A. TON Block Compression

time limit per test: 2 seconds
memory limit per test: 256 megabytes

A *block* is a fundamental data structure in the TON blockchain. It contains a list of transactions, messages, and the difference between the previous and next blockchain states.

To reduce network load, efficient block compression is essential. In this contest, your task is to implement a compression algorithm that will be evaluated on blocks from the real TON blockchain.

Everything in TON is composed of *cells*. If you participated in FunC contests, you might already be familiar with this concept. A *cell* is a data structure that contains up to 1023 bits and up to 4 references to other cells. A root cell, together with all cells reachable from it, forms a *bag of cells*. A block is represented as a bag of cells, and its structure can be formally described using *TL-B schema* (see links in the Notes section).

The block is serialized to a byte string using the bag of cells serialization algorithm. This byte string is the representation of the block that serves as the input data in this problem.

Interaction

For each test, your program is executed twice: once for compression and once for decompression.

First run (compression)

The first line of the input contains one word: "`compress`". The second line of the input contains a base64-encoded block.

Output base64-encoded compressed block.

Second run (decompression)

The first line of the input contains one word: "`decompress`". The second line of the input contains the base64-encoded compressed block from the output of the first run.

Output base64-encoded decompressed block.

Limits

The maximum block size is $2 \cdot 2^{20}$ bytes. The compressed block size should be at most $2 \cdot 2^{20}$ bytes.

Note: These limits refer to the size of the decoded data, not its base64-encoded representation.

Scoring

For each test, your solution will be scored based on the compression efficiency.

The decompressed block must be equal to the original block: otherwise, the solution will receive WA and zero points for that test.

If the size of the original and compressed blocks are x and y, respectively, then the score for that test will be $1000 \cdot \frac{2 \cdot x}{x+y}$.

Testsets

1. During the contest, your solution will be evaluated on a closed preliminary test set. This set consists of various recent blocks from the TON mainnet. The results will be available on a public scoreboard.
2. The first 25 tests from this test set are publicly available in `ton-sample-tests.zip`. You can also download more blocks from the TON blockchain for local testing.
3. After the contest, your solution will be evaluated on a different final test set. This set will consist of blocks from the TON mainnet that will be generated after the end of the contest.

MaraTON Challenge 1
Contest is running
3 weeks
Contestant

→ Languages

The following languages are only available languages for the problems from the contest

MaraTON Challenge 1:
- C++17 (clang 18-64 + ton_crypto_lib)

→ Submit?

Language: C++17 (clang 18-64 + ton_c
Choose file: Seleccionar archivo Sin a...nados
Submit

→ Contest materials

- problem-a-ton-local-tester.zip
- problem-a-ton-compress-contest-examples.zip
- problem-a-ton-sample-tests.zip
- problem-a-ton-compress-contest-windows.zip

→ Your points

	Points
A	

For this final testing, your latest submission with non-zero score on the preliminary testing will be taken.

Note

Your solution will be linked with the C++ library `ton_crypto_lib`. This library is based on the main TON repository and contains utilities for working with cells, parsing blocks, and the LZ4 compression algorithm. When submitting your solution, select the language "`C++17 (clang 18-64 + ton_crypto_lib)`". You can find instructions for downloading and linking the library here.

Here is the solution example. It demonstrates using `ton_crypto_lib` to decode and encode base64, serialize and deserialize cells, LZ4-compress and decompress data.

Local build and ton_crypto_lib examples

Download `ton-compress-contest-examples.zip`. It contains code examples that explain TON concepts and show how to work with the library, as well as the scripts for building your solution for Linux and MacOS. Usage:

1. **Linux**: Extract `ton_crypto_lib-x86_64-linux.zip`, run `./run-linux.sh solution.cpp`
2. **MacOS**: Extract `ton_crypto_lib-x86_64-macos.zip`, run `./run-macos.sh solution.cpp`
3. **Windows**: Download `ton-compress-contest-windows.zip` with the configured MSVC++ project.

Instructions for manual configuration can be found here.

Local testing

You can use the docker image in `ton-local-tester.zip` to test your solutions:

- Put your solution to `solution.cpp`
- `tests` directory contains the 25 publicly available tests. You can add your own tests to this directory.
- Install Docker and run:

```
docker build -t local_tester . && docker run --rm -it local_tester
```

Useful documentation

1. TON blockchain whitepaper: contains a detailed description of the blockchain structure and the block format.
2. block.tlb: formal specification of TON data structures. TL-B format overview.
3. Cells: documentation on cells.
4. Mainnet explorer: blockchain explorer.