and figure out for which target time it is impossible, for which target time the answer is infinite and what happens in between.

37. Suppose that n is odd. Then we can take $m = k + 1$.

38. If at some point there are more than 2^{61} bacteria before taking bacteria away, then after taking away $b_i \leq 2^{60}$ bacteria there will be more than 2^{60} bacteria, which will double to again more than 2^{61}. So then the number of bacteria remains more than 2^{61}. Hence, if there are more than 2^{61} bacteria we can start taking the modulo, and stop checking whether there are enough bacteria.

39. Write $\mathcal{A} = \mathcal{A}_1 \cup \mathcal{A}_2 = \{(u, v) \in \mathcal{A} : u < v\} \cup \{(u, v) \in \mathcal{A} : u > v\}$, where $\mathcal{A}$ is the set of arcs (corridors). When deleting all arcs in $\mathcal{A}_1$, then $n, n-1, \ldots, 1$ is a topological sorting (why?). Similarly, when deleting all arcs in $\mathcal{A}_2$, then $1, 2, \ldots, n$ is a topological sorting.

40. Consider the highest building in the grid.

41. We can check whether a sequence is sorted after reversing by reversing the elements and checking whether the resulting sequence is non-decreasing. In Python, reversing index i (including) up to j (excluding) can be done using `a[i:j] = reversed(a[i:j])`. In C++, you can use `std::reverse(std::begin(a)+i, std::begin(a)+j)`, assuming you used a vector to store the sequence.

42. Brute-force: for each pair (c, d) with $c < d$ and c and d not equal to Alice's digits p, q: make an array (or set) of Alice's digits $p < q$. Then make an array with all four digits and sort it. Check whether the ith number in the sorted array matches the `A` or `B` in the given string by checking whether the ith number is (or is not) among Alice's digits.

43. If the first problem is submitted at time s_{p_1} and requires t_{p_1} computer time, then $\left\lceil \frac{t_{p_1}}{s_{p_1}} \right\rceil$ computers are required. Now write a formula for number of computers required for the two earliest submitted problems, etc. In the end, try to generalize this to a formula for the number of computers required for the ℓ earliest submitted problems.

44. Mathematical solution: If one of Bob's digits is the smallest, then there are usually multiple options for it, unless Alice's smallest digit has a specific value. Similar considerations apply if Bob's digit is the largest, or between two of Alice's digits.

45. Note that in C++ you need to use `long long`s for this problem.

46. A difference between squares factorizes: $n = m^2 - k^2 = (m-k)(m+k)$. This factorization is useful to study the number theoretic properties of numbers of the form $m^2 - k^2$.

47. Suppose that you take the first $n/2$ players for the first team, and the last $n/2$ players for the second team. How can the difference in strengths be expressed? Try to find another way to express this difference.

48. Think about the complexity: $n \leq 10^6$, so your algorithm needs to run in time $O(n)$. This means that you can make only a few passes over the array.

49. To extract the kth least significant digit of a number n, divide by 10^{k-1} and take the result modulo 10. In Python, this is simply `(n//10**(k-1))%10`.
In C++, there is no built-in for raising an integer to an integer power; the easiest way is to keep a variable containing 10^{k-1} while looping over k.

50. We have $\sum_{i \in S} \sum_{j \in T} c_{i,j} = \sum_{i \in S} \sum_{j \in T} c_{j,i} = \sum_{j \in T} \sum_{i \in S} c_{j,i} = \sum_{i \in T} \sum_{j \in S} c_{i,j}$ since C is symmetric. Use this to find another way to express the difference in strengths.