문제 B. NPU 최적화

시간 제한 3 초 메모리 제한 1024 MB

Furiosa AI에서는 인공지능 모델의 학습 및 추론을 기존의 처리 유닛(processing unit)보다 더욱 빠르게 할 수 있는 NPU(Neural Processing Unit)를 만들고 있다. 일반적인 처리 유닛에서 동작하는 프로그램은 호스트(host)에 저장된 데이터를 여러 종류의 연산자를 사용해 처리하여 원하는 값을 계산해 낸다. 이 문제에서는 이 과정을 단순화해, 다음과 같은 환경을 생각한다.

- 호스트에는 1 000 000 개의 데이터를 저장할 수 있는 공간이 있고, 각 공간에는 0 번부터 999 999 번까지의 번호가 붙어 있다.
- 각 연산자는 한 개 이상의 입력 데이터를 받아서 한 개의 출력 데이터를 계산한다. 연산자에도 0 이상 999 999 이하의 번호가 붙어있다.
- 프로그램은 다음과 같은 BNF(Backus-Naur Form)으로 표현된다.

```
<number> ::= 0 | 1 | ... | 999999
<value> ::= <number> | <number>(<list>)
<list> ::= <value> | <list>,<value>
```

- 프로그램은 값 하나를 계산한다. 즉 프로그램은 <value>에 해당하는 문자열이다.
- <value>의 값은 다음과 같이 계산된다.
  - <value>가 <number>로 표현되는 경우, 프로그램의 시작 전 호스트의 <number>번 공간에 저장된 데이터이다.
  - <value>가 <number>(<list>)로 표현되는 경우, <list> 안의 <value>들을 차례대로 입력 데이터로 하여 <number>번 연산자를 사용해 계산한 출력 데이터이다.
- 하나의 프로그램에는 같은 <number>가 여러 번 등장하지 않는다.

Furiosa AI에서 개발하는 NPU는 같은 프로그램을 일반 처리 유닛보다 더 빠르게 실행할 수 있지만, 이를 위해서는 프로그램을 NPU가 사용하는 저수준의 명령어들로 새로 컴파일하는 컴파일러가 필요하다. 같은 프로그램이라도 어떻게 컴파일하느냐에 따라 프로그램의 메모리 사용량이나 실행 시간이 바뀌기 때문에, 효율적으로 NPU를 사용하기 위해서는 최적화된 컴파일러가 필요하다.

NPU에는 M개의 데이터를 저장할 수 있는 메모리가 있으며, 메모리의 각 공간에는 0 번부터 $M-1$ 번까지의 번호가 붙어 있다. NPU는 다음과 같은 세 종류의 명령어를 지원한다.

- a >> b
  - 호스트의 a번 데이터를 메모리의 b번 공간에 복사한다. $(0 \le a < 1\,000\,000;\ 0 \le b < M)$ 데이터가 이미 있는 경우에는 덮어쓴다.
- a << b
  - 메모리의 b번 데이터를 호스트의 a번 공간에 복사한다. $(0 \le a < 1\,000\,000;\ 0 \le b < M)$ 데이터가 이미 있는 경우에는 덮어쓴다.

- $o = w \mid m_1\ m_2 \cdots m_l$
  - 메모리의 $m_1, m_2, \cdots, m_l$ 주소의 값을 차례대로 입력 데이터로 하는 w 연산자의 출력 데이터를 메모리의 o 공간에 저장한다.
  - 출력 데이터가 저장되는 공간은 입력 데이터가 저장된 공간과 달라야 한다. 즉 $o \neq m_i$ $(1 \le i \le l)$ 이어야 한다.

NPU 프로그램은 위의 명령어들의 나열으로, 프로그램을 실행하면 프로그램을 이루는 명령어들이 차례대로 실행된다.

프로그램의 효율은 다양한 요인에 의해 결정되지만, 일단 사용하는 명령어 수가 적다면 효율적인 프로그램일 가능성이 높다. `<value>`에 해당하는 문자열이 주어지면, 명령어를 최대한 적게 사용해 `<value>`의 값을 계산해서 0번 메모리에 저장하는 NPU 프로그램으로 변환해 보자.

입력

첫 번째 줄에는 NPU의 메모리의 크기 M이 주어진다. $(1 \le M \le 1\,000\,000)$

두 번째 줄에는 계산해야 하는 `<value>`에 해당하는 문자열이 주어진다. 이 문자열의 길이는 1 000 000 이하이다.

출력

NPU의 메모리가 부족해서 프로그램을 컴파일할 수 없는 경우에는 -1을 출력한다.

프로그램을 컴파일할 수 있다면 첫 번째 줄에 명령어를 최소한으로 사용해 `<value>`를 계산하는 프로그램의 명령어 수를 출력한다. 다음 줄부터 프로그램을 구성하는 명령어들을 한 줄에 하나씩 순서대로 출력한다. 한 명령어의 모든 토큰 사이에는 공백이 하나씩 들어간다.

여러 가지가 답으로 가능한 경우 아무거나 하나 출력한다.

입출력 예시

표준 입력(stdin)	표준 출력(stdout)
7 71(72(41,42),73(43,44))	7 41 >> 3 42 >> 4 43 >> 5 44 >> 6 1 = 72 | 3 4 2 = 73 | 5 6 0 = 71 | 1 2
3 71(72(41,42),73(43,44))	9 43 >> 2 44 >> 0 1 = 73 | 2 0 59 << 1 41 >> 2 42 >> 0 1 = 72 | 2 0 59 >> 2 0 = 71 | 1 2
2 71(72(41,42),73(43,44))	-1