

License Key Generator (keygen)

Edoardo just found a beautiful 4D rendering software. Unfortunately, this software is not free, and a valid license key must be entered to use it. Edoardo, however, reverse engineered the program and found the function that checks the key!

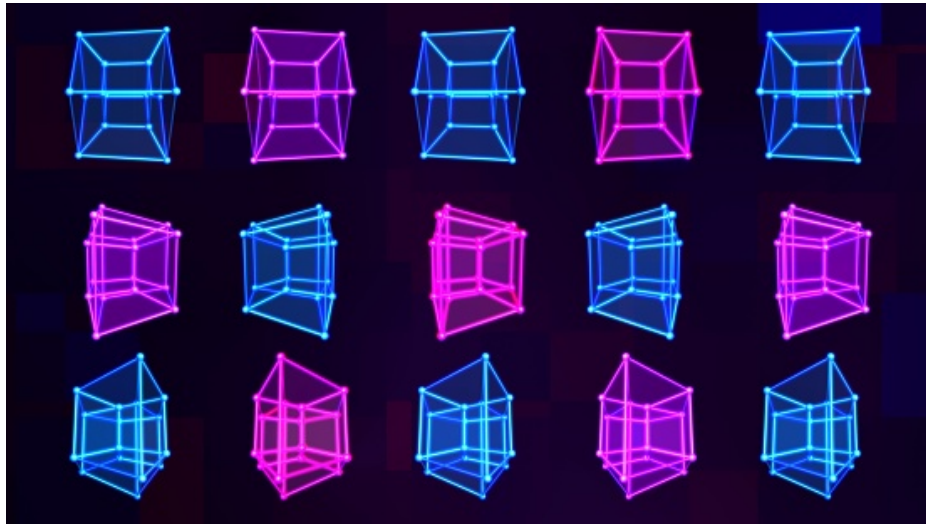


Figure 1: Edoardo illegally using the software.

The function takes K boolean variables, named as the first K lowercase letters of the English alphabet (i.e. “a”, “b”, etc...). These variables are combined in a boolean formula represented by a string S .

The formula is a **XOR** of multiple clauses, where each clause is the **AND** of some literals: a variable (positive literal) or its negation (negative literal). In order to generate a valid license, Edoardo needs to find a way to assign each variable 0 or 1 such that the given formula evaluates to 1. Edoardo wants to generate as many valid licenses as possible to sell them on the dark web. Help Edoardo find the number of different licenses he can sell!

Among the attachments of this task you may find a template file `keygen.*` with a sample incomplete implementation.

Input

The first line contains the only integer T . The description of T testcases follow. For each testcase, the first line contains the only integer K , and the second line contains the string S .

Output






You need to write T lines, where each line contains the number of different valid licenses (i.e., assignments to variables that satisfy the given formula).

Constraints

- $1 \leq T \leq 20$.
- $1 \leq K \leq 15$.
- $1 \leq |S| \leq 250\,000$, where $|S|$ is the length of S .
- S is a concatenation of clauses.
- Clauses are separated by a XOR: “^”.
- Each clause is bounded by round brackets: “(” and “)”.
- Each clause is a concatenation of literals separated by AND: “&”.
- Each clause contains at least 1 literal.
- A literal is just a variable or a variable preceded by a NOT: “!”.

Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

- **Subtask 1** (0 points) Examples.

- **Subtask 2** (20 points) There are no negative literals and $1 \leq K \leq 10$.

- **Subtask 3** (30 points) There are no negative literals.

- **Subtask 4** (20 points) $1 \leq K \leq 12$.

- **Subtask 5** (30 points) No additional limitations.


Examples

input	output
3 4 (c&d)^(!a&b&!c)^(b) 4 (!c) 4 (!c&!d)^(!a&c&!d)^(a&!b&c)	6 8 8
3 5 (d)^(!b&e) 5 (b&d)^(b&!e)^(c&!d&e) 5 (b&!c&d&e)^(!a&d)^(!b&c)^(c&d&!e)	16 12 12

Explanation

In the **first testcase** of **first sample case** there are 6 possible valid combinations:

- a=0, b=1, c=1, d=0.
- a=1, b=1, c=1, d=0.
- a=1, b=1, c=0, d=1.
- a=0, b=0, c=1, d=1.
- a=1, b=0, c=1, d=1.

In the **second testcase** of **first sample case** there are 8 possible valid combinations:

- a=0, b=0, c=0, d=0.
- a=1, b=0, c=0, d=0.
- a=0, b=1, c=0, d=0.
- a=1, b=1, c=0, d=0.
- a=0, b=0, c=0, d=1.
- a=1, b=0, c=0, d=1.
- a=0, b=1, c=0, d=1.
- a=1, b=1, c=0, d=1.

In the **third testcase** of **first sample case** there are 8 possible valid combinations:

- a=0, b=0, c=0, d=0.
- a=1, b=0, c=0, d=0.
- a=0, b=1, c=0, d=0.
- a=1, b=1, c=0, d=0.
- a=0, b=0, c=1, d=0.
- a=1, b=0, c=1, d=0.
- a=0, b=1, c=1, d=0.
- a=1, b=0, c=1, d=1.