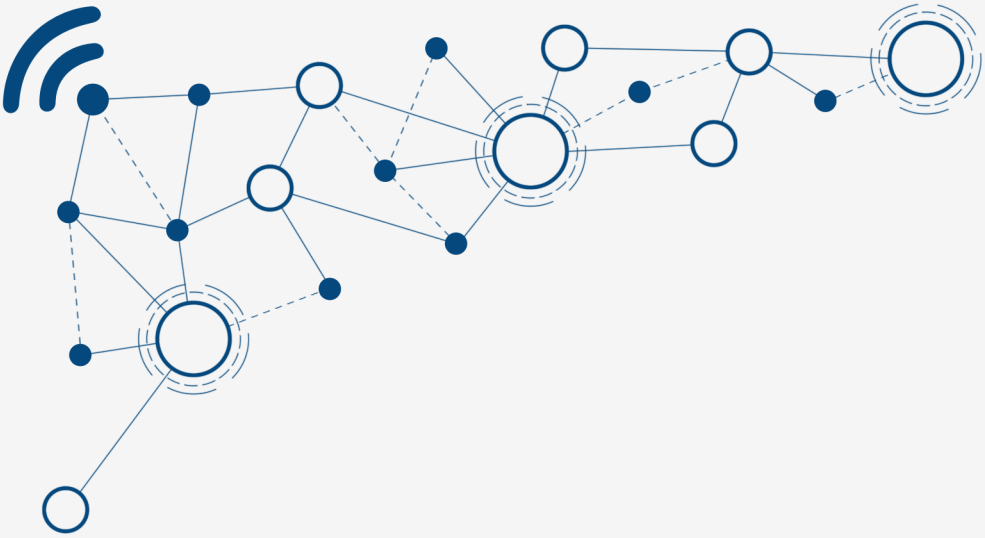
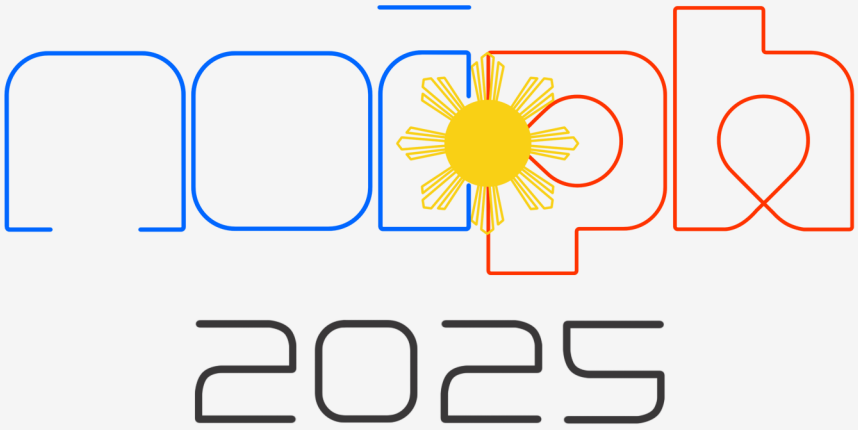
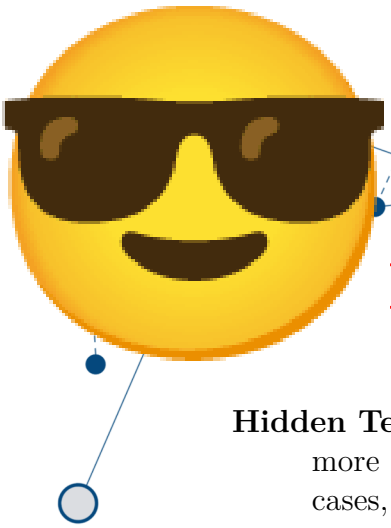




National Olympiad in Informatics

Finals Round 2



Important! Read the following:

Hidden Test Cases. Your solution will be checked by running it against one or more (usually several) hidden test cases. You will not have access to these cases, but a correct solution is expected to handle them correctly.

Strict Output Format. The output checker is **strict**. Follow these guidelines strictly:

- It is **space sensitive**. Do not output extra leading or trailing spaces. Do not output extra blank lines unless explicitly stated.
- It is **case sensitive**. So, for example, if the problem asks for the output in lowercase, follow it.
- Do not print any tabs. (No tabs will be required in the output.)
- Do not output anything else aside from what's asked for in the Output section. So, do not print things like "`Please enter t`".

Not following the output format strictly and exactly will likely result in the verdict "*Output isn't correct*".

Use Standard I/O. Do not read from, or write to, a file. You must read from the standard input and write to the standard output.

Submit Code Only. Only include **one** file when submitting: the source code (.cpp, .py, etc.) and nothing else.

No Java Package. For Java submissions, do not include a `package` line.

No Weird Filenames. Only use letters, digits and underscores in your filename. Do not use spaces or other special symbols.

Use Fast I/O. Many problems have large input file sizes, so use fast I/O. For example:

- In C/C++, use `scanf` and `printf`.
- In Python, use `sys.stdin.readline()`

Flush On Interactive Problems. On interactive problems, make sure to **flush** your output stream after printing.

- In C++, use `fflush(stdout);` or `cout << endl;`
- In Python, use `sys.stdout.flush()` or `print(flush=True)`
- For more details, including for other languages, ask a question/clarification through CMS.

Good luck and enjoy the contest! 😃

Contents

Notes

- Many problems have large input file sizes, so use fast I/O. For example:
  - In C/C++, use `scanf` and `printf`.
  - In Python, use `sys.stdin.readline()`
- On interactive problems, make sure to **flush** your output stream after printing.
  - In C++, use `fflush(stdout);` or `cout << endl;`
  - In Python, use `sys.stdout.flush()` or `print(flush=True)`
  - For more details, including for other languages, ask a question/clarification through CMS.

Good luck and enjoy the problems!

Problem D
CJ's Marble Runs

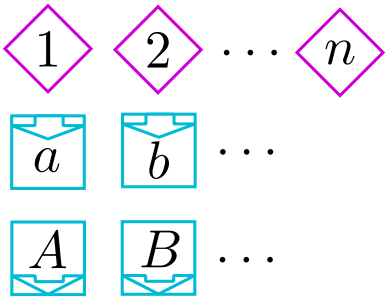
CJ noticed that the "simulation" genre of game is popping off on Steam right now, turning otherwise banal premises into highly addictive and satisfying gameplay. He has a great idea for a simulation game, he just hopes you won't think he's lost his marbles.

CJ's Marble Runs *is a brand new simulation game, all about setting up elaborate marble networks and then wasting hours watching marbles whizz by in your grand creations. The fun thing about video games though... is that they're not constrained by the limitations of reality.*

A **marble network** is a set of nodes and pipes. Marbles go between nodes using pipes.

Let's examine the types of nodes that a marble network might have.

- n **router nodes** numbered $1, 2, \ldots, n$;
- one or more **input nodes** labeled $a, b, \ldots$;
- one or more **output nodes** labeled $A, B, \ldots$

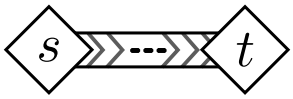


A marble network also has m pipes, labeled $1, 2, \ldots, m$. Each pipe has a **source node** and a **target node**. We can draw this with an arrow going from the source node to the target node.

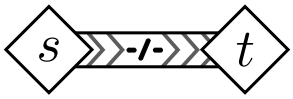
Marbles are placed into our network from the input nodes, and cause output to be printed when they enter an output node. Router nodes are just additional extra nodes that our marbles might pass through along the way.

Whenever a marble is sent to some node, a marble then enters *each pipe* that has that node as a source node. Whenever a marble enters a pipe, the pipe *may* send a marble to its target node, depending on the kind of pipe.

There are three kinds of pipes in CJ's Marble Runs. Let's familiarize ourselves with all of them!

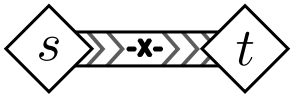


Repeaters, denoted by `-` characters: For very marble sent to the source node, the repeater sends a marble to the target node.



Dividers, denoted by `/` characters: For every *other* marble send to the source node, the divider sends a marble to the target node. So...

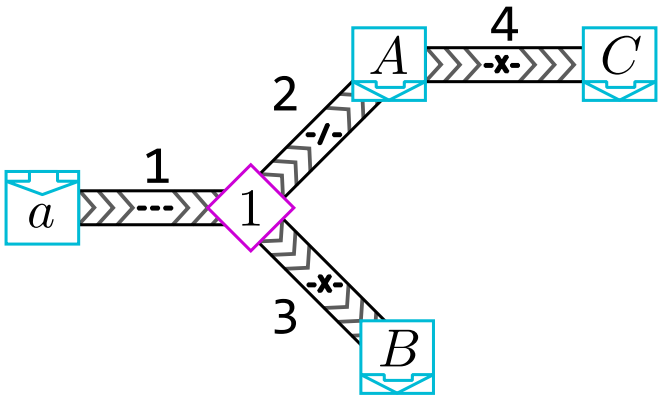
- the first marble sent to the source does nothing,
- the second marble sent to the source sends a marble to the target,
- the third marble sent to the source does nothing,
- the fourth marble sent to the source sends a marble to the target, etc.



Sinks, denoted by `X` characters: For *only the first* marble sent to the source node does the sink send a marble to the target node. So...

- the first marble sent to the source sends a marble to the target,
- the second marble sent to the source does nothing,
- the third marble sent to the source does nothing, etc.

Putting everything together, here's one possible example of a marble network:



Recall that the pipes are *labeled* from 1 to m, as indicated in the above completed network. Why? Well...

One by one, we will send marbles into input nodes and record the results as the simulation plays out. We write the label of an input node whenever we send a marble to it, and write the label of an output node when a marble is sent to it.

We always wait for the current marble run to finish completely before manually sending another marble.

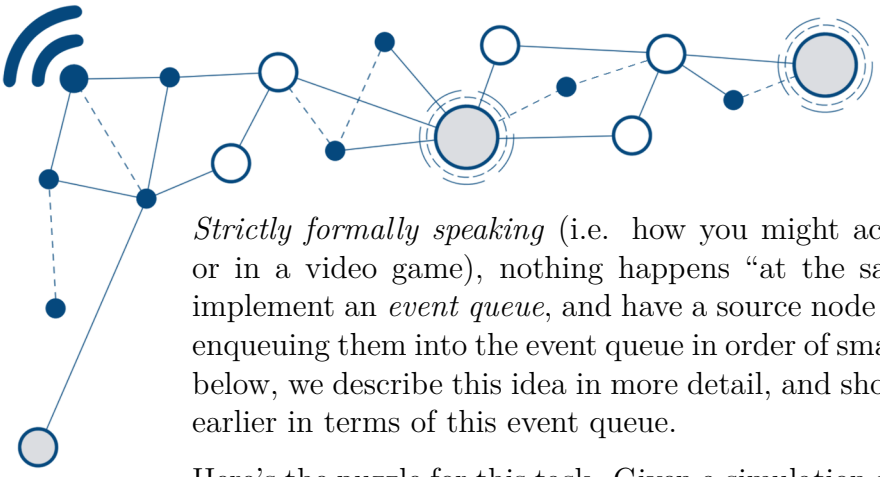
The pipes' labels act as a "tie-breaker" to unambiguously resolve what would happen if multiple marbles enter different output nodes "at the same time", where the pipes with smaller labels trigger .

Note that input nodes are not disallowed from being target nodes of some pipe, but we only write an input node's label whenever *we* manually send a marble to it, **never** when a pipe sends a marble to it.

So, a sequence of input nodes (meaning we sent marbles into those input nodes in that order) defines a **simulation string**, the string that would be produced by our marble network.

You can verify that in the network illustrated above, the sequence $[a, a, a, a]$ (meaning we send a marble to input node a four times) would produce the simulation string `aBaACaaA` (color only added for illustration).

- We send a marble to a, which sends a marble to repeater pipe 1.
  - This sends a marble to node 1, which sends marbles to divider pipe 2 and sink pipe 3.
  - Divider pipe 2 does not send out a marble (it only does every *other* time). Sink pipe 3 sends a marble to node B.
- We send a marble to a, which sends a marble to repeater pipe 1.
  - This sends a marble to node 1, which sends marbles to divider pipe 2 and sink pipe 3.
  - Divider pipe 2 now *does* send a marble to node A, which sends a marble to sink pipe 4. Sink pipe 3 does not send out a marble anymore (it's been "used up").
  - Sink pipe 4 sends a marble to node C.
- We send a marble to a, which sends a marble to repeater pipe 1.
  - This sends a marble to node 1, which sends marbles to divider pipe 2 and sink pipe 3.
  - Neither divider pipe 2 nor sink pipe 2 send out a marble.
- We send a marble to a, which sends a marble to repeater pipe 1.
  - This sends a marble to node 1, which sends marbles to divider pipe 2 and sink pipe 3.
  - Divider pipe 2 now *does* send a marble to node A, which sends a marble to sink pipe 4. Sink pipe 3 does not send out a marble anymore.
  - Sink pipe 4 does not send out a marble anymore (it's been "used up").



Strictly formally speaking (i.e. how you might actually implement this in code, or in a video game), nothing happens "at the same time". Instead, we would implement an *event queue*, and have a source node send marbles to all its pipes by enqueuing them into the event queue in order of smallest label first. In an appendix below, we describe this idea in more detail, and show the example simulation from earlier in terms of this event queue.

Here's the puzzle for this task. Given a simulation string (and the number of input and output nodes, as mandated by that simulation string), construct *any* marble network that would produce it. It is guaranteed that this task will be possible for all simulation strings given as input.

You may use as many router nodes and pipes as you like (within reason). The fewer pipes you use, the higher your score will be!

Input Format

The first line of input contains three space-separated integers: the number of input nodes, the number of output nodes, and the length of the simulation string.

The second line of input contains the simulation string.

Output Format

First, output two space-separated integers n and m: the number of router nodes in your solution, and the number of pipes.

Then, output m lines, each in the form of three space-separated tokens,

```
<source_node> <pipe_kind> <target_node>
```

where each node is a valid node in the network (a letter of an existing input/output node, or an integer from 1 to n), and `pipe_kind` is one of `-` or `/` or `X` for a repeater, divider, or sink pipe, respectively.

These labels will be labeled 1 to m in the order you give them in the output. If there are multiple answers, any will be accepted within the following constraints:

- You may only use at most 10^4 router nodes.
- You may only use at most 10^5 pipes.
- There must only be at most 10^3 pipes in the event queue at any given time (see appendix for a formal description of the event queue).
- The pipes "act" (triggered by receiving a marble, whether or not it will then send one out) no more than 10^6 times in total.

Constraints

The test cases are publicly available to you, included as the attachment `marble-network.zip` to this problem.

Each test case is worth up to 10 points, where your score is determined by the following formula.

If your program does not produce the indicated simulation string, you get 0 points. Otherwise, let m be the number of pipes in your solution for that test case, and let M be the number of pipes in the judge's best solution. The scoring formula is as follows.

$$\text{score}(m, M) = \begin{cases} 0 & \text{if } 10^5 < m \\ 3 + 5 \cdot \left(1 - \frac{\log_{3M}(m-M)-1}{\log_{3M}(10^5-M)-1}\right)^3 & \text{if } 4M < m \leq 10^5 \\ 8 + \frac{1.9}{3} \cdot \left(4 - \frac{m}{M}\right) & \text{if } M < m \leq 4M \\ 10 & \text{if } m \leq M \end{cases}$$

Sample I/O

Input 1	Output 1
1 3 8 aBaACaaA	1 4 a - 1 1 / A 1 X B A X C

Formalization of Event Queue

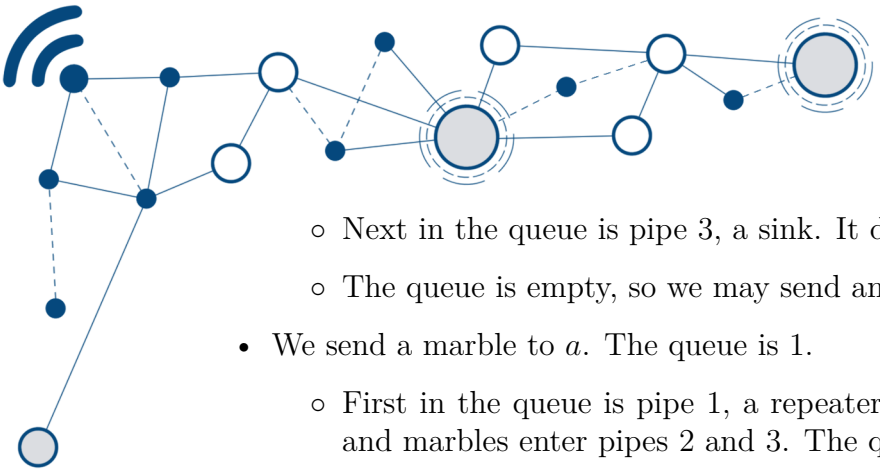
Formally, we consider the pipes to act in a queue, which initially starts empty.

- Whenever a marble enters a pipe, it is appended to the queue.
- Then, the first pipe in the queue possibly sends a marble to its target node, before being removed from the queue.
- Nodes always act instantly, and send their marbles to pipes in order from smallest to largest index.

Marbles can only be sent to an input node if the queue is empty, i.e. you can't send another marble into an input node until the current marble run has finished resolving.

Suppose we consider the sequence of input nodes $[a, a, a, a]$. Then, the simulation string produced by that sequence using the above illustrated marble network would be `aBaACaaA`, describing the following simulation:

- We send a marble to a, which sends a marble to pipe 1. The queue is $[1]$.
  - First in the queue is pipe 1, a repeater. It sends a marble to node 1, and marbles enter pipes 2 and 3. The queue is $[2, 3]$.
  - Next in the queue is pipe 2, a divider. It does nothing. The queue is $[3]$.
  - Next in the queue is pipe 3, a sink. It sends a marble to node B, which is source node to no pipe.
  - The queue is empty, so we may send another marble.
- We send a marble to a. The queue is $[1]$.
  - First in the queue is pipe 1, a repeater. It sends a marble to node 1, and marbles enter pipes 2 and 3. The queue is $[2, 3]$.
  - Next in the queue is pipe 2, a divider. It sends a marble to node A, and a marble enters pipe 4. The queue is $[3, 4]$.
  - Next in the queue is pipe 3, a sink. It does nothing. The queue is $[4]$.
  - Next in the queue is pipe 4, a sink. It sends a marble to node C, which is source node to no pipe.
  - The queue is empty, so we may send another marble.
- We send a marble to a. The queue is 1.
  - First in the queue is pipe 1, a repeater. It sends a marble to node 1, and marbles enter pipes 2 and 3. The queue is $[2, 3]$.
  - Next in the queue is pipe 2, a divider. It does nothing. The queue is $[3]$.



- Next in the queue is pipe 3, a sink. It does nothing.
- The queue is empty, so we may send another marble.

- We send a marble to a. The queue is 1.
  - First in the queue is pipe 1, a repeater. It sends a marble to node 1, and marbles enter pipes 2 and 3. The queue is $[2, 3]$.
  - Next in the queue is pipe 2, a divider. It sends a marble to node A, and a marble enters pipe 4. The queue is $[3, 4]$.
  - Next in the queue is pipe 3, a sink. It does nothing.
  - Next in the queue is pipe 4, a sink. It does nothing.
  - The queue is empty, so we may send another marble.