香港電腦奧林匹克競賽
Hong Kong Olympiad in Informatics

TFT Contest Environment Tips

2026-05-02

Overview

The IOI / NOI TFT contest format will be similar to IOI / APIO.

IOI / NOI TFT

1. **Linux Environment**
- You need to know how to compile and run code in Linux.
- Knowledge of basic shell commands for navigation is required.
2. **Grader Task Format**
- You are not required to handle input/output directly.
- You only need to implement specific functions to perform the required tasks.
    - You may be given library functions to call (e.g., functions to report answers or handle interaction).

The best way to practice is to install Linux on your own computer (e.g., using WSL - Windows Subsystem for Linux) and attempt tasks from previous Minicomp 3 or TFT.

Overview

The IOI / NOI TFT contest format will be similar to IOI / APIO, while EGOI TFT format will be similar to EGOI.

IOI / NOI TFT

1. **Linux Environment**
- You need to know how to compile and run code in Linux.
- Knowledge of basic shell commands for navigation is required.
2. **Grader Task Format**
- You are not required to handle input/output directly.
- You only need to implement specific functions to perform the required tasks.
    - You may be given library functions to call (e.g., functions to report answers or handle interaction).

The best way to practice is to install Linux on your own computer (e.g., using WSL - Windows Subsystem for Linux) and attempt tasks from previous Minicomp 3 or TFT.

Overview

The IOI / NOI TFT contest format will be similar to IOI / APIO, while EGOI TFT format will be similar to EGOI.

EGOI TFT

1. **Linux Environment**

- You need to know how to compile and run code in Linux.
- Knowledge of basic shell commands for navigation is required.

2. **Standard I/O Task Format** (for EGOI TFT)

- You will process input/output through standard I/O.
  - I/O Optimization trick like **ios_base::sync_with_stdio(false)**, **cin.tie(nullptr)** may be useful
  - For interactive task, you should interact with the system through standard I/O, and flush the output after every interaction, i.e. **fflush(stdout)**

The best way to practice is to install Linux on your own computer (e.g., using WSL - Windows Subsystem for Linux) and attempt tasks from previous Minicomps and TFT (2021 or before).

Basic Linux Shell Commands

Concept: Your terminal session always has a “current directory”, i.e., the folder your terminal is currently operating within.

Command 1: **`pwd`**
> Print Working Directory: Shows the full path of the current directory.

Command 2: **`ls`**
> List: Shows files and directories within the current directory.

Command 3: **`cd [dir_name]`**
> Change directory: Changes the current directory to the one specified by [dir_name].

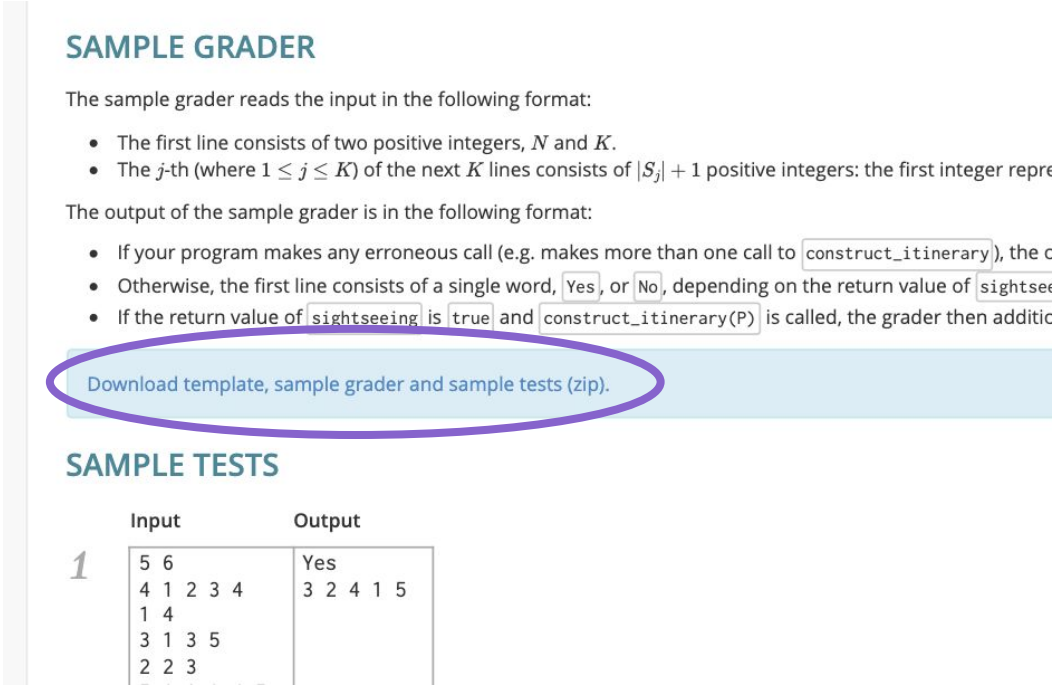
Command 4: **`cd ..`**
> Move up one level: Changes the current directory to the parent directory of the one you are currently in.

Other useful commands include **`mkdir`** (make directory), **`cp`** (copy), **`mv`** (move/rename), and **`rm`** (remove). They could be useful but you could handle these with GUI during contest.

Compile & Run

Find the **"Download template, sample grader and sample tests (zip)."** within the task and download the zip.

Unzip the zip file.



SAMPLE GRADER

The sample grader reads the input in the following format:

- The first line consists of two positive integers, N and K.
- The j-th (where $1 \leq j \leq K$) of the next K lines consists of $|S_j| + 1$ positive integers: the first integer repre

The output of the sample grader is in the following format:

- If your program makes any erroneous call (e.g. makes more than one call to `construct_itinerary`), the c
- Otherwise, the first line consists of a single word, `Yes`, or `No`, depending on the return value of `sightsee`
- If the return value of `sightseeing` is `true` and `construct_itinerary(P)` is called, the grader then additic

Download template, sample grader and sample tests (zip).

SAMPLE TESTS

	Input	Output
1	5 6 4 1 2 3 4 1 4 3 1 3 5 2 2 3	Yes 3 2 4 1 5

Compile & Run

There are two steps (*<< simplified*) to execute your C++ program: compile & run

- *Compile: Convert C++ source code into an executable file*
- *Run: Load the executable into memory and execute*

1. [Suggested] Use the provided script (**compile_cpp.sh**) in the downloaded zip.
   To compile, simply run: `sh compile_cpp.sh`

2. Running the Executable
   After successful compilation, run the created executable file: `./sightseeing`
   (Replace `sightseeing` with the actual executable name).

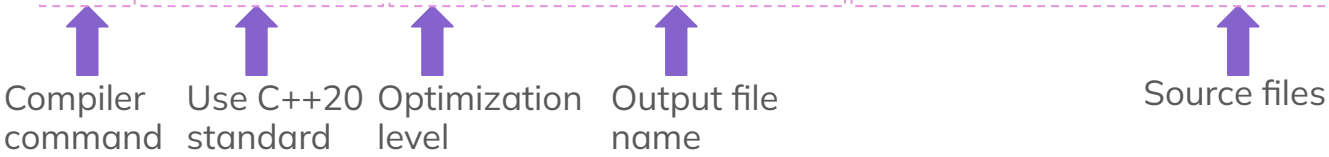
Compile & Run

- Understanding the compile_cpp.sh Compile Script:
  `g++ -static -std=c++20 -Wno-unused-result -s -O2 -Ilib -o sightseeing sightseeing.cpp sample_grader.cpp`
  - **`g++`**: Compiler command.
  - **`-std=c++20`**: Use C++20 standard.
  - **`-O2`**: Optimization level.
  - **`-o sightseeing`**: Output executable named `sightseeing`. The argument directly after the -o flag is the output name.
  - **`sightseeing.cpp`**: Your solution file.
  - **`sample_grader.cpp`**: The grader file.
  - Other flags (-static, -Wno..., -s, -Ilib): Handle linking, warnings, symbols, libraries (you usually don't need to worry about these when using the script).

Compile & Run

- Simpler compile script you can recite (if some contest doesn't provided script)

```
g++ -std=c++20 -O2 -o sightseeing sightseeing.cpp sample_grader.cpp
```



- To compile non-grader style program (already contain `main()`), just omit the grader file

```
g++ -std=c++20 -O2 -o sightseeing sightseeing.cpp
```

- Some task may require you to compile multiple source files together

```
g++ -std=c++20 -O2 -o communication_task alice.cpp bob.cpp sample_grader.cpp
```

(Just put all source files in the command)

Output Only

- For Output-Only task, the grader zip is not provided. You should write and compile a program first.
  e.g. `g++ -std=c++20 -O2 -o cycle cycle.cpp` (the executable is named `cycle`)

- You can either use file I/O or I/O redirection to read from input file and write to input file. (Choose 1 below)
  - Easiest to do file I/O is to add
    `freopen("coprime1.in", "r", stdin)` and `freopen("coprime1.out", "w", stdout)`.
  - To do I/O redirection, just do `./cycle < coprime1.in > coprime1.out`
- You can do loop with the redirection command:

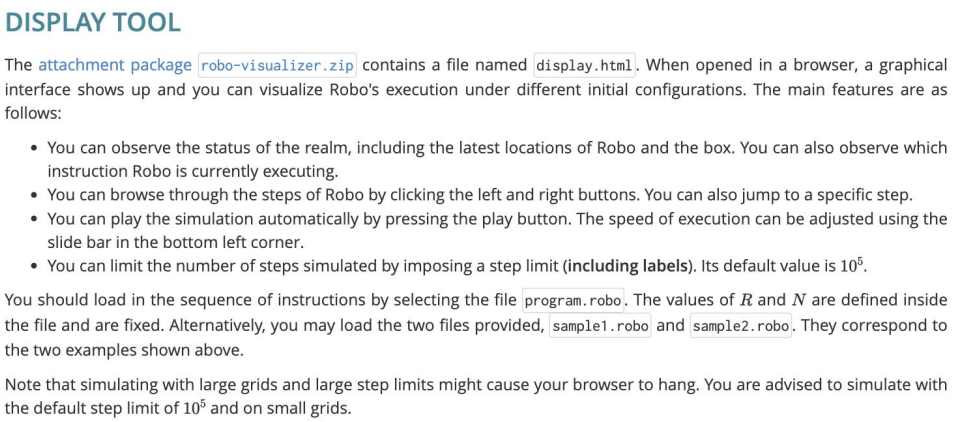
```
for i in {1..10}; do
    ./cycle < "coprime${i}.in" > "coprime${i}.out"
done
```

- Easy way to zip the output files: `zip answer.zip *.out`

Other tools

- Some task provide additional tools e.g. visualizer, testers to help you debug your code. The instructions will be provided per task. You can learn to use those from previous contests to get familiar with them.

- e.g. T254 - Robo's Recursive Realm (HTML-based, use browser to open it)



DISPLAY TOOL

The attachment package `robo-visualizer.zip` contains a file named `display.html`. When opened in a browser, a graphical interface shows up and you can visualize Robo's execution under different initial configurations. The main features are as follows:

- You can observe the status of the realm, including the latest locations of Robo and the box. You can also observe which instruction Robo is currently executing.
- You can browse through the steps of Robo by clicking the left and right buttons. You can also jump to a specific step.
- You can play the simulation automatically by pressing the play button. The speed of execution can be adjusted using the slide bar in the bottom left corner.
- You can limit the number of steps simulated by imposing a step limit (**including labels**). Its default value is 10^5.

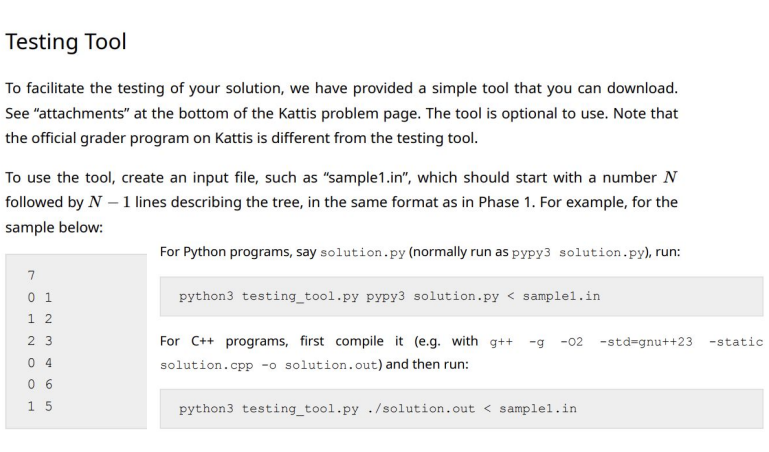
You should load in the sequence of instructions by selecting the file `program.robo`. The values of R and N are defined inside the file and are fixed. Alternatively, you may load the two files provided, `sample1.robo` and `sample2.robo`. They correspond to the two examples shown above.

Note that simulating with large grids and large step limits might cause your browser to hang. You are advised to simulate with the default step limit of 10^5 and on small grids.

Other tools

- Some task provide additional tools e.g. visualizer, testers to help you debug your code. The instructions will be provided per task. You can learn to use those from previous contests to get familiar with them.

- e.g. EGOI 2025 D2T4 - Laser Strike (Python based)



Testing Tool

To facilitate the testing of your solution, we have provided a simple tool that you can download. See "attachments" at the bottom of the Kattis problem page. The tool is optional to use. Note that the official grader program on Kattis is different from the testing tool.

To use the tool, create an input file, such as "sample1.in", which should start with a number N followed by $N-1$ lines describing the tree, in the same format as in Phase 1. For example, for the sample below:

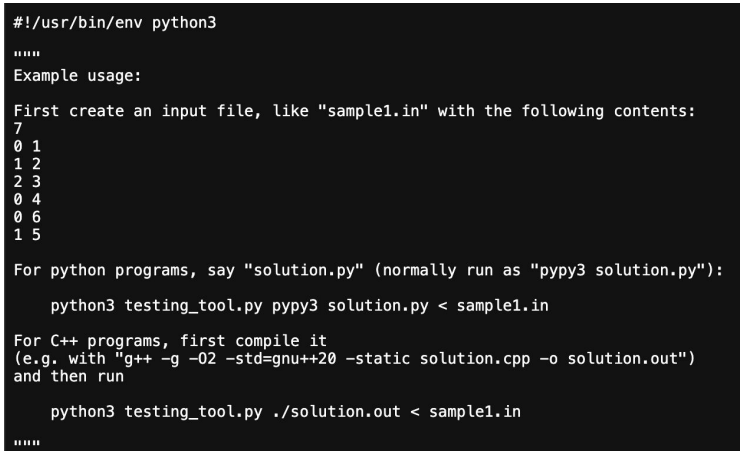
```
7
0 1
1 2
2 3
0 4
0 6
1 5
```

For Python programs, say `solution.py` (normally run as `pypy3 solution.py`), run:

```
python3 testing_tool.py pypy3 solution.py < sample1.in
```

For C++ programs, first compile it (e.g. with `g++ -g -O2 -std=gnu++23 -static solution.cpp -o solution.out`) and then run:

```
python3 testing_tool.py ./solution.out < sample1.in
```



```
#!/usr/bin/env python3

"""
Example usage:

First create an input file, like "sample1.in" with the following contents:
7
0 1
1 2
2 3
0 4
0 6
1 5

For python programs, say "solution.py" (normally run as "pypy3 solution.py"):

    python3 testing_tool.py pypy3 solution.py < sample1.in

For C++ programs, first compile it
(e.g. with "g++ -g -O2 -std=gnu++20 -static solution.cpp -o solution.out")
and then run

    python3 testing_tool.py ./solution.out < sample1.in

"""
```

Good Luck!