Store to Nowhere

Translated by Gemini 3

Important Notice

This is an interactive problem. Please pay attention to the extremely unusual memory limits (256 Bytes).

Problem Description

A permutation p is a sequence containing every integer from 1 to n exactly once.

For a permutation $p = (p_1, p_2, \ldots, p_n)$, its inverse permutation $q = (q_1, q_2, \ldots, q_n)$ is defined as follows: if $p_i = j$, then $q_j = i$.

In this problem, you will be given a permutation of size n. Your task is to calculate its inverse permutation.

Implementation Details

Unlike conventional problems, you do not need to write a complete program (i.e., you do not need to implement the `main` function). Instead, you must include the header file `perm.h` in a C/C++ source file and implement the following function:

```
1 void perm_work(int n);
```

The grading system will call your `perm_work` function exactly once. The parameter n is the size of the permutation.

You cannot directly access the array storing the permutation. You must interact with the permutation through the following functions provided by the grading system:

- `int perm_get(int x)`: Returns the value of the x-th element in the permutation, i.e., p_x. Here, x is a 1-based index satisfying $1 \le x \le n$. If x is out of range, the function returns -1.
- `void perm_set(int x, int y)`: Sets the value of the x-th element in the permutation to y, i.e., $p_x = y$. Here, x is a 1-based index. You must ensure $1 \le x, y \le n$; otherwise, the function has no effect.

Your `perm_work` function should modify the permutation in-place by calling `perm_get` and `perm_set` so that it becomes its own inverse permutation.

Stack Configuration

Specifically, the first line of your submitted program may contain a comment in the format `/* stack = S */` to specify the size of the program stack. S should be a positive integer representing the stack size in bytes. If not specified, the grading system will use the default value of **128** bytes.

Environment Limits

Your program is allowed a total linear memory of **only 256 bytes**. You can understand this as the entire memory space your C/C++ code can directly access. This memory is divided into the following parts:

- **Program Stack**: S bytes. This memory is used to store local variables, return addresses, etc., during function calls. For example, `int i;` declared inside `perm_work` will occupy this space. This is the same concept as the stack in a standard C/C++ program.
- **Static Memory**: $256 - 16 - S$ bytes. This space is used to store all global variables, static variables, and constants (e.g., string literals) in the program. All variables defined outside of functions will occupy this space.
- **System Reserved**: **16** bytes.

In addition to the linear memory above, the WebAssembly runtime itself requires an independent **Execution Stack** (Runtime Stack) to manage the program's execution flow. This stack is not in linear memory, and contestants cannot access it directly. Its main role is to handle function calls at the WebAssembly instruction level.

- **Execution Stack**: In this problem, its size is set to **1024** bytes. For most programs, you do **not** need to worry about the usage of the execution stack. You primarily need to worry about whether the "Program Stack" will overflow. However, extremely deep function recursion may exhaust the execution stack space, causing a runtime error.

To ensure programs can compile and run under such strict memory limits, there are special restrictions:

- **No Standard Library**: You **cannot** use any C/C++ standard libraries.
- **Disabled C++ Features**: For C++ contestants, disabling the standard library also means that `new`, `delete` operators, and exception handling are unavailable.

Grading Method

Your C/C++ source code will be compiled into a format called **WebAssembly (WASM)**. The grading system will execute this WASM module and interact with it. The grader only counts the WASM Ticks and memory of your program, not the interactor.

To be compatible with UOJ's API:

- **Time**: Displayed as WASM Ticks divided by 10^6 (rounded down).

- **Memory**: 1 actual byte is displayed as 1 "kb". (Since UOJ's "kb" is actually KiB, this is equivalent to displaying the usage multiplied by **1024**).

Self-Testing

Two methods are provided for self-testing:

1. **UOJ Custom Test**: Input format: The first line contains an integer n. The next line contains n integers $p_1, p_2, \ldots, p_n$. The output will be the modified permutation. This follows the actual grading process but does not check correctness.
2. **Local Testing**: Download the attachment package containing `perm.h` and `user_grader.c`. Place these files with your code. Compile using:

   ```
   <gcc | g++> your_code.cpp user_grader.c -o prog
   ```

   Run `./prog` and input data as described above. Note that local testing cannot provide WASM Ticks or accurate memory usage.

Constraints & Scoring

Subtasks: You must pass all test points in a subtask to receive credit.

Subtask	Score	Constraints
1	10	$n \leq 10$
2	50	$n \leq 10^4$
3	40	No special restrictions

For all data: $1 \leq n \leq 3 \times 10^5$.

- **WASM Tick Limit**: 10^{10} ticks.
- **Memory Limit**: **256** bytes.

Compiler Information

- **Compiler Suite**: wasi-sdk 28_judge.0 (Based on LLVM/Clang 21.1.4).
- **Versions**: wasi-libc: `47e504e7c024`, llvm: `222fc11f2b8f`.
- **Repo**: https://github.com/aberter0x3f/wasi-sdk

Compilation Flags:

- `-nostdlib`, `-nostdinc`, `-ffreestanding`: Disable standard library.

- `-fno-exceptions`: Disable C++ exceptions.
- `-O0`: Disable optimizations.
- `-Wl,-z,stack-size=S`: Set stack size to ***S*** bytes.
- `-Wl,-max-memory=256`: Set total available linear memory to **256** bytes.
- `-Wl,-export=perm_work`: Export your function.
- `-Wl,-no-entry`: No `main` entry point.