Algorithms for Programming Contests - Week 01

Prof. Dr. Javier Esparza,
Vincent Fischer,
Jakob Schulz,
conpra@model.cit.tum.de

14. Oktober 2025

General Concept

- Theoretical background about several concepts and algorithms.
- Implementing algorithms is a useful skill, for CS research, other research, software development... sometimes for hobbies.
- Contest-format algorithm implementation practical course.

The Problem Sets

- Available on `judge.db.cit.tum.de`
- Usually, $3-5$ problems per week, and at least one problem of every difficulty level (easy, medium, hard).
- Each problem gives up to $N+N$ points, where usually $N \in \{4, 6, 8\}$ (more on that later).
- Each week's problems are (mostly) about the topics, algorithms and concepts from that week's lecture.
- There will be hints in the lectures.

Have a Problem with a Problem?

Should difficulties arise: Ask questions! If needed, we can schedule a meeting in our office.

Grading

Your final grade will be determined by how many points you earned, as well as an oral discussion of the topics learned at the end of the semester. The oral part will account for 25% of the grade (using rounding to the nearest). Only one final grade goes on record.
The (tentative) key for grading the problems is the following:

Percentage	Grade
$\geq 90\%$	1.0
$\geq 85\%$	1.3
$\geq 80\%$	1.7
$\geq 75\%$	2.0
$\geq 70\%$	2.3
$\geq 65\%$	2.7
$\geq 60\%$	3.0
$\geq 55\%$	3.3
$\geq 50\%$	3.7
$\geq 40\%$	4.0

Topics (preliminary list)

1. Introduction
2. Data Structures (UF, Binary Search, Graphs)
3. Graphs, Minimum Spanning Trees, DFS, BFS
4. Shortest Paths
5. Maximum Flows
6. Brute Force / Backtracking
7. Greedy
8. Dynamic Programming
9. Number Theory
10. Trees
11. Geometry
12. Contest
13. Conclusion

Judging system

`https://judge.db.cit.tum.de`

- Please register!
- We need to give you access to our course, which is invisible at first.
- The AACPP courses are *not* from us!

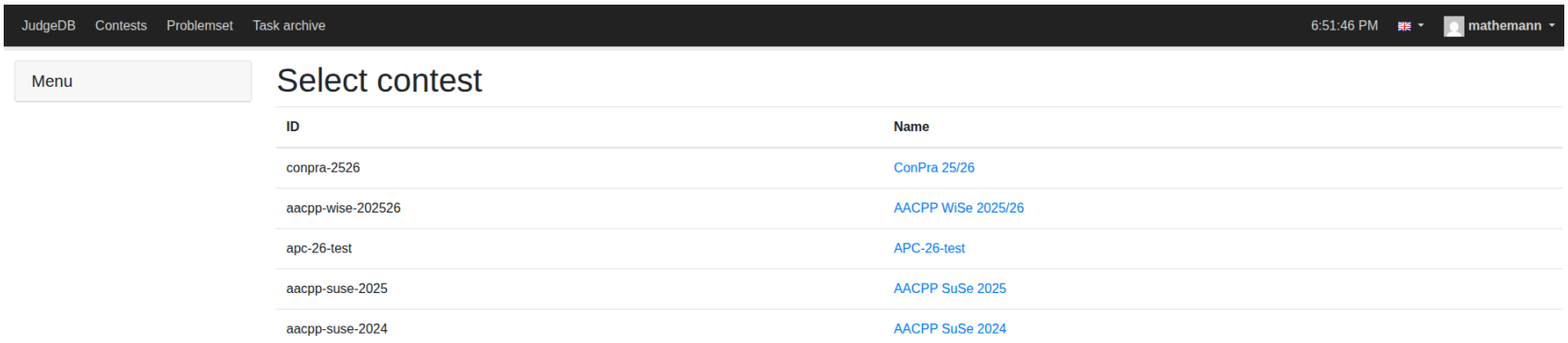
Registration Details

`https://judge.db.cit.tum.de`

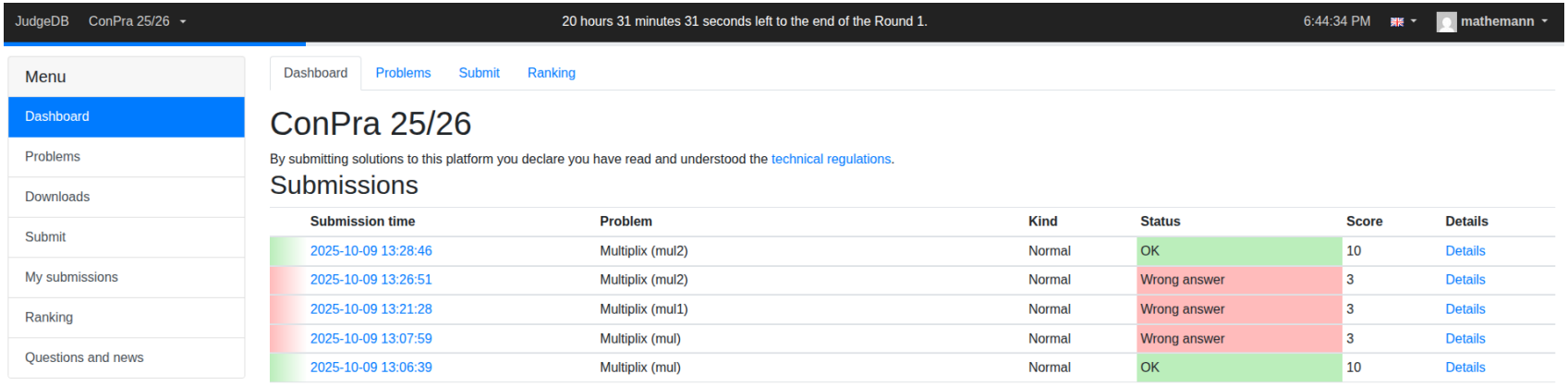
- We will have a scoreboard in our course, where you will be listed with the names you enter in the registration form. You are free to use pseudonyms. → the names will appear on our course-scoreboard
- **Don't forget your password! The reset links do not work!**
- Send an email to `conpra@model.cit.tum.de` containing your username and your *real* name.
- Afterwards, we will give you access to our ConPra course :)

Judging system

Once you have access to the ConPra course, you can find it under "Contests" as "conpra-2526" / "ConPra 25/26".



Dashboard



Read through the technical regulations once!

Problem structure

A problem consists of several parts:

- name, abbreviation, difficulty,
- problem author,
- problem statement,
- input format specification,
- output format specification,
- constraints,
- sample input and output.

SS15N01A Hello World!

Author: Stefan Toman

This is probably the first problem you will solve and it should help you set up and test your system. Solve this problem first to make sure everything is in place.

We would like to introduce you to Lea. You will meet her in many of the problems you will solve. After reading all of them you will know her quite well.

Lea is a very friendly person who likes to say hello to everybody, but she doesn't want to say the same thing to every person she meets. Therefore, she never knows what to say. For greeting Bob it is appropriate to say "Hello Bob!", whereas for greeting Peter it is better to say "Hello Peter!". Help her and tell her which sentence to use.

Input

The first line of the input contains an integer t. t test cases follow.

Each test case consists of a single line containing a name *name*.

Output

For each test case, print a line containing "Case #i: Hello *name*!" where i is its number, starting at 1. Each line of the output should end with a line break.

Constraints

- $1 \leq t \leq 20$.
- No name will contain whitespaces.
- The names' lengths will be at most 100.

Sample Data

Input

```
1 2
2 Bob
3 Peter
```

Output

```
1 Case #1: Hello Bob!
2 Case #2: Hello Peter!
```

Submitting programs

Submitting programs is done on the JudgeDB web interface entirely.

- No files to be sent via e-mail etc.
- Only a single source code file is to be uploaded, no object files, executables or similar.

Submit

- Only C, C++ or Rust supported.
- Language automatically selected based on file extension (.c, .cpp or .rs)
- You may need to reload the page to see results (the page does not reload automatically in most cases, even when it tells you so)

Judging

JudgeDB

- compiles,
- executes,
- tests

the submission against several test cases.
The submission is treated instantaneously and JudgeDB (usually) announces its verdict within a few moments.

Judging

The following report codes can occur:

OK

The submission successfully solved all the test cases.

WRONG ANSWER

The submission's output is incorrect.
Possible reasons:

- The answer is just wrong.
- The answer does not conform to the output format specification given on the problem set.
- The answer is not exact enough (e.g. with floating point answers with a desired precision).

Judging

TIME LIMIT EXCEEDED

The submission runs longer than the maximal allowed time and was terminated.
Possible reasons:

- The submission runs in an endless loop.
- The submission is not efficient enough.

MEMORY LIMIT EXCEEDED

Used more memory than the allowed memory limit.

RUNTIME ERROR

The program returned a non-zero exit code.

Judging

RULES VIOLATION

The program attempts illegal syscalls or tries to tamper with the sandbox.

SYSTEM ERROR

Not your fault. Report this to the system administrator Mateusz Gienieczko (giem@in.tum.de).

Limits

Number of submissions	100 (default), can be increased case-by-case
File size	10KiB
Compilation time	30s
Executable size	5MiB
Execution time limit	depending on task
Execution memory limit	depending on task
Architecture	32-bit x86

- No networking, no file system access, no manipulating file descriptors (except stout/stdin), no multithreading.
- In case of any problems: contact us! If you refer to your submission, *always* include the unique submission number.

Points

- The number N of points achievable depends on the task (usually, 4 for easy, 6 for medium, and 8 for hard problems)
- In the first week after the lecture, you only receive partial feedback (report code, and the achieved score, but no feedback on the violated test cases)
- After the first week, results are published, and you receive x points, where x is the best submission score.
- From then on, a second phase starts, where you can still upload new submissions to improve your score.
- In this second phase, you get immediate feedback.
- Finally, you are awarded $x + y$ points, where y is the achieved score after the second phase. ($y \geq x$ always holds, so you don't need to reupload submissions without change just to get points)
- Examples:
  - You uploaded a correct submission in the first phase. You will get $N + N$ points.
  - You uploaded a submission with 3 points in the first phase, and improved to N in the second phase. You get $3 + N$ points.

zulip

Sometimes you want to discuss with other students.
Please discuss problem statements, corner cases, algorithms and approaches. The code should be your own.

You are welcome to use our zulip channels!

C++ Submission

```
#include <iostream>
#include <stdio.h>

int main() {
    // loop over all test cases
    int t;
    scanf("%d", t);
    for(int i = 1; i <= t; i++) {

        // read several types of input
        int j;
        std::string s1;
        char s2[101];
        // use the possibility you like more
        std::cin >> j >> s1;
        scanf("%d %100s", &j, s2);
        // output: use the possibility you like more
        std::cout << "Case #" << i << ": " << s1 << std::endl;
        printf("Case #%d: %s %d", i, s2, j);
    }
    return 0;
}
```

Rust submission

```
fn main () -> Result<(), Box<dyn std::error::Error>> {
    let stdin = std::io::stdin();
    let mut buffer = String::new();
    stdin.read_line(&mut buffer)?;
    let n = buffer.trim_end().parse::<usize>()?;
    for k in 1..n+1 {
        buffer = String::new();
        stdin.read_line(&mut buffer)?;
        let value: isize = std::iter::Sum::sum(
                buffer.split_ascii_whitespace().
                map(|x| x.parse::<isize>().
                        unwrap_or_default()));
        println!("Case␣#{}:␣{}", k, value);
    }
    return Ok(());
}
```

Understanding Problems

- Read the problem statement very carefully.
- Also the constraints, think about special cases:
  - E.g. if there are negative values or 0 allowed, then there is probably a test case for that.
  - E.g. special characters or a space when dealing with strings.
  - ...

Solving Problems

- Code efficiently.
  - Think about which data types to use.
  - Sometimes arrays might not have to be two- or three-dimensional.
  - Sometimes objects add runtime overhead without making code clearer.
  - Implement algorithms given in the lecture with their amortized running times.
- Look carefully at the input and output specifications and let your program be conform to those!
- Remove all debug messages before submitting.
- Write comments!

Code from the Internet

- You are allowed to download and include a library, but you need to cite the correct source.
- Do this by putting a comment in your code stating the url or similar.
  please configure any bundlers/formatters not to lose the reference!
- Including other people's published code without a reference, or using other students' code, is a rule violation
- However, we advise you to code on your own as it improves the understanding about the algorithms involved. You probably need this in subsequent problems anyway.

Debugging your programs

- RUNTIME ERROR supersedes WRONG ANSWER
- For C++ and memory corruption: Valgrind
- Generating completely random input can be faster than waiting for us to answer
  ... do not only write tests by hand, write a generator
- Shortest possible input?
- If you get TIME LIMIT EXCEEDED, look at your program and try constructing its worst input
- Small function mean less variables in scope (more mistakes become compile-time errors)
  ... and small functions can be tested separately
- We may omit the very worst test cases — so if optimisation is useful *almost* always, use it
- If someting is formally allowed to be zero, we might have such a case

Debugging your programs: using a testcase

- Generating completely random input can be faster than waiting for us to answer
  ... do not only write tests by hand, write a generator
- Might be able to write *really* slow but reliable solution to compare
- Check internal assumptions
  Can find mistakes without knowing correct answer
- Print intermediate values and look at them

Big Inputs / Outputs

- Some problems require you to read/write a substantial amount of input/output.
- Even without doing anything else, this can take longer than the allowed time limit when not handled correctly!

Speed up your code easily!

Use the faster readers / writers when handling big data.

C++ - Input

cin

```
#include  <iostream.h>
...
int n;
double d;
cin >> n >> d;
```

scanf

```
int n;
double d;
scanf("%d %i", &n, &d);
```

C++ - Input

Input Size	cin	scanf
5 Mio Integers	1887 ms	552 ms
50 Mio Integers	18789 ms	5467 ms

cin synchronizes with stdio buffers.
Turning this off can make it even faster than scanf.

```
std::ios_base::sync_with_stdio(false);
```

C++ - Output

cout

```
#include  <iostream.h>
...
cout << "Case #" << i << ": " << x << endl;
```

printf

```
printf("Case #%d: %i\n", i, x);
```

C++ - Output

Output Size	cout	printf
5 Mio Integers	11927 ms	492 ms
50 Mio Integers	-	4919 ms

Again, cout can be improved by using the following line.

```
std::ios_base::sync_with_stdio(false);
```

Rust

For Rust: you might need to lock standard IO streams

You can switch languages even between submissions to the same problem

That's it!

Questions?

Until next week:

Homework:

- Register on `https://judge.db.cit.tum.de` and send me your username and real name
- Check out the problem in APC-26-test to try the submission process and code restrictions (cf. technical regulations)

Also:

- Slides will be uploaded to `https://www.cs.cit.tum.de/en/tcs/lehre/ws25/conpra/`
- Keep an eye on zulip, all relevant announcements will be made there