Problem A. The $n^2 + 1$ Conjecture

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 megabytes

While walking around the university, waiting for the olympiad to begin, you accidentally came across an open door to one of the classrooms.

It turned out to be a laboratory for multiplying and dividing positive integers.

On one of the laboratory boards, the following hypothesis was written, left by one of the lab assistants:

Take any positive integer n.

While the given number is not equal to one, perform the following steps:

1. If n is even, divide the number by 2.
2. If n is odd, square the number and add one.

I know for which numbers this process terminates in a finite number of steps. Do you?

So you started wondering how, for a given number n, to determine whether it can be transformed into one in a finite number of steps.

Input

The only line contains one integer n ($1 \le n \le 10^9$) — the number for which it is required to determine whether it is transformed into one in a finite number of steps of the described process.

Output

Print one line: "YES" if the given number n is transformed into one in a finite number of steps, and "NO" otherwise.

It is guaranteed that the answer always exists and is unique.

Examples

standard input	standard output
1	YES
17	NO
4	YES

Note

Example 1. For $n = 1$, the number is already equal to one, so no steps of the process need to be performed. The answer is "YES".

Example 2. For $n = 17$, the process begins as follows: $17 \to 290 \to 145 \to 21026 \to \ldots$ It can be proved that one will never be obtained for this sequence, and the answer — "NO".

Example 3. For $n = 4$, the process looks like this: $4 \to 2 \to 1$. One is obtained in two steps. The answer is "YES".

Problem B. Pizza After the Contest

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 megabytes

After five hours of intense competition, the participants deserved some rest and pizza! The organizers only need to determine how many pizzas to order so that nobody is left out.

There were N teams in the contest, and each team had exactly three people, so there were $3N$ participants in total. The pizzeria offers pizzas of two sizes:

- diameter 30 centimeters, cut into 6 equal slices;
- diameter 40 centimeters, cut into 8 equal slices.

All pizzas are round and have the same thickness, so the amount of pizza received by a participant is determined by the total area of the slices given to them.

Each participant must receive at least one slice. At the same time, **all participants must receive the same amount of pizza by area**. One participant may be given slices from different pizzas, including pizzas of different sizes. Slices cannot be cut further. Some slices may be left undistributed.

The organizers want to order the **minimum total number of pizzas**. Among all ways to order and distribute pizzas with that number of pizzas, they choose the one with the **minimum total area of undistributed slices**.

Determine how many pizzas of each size should be ordered.

Input

The only line contains one integer N ($1 \le N \le 10^9$) — the number of teams. Each team has exactly three people, so there are $3N$ participants in total.

Output

Print two non-negative integers separated by a space — the number of pizzas with diameter 30 centimeters and the number of pizzas with diameter 40 centimeters in the optimal order.

An answer always exists. If there are several optimal answers (with the minimum total number of pizzas and the minimum total area of undistributed slices), print any of them.

Note that the number of ordered pizzas should be minimized first. The total area of undistributed slices is minimized among the options with the same number of ordered pizzas.

Examples

standard input	standard output
1	1 0
3	2 0
5	0 2

Note

Let the area of one slice of a 30 cm pizza be 37.5π (the area of a circle of radius 15, divided by 6), and the area of one slice of a 40 cm pizza be 50π (the area of a circle of radius 20, divided by 8).

Example 1. For $N = 1$, there are 3 people. If one 30 cm pizza is ordered, it is cut into 6 slices, and each of the 3 participants can be given 2 slices — all participants receive the same area $2 \cdot 37.5\pi = 75\pi$, and all

6 slices are distributed without leftovers. Fewer than one pizza cannot be ordered, so this is the optimal order. Answer: `1 0`.

Example 2. For $N = 3$, there are 9 people. Consider an order of two pizzas:

- two 30 cm pizzas give 12 slices; each of the 9 participants can be given 1 slice of area 37.5π, and 3 slices will remain undistributed — the total area of the remainder is $3 \cdot 37.5\pi = 112.5\pi$;
- two 40 cm pizzas give 16 slices; each of the 9 participants can be given 1 slice of area 50π, and 7 slices will remain undistributed — the total area of the remainder is $7 \cdot 50\pi = 350\pi$.

It is impossible to order fewer than two pizzas: one pizza of either size gives at most 8 slices, but at least 9 slices are needed (one per participant), so the minimum total number of pizzas is two. Among orders of two pizzas, the option with two 30 cm pizzas gives a smaller area of undistributed slices ($112.5\pi < 350\pi$), so it is optimal. Answer: `2 0`.

Example 3. For $N = 5$, there are 15 people. By ordering two 40 cm pizzas, we get 16 slices; each of the 15 participants receives 1 slice of area 50π, and 1 slice remains undistributed. It can be checked that among all orders with the minimum total number of pizzas, this option gives the smallest remainder area. Answer: `0 2`.

Problem C. Digital MEGAROOT

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	256 megabytes

You have probably encountered the notion of a "digital root": for a number, you take the sum of its digits, then the sum of the digits of the obtained number, and so on, until only one digit remains. Today, the author of the problem suggests moving to a new level — the "digital MEGAROOT".

Let $S(x)$ for a digit string x be the sum of all digits contained in x.

Let N_0 be the starting number, written without leading zeros as a string of K digits. For every pair of integers L, R ($1 \le L \le R \le K$), consider the substring of digits of the number N_0 from position L to position R inclusive and compute S of that substring (that is, the sum of the digits in it). The number N_1 is defined as the sum of the values $S(N_0[L..R])$ over all pairs (L, R), $1 \le L \le R \le K$.

Then the process is repeated: from N_1, by the same rule (all substrings of its digits are considered and their digit sums are added), N_2 is built; from N_2, N_3 is built; and so on — an infinite sequence of numbers $N_0, N_1, N_2, N_3, \ldots$ is obtained.

The digital MEGAROOT of the number N_0 is the minimum number among the entire infinite sequence $N_0, N_1, N_2, \ldots$. It can be shown that for any starting number N_0, this value is finite.

However, something is still missing... The author will perform Q update queries on the number N (which always remains a string of K digits without leading zeros). Each query changes exactly one digit of the number N at a given position; **all changes are preserved and affect all subsequent queries**.

You need to output the digital MEGAROOT of the initial number N, as well as the digital MEGAROOT after each of the Q update queries.

Input

The first line contains one integer K ($1 \le K \le 2 \cdot 10^5$) — the number of digits in the number N.

The second line contains K digits forming the initial number N (without leading zeros).

The third line contains one integer Q ($1 \le Q \le 2 \cdot 10^5$) — the number of update queries.

The fourth line contains Q integers $P_1, P_2, \ldots, P_Q$ ($1 \le P_i \le K$) — the positions of the digit changed in the i-th query.

The fifth line contains Q integers $D_1, D_2, \ldots, D_Q$ ($0 \le D_i \le 9$) — the new value of the digit in the i-th query: in the i-th query, the digit at position P_i is replaced by D_i.

It is guaranteed that if $P_i = 1$, then $D_i \ge 1$ (that is, the number N never gets a leading zero).

Output

Print $Q + 1$ integers $R_0, R_1, \ldots, R_Q$ ($0 \le R_i \le 10^{18}$) in one line separated by spaces: R_0 is the digital MEGAROOT of the number N before any queries, and R_i for $i \ge 1$ is the digital MEGAROOT of the number N after applying the first i update queries.

It is guaranteed that the answer always exists and is unique; there is no need to output any special value in case of no answer.

Example

standard input	standard output
5 14974 4 1 2 1 4 3 7 7 0	6 18 8 4 18

Note

In the first example, $K = 5$, $N_0 = 14974$. Let us compute its digital MEGAROOT:

- $N_0 = 14974$:
  - substrings of length 1: $1 + 4 + 9 + 7 + 4 = 25$
  - substrings of length 2: $(1 + 4) + (4 + 9) + (9 + 7) + (7 + 4) = 45$
  - substrings of length 3: $(1 + 4 + 9) + (4 + 9 + 7) + (9 + 7 + 4) = 54$
  - substrings of length 4: $(1 + 4 + 9 + 7) + (4 + 9 + 7 + 4) = 45$
  - substring of length 5: $1 + 4 + 9 + 7 + 4 = 25$
  - $N_1 = 25 + 45 + 54 + 45 + 25 = 194$
- $N_1 = 194$:
  - substrings of length 1: $1 + 9 + 4 = 14$
  - substrings of length 2: $(1 + 9) + (9 + 4) = 23$
  - substring of length 3: $1 + 9 + 4 = 14$
  - $N_2 = 14 + 23 + 14 = 51$
- $N_2 = 51$:
  - substrings of length 1: $5 + 1 = 6$
  - substring of length 2: $5 + 1 = 6$
  - $N_3 = 6 + 6 = 12$
- $N_3 = 12$:
  - substrings of length 1: $1 + 2 = 3$
  - substring of length 2: $1 + 2 = 3$
  - $N_4 = 3 + 3 = 6$
- starting from N_4, all subsequent terms are equal to 6: $N_4 = N_5 = N_6 = \ldots = 6$.

The digital MEGAROOT is the minimum among $14974, 194, 51, 12, 6, 6, 6, \ldots$, that is, 6. This value is output as R_0.

Next, let us see how the number N changes after the update queries (the changes accumulate):

- after the first query ($P_1 = 1$, $D_1 = 3$) the number becomes $N = 34974$; the corresponding sequence is: $34974, 204, 18, 18, \ldots$; the MEGAROOT is 18;
- after the second query ($P_2 = 2$, $D_2 = 7$) the number becomes $N = 37974$; the sequence is: $37974, 228, 38, 22, 8, \ldots$; the MEGAROOT is 8;

- after the third query ($P_3 = 1$, $D_3 = 7$) the number becomes $N = 77974$; the sequence is: $77974, 248, 46, 20, 4, \ldots$; the MEGAROOT is 4;
- after the fourth query ($P_4 = 4$, $D_4 = 0$) the number becomes $N = 77904$; the sequence is: $77904, 192, 45, 18, 18, \ldots$; the MEGAROOT is 18.

The final answers are $6, 18, 8, 4, 18$.

Problem D. Infinite August

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 megabytes

Kira loves summer holidays: she can rest, go for walks, and not go to school. But August ends too quickly, and already on September 1 she has to go back to classes.

So Kira came up with her own calendar, in which August lasts forever. After August 31 comes August 32, then August 33, and so on. The holidays continue!

But an important problem arose: Kira's birthday is on December 6. She is not going to skip it, so she needs to find out what day number of August this date will become. For example, if we extend August 2026, Kira's birthday will fall on August 128.

Kira decided that in this way any month can be extended, and any later date can be converted into her calendar: choose a month m and a year y, after which month m of year y becomes infinite — it is followed by "(day of month +1)-th m", then "(day of month +2)-th m", and so on, without moving on to the following months and years.

The first day of month m of year y is day number 1 in Kira's calendar, the next day is number 2, and so on: each next calendar day gets a number one greater than the previous one.

You are given the chosen month m and year y, as well as a date in the usual (Gregorian) calendar: day d, month m_2, year y_2. Determine the day number of this date in Kira's calendar, that is, the number this date gets if month m of year y is extended infinitely.

The Gregorian calendar is used: read the note for details about the number of days in months and leap years.

Input

The first line contains two integers m and y ($1 \le m \le 12,\ 1 \le y \le 9999$) — the number of the month and the number of the year Kira chose to extend.

The second line contains three integers d, m_2, and y_2 ($1 \le m_2 \le 12,\ 1 \le y_2 \le 9999$) — the day, month, and year of the date for which the day number in Kira's calendar must be determined.

It is guaranteed that the date d, m_2, y_2 exists in the Gregorian calendar (the rules for determining the number of days in a month and leap years are given in the note) and is not earlier than the first day of month m of year y.

Output

Print one integer — the day number of the date d, m_2, y_2 in Kira's calendar, where the first day of month m of year y has number 1, the next calendar day has number 2, and so on.

It is guaranteed that under the given constraints the answer always exists and is uniquely determined.

Examples

standard input	standard output
8 2026 6 12 2026	128
8 2026 1 8 2026	1
2 2024 1 3 2024	30
12 3456 11 1 3457	42
6 7 8 9 1023	371186

Note

The Gregorian calendar is used.

A year is a leap year if its number is divisible by 400 or divisible by 4 but not divisible by 100.

In a leap year, February has 29 days; in other years — 28 days. April, June, September, and November have 30 days, and the remaining months have 31 days.

Note — although historically the Gregorian calendar appeared only in the 16th century, in this problem its leap-year rules apply to all valid year values.

In the first example, August 2026 is chosen, and the given date is December 6, 2026. From August 1 to December 1, 2026, August (31 days), September (30 days), October (31 days), and November (30 days) pass, for a total of $31+30+31+30=122$ days, so December 1 is August 123. Then December 6 becomes August $123+5=128$, which matches the answer 128.

In the second example, the given date is August 1, 2026, that is, exactly the first day of the chosen month. By definition, it gets number 1 in Kira's calendar, which matches the answer 1.

In the third example, February 2024 is chosen, and the given date is March 1, 2024. The year 2024 is divisible by 4 and not divisible by 100, so it is a leap year, and February 2024 has 29 days. Therefore, March 1 comes the day after February 29, that is, it becomes February 30 in Kira's calendar, which matches the answer 30.

In the fourth example, December 3456 is chosen, and the given date is January 11, 3457. The answer is $42=31+11$, which is also the answer to the "Ultimate Question of Life, the Universe, and Everything".

You can use the fifth test case to check your solution on large time intervals.

Problem E. Unusual Fractions

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 megabytes

Martians write fractions as follows: the notation $\{a\}/\{b\}$ denotes the fraction with numerator "a" and denominator "b". Martians do not allow spaces in fraction notation, and curly braces are used only to write fractions.

The following rules are used to convert the Martian notation into the ordinary one:

- the fraction bar is made of "minus" signs and is placed so that the numerator is entirely above it and the denominator is entirely below it;
- the numerator and the denominator are placed so that the width of the fraction is the minimum possible;
- then the fraction bar is extended by two more characters: one "minus" on the left and one on the right;
- the fraction bars of adjacent fractions are at the same height; two fractions are called adjacent if there are no other fractions between them (in the Martian notation);
- the characters written between adjacent fractions are placed at the same height as their fraction bars.

For example, the notation

```
{m}/{{n}/{q}}+{{x}/{y}}/{z}
```

is converted into the ordinary one as follows:

```
      x
     ---
 m    y
-----+-----
 n    z
---
 q
```

Its width is 11 characters, and its height is 7 lines.

Given a Martian notation, determine the width and the height of its ordinary notation. You do not need to build the ordinary notation itself.

Input

The only line contains a Martian notation S ($7 \le |S| \le 250$). The string consists of printable ASCII characters except space.

It is guaranteed that S is a correct Martian notation (the definition is given in the notes).

Output

Print two positive integers separated by a space — the width of the ordinary notation in characters and its height in lines.

Examples

standard input	standard output
{m}/{{n}/{q}}+{{x}/{y}}/{z}	11 7
{2m}/{{n}/{q}}+{{x}/{y}}/{z+1}	11 7
{a-+-b/cc}/{3}	10 3

Note

Let a character be any printable character except space and curly braces (including "/"). A correct Martian notation is defined as follows:

- a fraction is a notation $\{A\}/\{B\}$, where the numerator A and the denominator B are expressions;
- an expression is a non-empty sequence of characters and fractions in which there is at least one character between any two adjacent fractions;
- the whole string S is an expression that begins and ends with a fraction;
- the sequence of characters between adjacent fractions cannot consist of a single character "/": notations like $\{a\}/\{b\}/\{c\}$ or $\{a\}/\{b\}/\{c\}/\{d\}$ never occur.

The character "/" outside the construction }/{ is an ordinary character.

The first example is explained in the statement. In the second example, the ordinary notation is:

```
      x
     ---
2m    y
-----+-----
 n   z+1
---
 q
```

The fraction $\{n\}/\{q\}$ has width $1 + 2 = 3$, so the left fraction has width $\max(2, 3) + 2 = 5$; the right one also has width $\max(3, 3) + 2 = 5$; together with the "+" sign, the width is $5 + 1 + 5 = 11$. There are 3 lines above the fraction bars (because of $\{x\}/\{y\}$) and also 3 below them (because of $\{n\}/\{q\}$), so the height is $3 + 1 + 3 = 7$.

In the third example, "/" inside the numerator is an ordinary character: the numerator "a-+-b/cc" has width 8, so the width of the fraction is $8 + 2 = 10$, and the height is 3.

Problem F. Circular Minesweeper

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 megabytes

Interactive problem

You are given a circle of $L = 10^9$ cells, numbered around the circle from 1 to L (cells 1 and L are adjacent). There are mines in n distinct cells; **in each test, the mine placement is chosen randomly and uniformly** among all sets of n cells.

You may probe cells. If the probed cell contains a mine, it explodes. In response to a probe of cell i, you learn whether there was a mine there, and the distance d from i to **some** remaining unexploded mine. Which mine is chosen is determined by an **unknown rule** — it is not necessarily the nearest mine. The distance between cells a and b is $\min(|a - b|, L - |a - b|)$.

After making at most 30 probes, specify a cell that definitely contains an unexploded mine.

Input

The first line contains an integer n ($2 \le n \le 2 \cdot 10^5$) — the number of mines.

Output

See the section "Interaction protocol".

Interaction Protocol

First read the number n.

To make a probe, output "? i" ($1 \le i \le L$) and read the response:

- "YES d", if cell i contained a mine (and it exploded),
- "NO d", if there was no mine.

Here d is the distance from i to some unexploded mine; the mine exploded by this probe is not taken into account.

You can make at most 30 probes. To give the final answer, output "! i" and terminate the program: at that moment, cell i must contain an unexploded mine. The final answer does not count toward the probe limit.

Your program will receive the verdict "Wrong answer" in the following cases:

- the output format is invalid;
- the number of available probes is exceeded;
- all mines have exploded;
- the output answer is incorrect (the cell does not contain an unexploded mine).

After each query, you must flush the output buffer, otherwise your program will receive the verdict "Idleness Limit Exceeded":

- C++: `std::endl` when using `iostream` (`cout << ... << endl`), `fflush(stdout)` after output using `stdio` (`printf`);

- Java: `System.out.println(...)`, and when using another stream (`PrintWriter`, `BufferedWriter`) — call its `flush()` after each line;
- Python: `print(..., flush=True)`, and read the interactor's responses using `input()`.

Example

interaction log

```
  > 4
< ? 1
  > YES 4
< ? 1
  > NO 11
< ? 13
  > NO 1
< ! 12
```

Note

Example. There are mines in cells 1, 5, 12, and 100 on the circle (the program does not know these positions; it only sees $n = 4$).

- For the query "? 1", the interactor responds "YES 4": cell 1 contained a mine and it exploded (the distances to mines 5, 12, and 100 are 4, 11, and 99 respectively; the interactor chose mine 5 and returned the distance to it);
- For the repeated query "? 1", the interactor responds "NO 11": there is no mine in cell 1 anymore, and the distance to mine 12, which the interactor chose, is 11;
- For the query "? 13", the interactor responds "NO 1": there is no mine in cell 13, the interactor chose mine 12 and returned the distance to it;
- the program outputs "! 12" and terminates — the answer is correct, since this cell contains an unexploded mine. In total, 3 probes were made out of the allowed 30 (the query "! 12" does not count toward this limit).

This example shows that the distance d in the interactor's response is the distance to **some** (not necessarily the nearest) unexploded mine, and that you cannot output an already exploded mine as the answer.

The strategy for finding an unexploded mine used in the example is not necessarily optimal.

Problem G. Are ya winning, son?

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	256 megabytes

The tournament features 2^N teams. The tournament lasts for K rounds.

Before the first round, the teams are placed in the table in an arbitrary order. In each round, the teams are split into pairs according to the current table: the 1-st team plays the 2-nd, the 3-rd — the 4-th, and so on. In each pair, one team wins and the other loses — there are no draws.

After each round, the table is resorted according to the following rules:

- a team with more wins is placed higher;
- if the number of wins is the same, the team that won the last played match is placed higher;
- if that is also the same — the results of the second-to-last match are compared, and so on, all the way back to the first round.

We define a team's **statistics** as the binary string $b_1 b_2 \dots b_K$, where $b_i = 1$ if the team won in the i-th round, and $b_i = 0$ otherwise. We will encode the statistics as the number $\sum_{i=1}^{K} b_i \cdot 2^{i-1}$: the result of the first round is the **least significant** bit, and the result of the last round is the most significant bit.

The teams are indistinguishable from one another, and the outcomes of the matches are unknown. It can be shown that the outcomes of the matches affect only **which exact team** ends up in a given position of the final table, but **not the statistics** of the team in that position.

You are given Q queries: for each position P_i, determine the statistics of the team that ends up in that position after K rounds.

Input

The first line contains three integers N, K, Q ($1 \le K \le N \le 30$; $1 \le Q \le 10^5$) separated by spaces — the logarithm of the number of teams (there are 2^N teams in total), the number of rounds, and the number of queries.

The second line contains Q integers $P_1, P_2, \dots, P_Q$ ($1 \le P_i \le 2^N$) separated by spaces — positions in the final table after K rounds. Positions are numbered from 1 to 2^N from top to bottom.

Output

Print Q integers $S_1, \dots, S_Q$ in one line separated by spaces. The number S_i ($0 \le S_i < 2^K$) is the statistics of the team standing at position P_i in the final table, encoded as a number: $S_i = \sum_{j=1}^{K} b_j \cdot 2^{j-1}$, where $b_j = 1$ if the team won in the j-th round, and $b_j = 0$ otherwise.

Examples

standard input	standard output
2 2 4 1 2 3 4	3 2 1 0
3 3 8 8 7 6 5 4 3 2 1	0 1 2 4 3 5 6 7
4 2 5 1 4 5 12 16	3 3 2 1 0

Note

Example 1. Consider one possible outcome of the tournament for teams A, B, C, D. In round 1, team A beats B, and team C beats D; the table becomes A, C, B, D (teams A and C each have one win). In round 2, team A beats C, and team B beats D; the teams' statistics are: $A = 11$, $C = 10$, $B = 01$, $D = 00$. The final table is: first A (two wins), then among the teams with one win, B is higher because it won the last match, then C, then D. The statistics at positions $1, 2, 3, 4$ are $11, 01, 10, 00$, that is, the numbers $1 + 2 = 3$, 2, 1, 0 (the first digit is the least significant bit).

For another outcome — B beats A, D beats C in round 1, then D beats B, A beats C in round 2 — the teams at positions $1, 2, 3, 4$ are D, A, B, C with statistics $11, 01, 10, 00$: the teams in the positions are different, but the statistics are the same.

Example 2. For $K = 3$, the statistics in the final table (from position 1 to 8) are ordered as $111, 011, 101, 110, 001, 010, 100, 000$, that is, $7, 6, 5, 3, 4, 2, 1, 0$. For example, both 011 and 101 won the last (third) round, so the second-to-last round is compared: 011 has a win there, while 101 has a loss, so 011 is higher. The statistic 110 lost the last round and is below both of them, despite having the same number of wins. The queries are given in the order $8, 7, \ldots, 1$, and the answers are printed in the same order.

Example 3. For $N = 4$, $K = 2$, there are 16 teams, and there are 4 different statistics. Each appears for exactly 4 consecutive teams: positions 1–4 correspond to statistic 11 (number 3), 5–8 to 01 (2), 9–12 to 10 (1), and 13–16 to 00 (0).

Problem H. Spin the Wheel!

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	256 megabytes

You are given an array A of N non-negative integers.

In one operation, you may choose any element A_i of the array and cyclically shift its digits by an arbitrary amount. When shifting, leading zeros are allowed in the resulting representation — they are considered part of the representation, but **do not affect the final numeric value**.

For example, from the number 10032 (digits $1, 0, 0, 3, 2$), **in one operation** you can obtain the following representations with different shifts: 10032, 00321, 03210, 32100, 21003.

When computing the sum, leading zeros obviously do not matter (that is, 00321 is equal to 321). Note that the number 0 does not contain leading zeros.

You are allowed to apply the operation to **at most** K elements of the array.

You need to choose which elements to change (at most K of them) so that the **sum of all elements of the resulting array is as small as possible**.

Input

The first line contains two integers N and K ($1 \le K \le N \le 2 \cdot 10^5$) — the size of the array and the maximum number of operations.

The second line contains N integers $A_1, A_2, \ldots, A_N$ separated by spaces ($0 \le A_i \le 10^9$) — the elements of the array. The numbers A_i are written without leading zeros.

Output

Print N integers $B_1, B_2, \ldots, B_N$ separated by spaces in a single line — the array obtained after applying the operations, with the minimum possible sum $\sum_{i=1}^{N} B_i$.

Each B_i must be the value of some cyclic shift of the number A_i ($B_i = A_i$ is allowed).

At the same time, the number of indices i for which B_i is obtained by a non-zero shift (that is, B_i is not equal to the representation of A_i without shifting) must not exceed K.

Each B_i is printed as an ordinary integer without leading zeros: for example, for $A_i = 10032$, a shift producing the representation 00321 is printed as 321.

If there are several arrays B with the minimum possible sum, you may output any of them.

Examples

standard input	standard output
5 2 10032 203 7 100 980	321 203 7 100 98
3 2 1003 21 7	31 12 7

Note

Example 1.

Original array: $10032, 203, 7, 100, 980$, and at most $K = 2$ operations are allowed.

In the shown answer, two elements are shifted: 10032 and 980.

The representation of 10032 (digits $1, 0, 0, 3, 2$) is shifted so that the first digit 1 is moved to the end: the representation 00321 is obtained, whose value is 321.

The representation of 980 (digits $9, 8, 0$) is shifted so that the first two digits $9, 8$ are moved to the end: the representation 098 is obtained, whose value is 98.

The remaining elements 203, 7, 100 are not shifted and remain equal to 203, 7, and 100 respectively.

The resulting array is $321, 203, 7, 100, 98$, and its sum is $321 + 203 + 7 + 100 + 98 = 729$.

There are other ways to apply two operations (for example, to 203 and 100, or to 203 and 10032), but the sum 729 is the minimum possible among all ways.

Example 2.

Original array: $1003, 21, 7$, and at most $K = 2$ operations are allowed.

In the shown answer, the elements 1003 and 21 are shifted.

The representation of 1003 (digits $1, 0, 0, 3$) is shifted so that the first digit 1 is moved to the end: the representation 0031 is obtained, whose value is 31.

The representation of 21 (digits $2, 1$) is shifted so that the first digit 2 is moved to the end: the representation 12 is obtained, whose value is 12.

The resulting array is $31, 12, 7$.

Note: the number 1003 cannot be turned into 13 from 31, because in fact you get 0031, not just 31.

Problem I. The Crooked Walk of Hilbert

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	256 megabytes

The Hilbert curve of order n is a polyline that visits all cells of a $2^n \times 2^n$ square, visiting each cell exactly once and moving each time to an adjacent cell by side. It is constructed recursively.

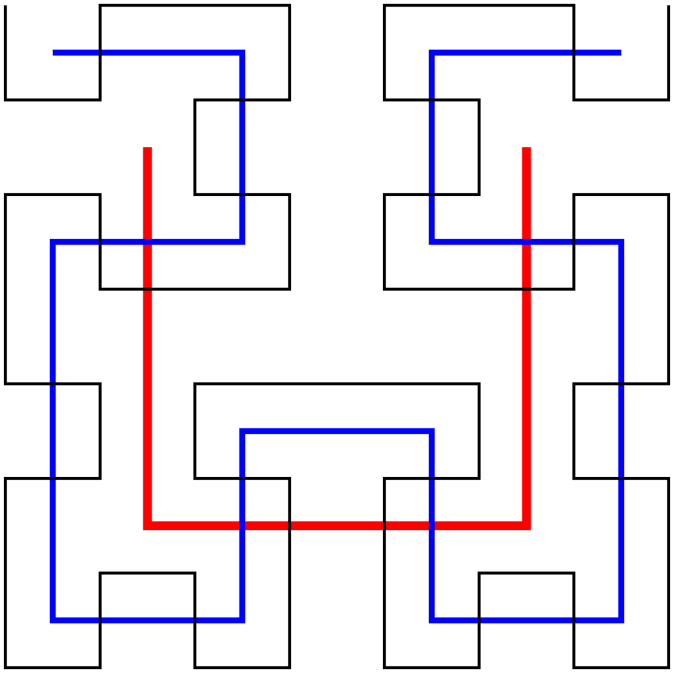
Order 1. The 2×2 square is traversed in the shape of the letter "U": from the upper-left cell down, then right, then up. The curve starts in the upper-left cell and ends in the upper-right one.

Order $n > 1$. The $2^n \times 2^n$ square is divided into four $2^{n-1} \times 2^{n-1}$ squares, and each one contains a copy of the order $n - 1$ curve:

- in the lower-left and lower-right squares — copies unchanged;
- in the upper-left square — a copy rotated by 90° counterclockwise;
- in the upper-right square — a copy rotated by 90° clockwise.

Then the copies are connected into one polyline in the order: upper-left → lower-left → lower-right → upper-right square. As with the order 1 curve, the resulting curve starts in the upper-left cell and ends in the upper-right one.

The figure shows Hilbert curves of orders 1, 2, and 3, overlaid on each other.



Number the cells of the square from 1 to 4^n in the order of traversal of the Hilbert curve of order n.

Input

The only line contains one integer N ($1 \le N \le 9$) — the exponent of the square side; the side length is 2^N.

Output

Print 2^N lines with 2^N integers separated by spaces in each — the matrix M_N of cell numbers in the order of traversal of the Hilbert curve: the j-th number in the i-th line is the number of cell (i, j).

Each integer from 1 to 4^N must appear in the output matrix exactly once, and cell $(1, 1)$ must contain 1. It can be shown that with this definition of the curve, the answer always exists and is unique.

Examples

standard input	standard output
1	1 4 2 3
2	1 2 15 16 4 3 14 13 5 8 9 12 6 7 10 11
3	1 4 5 6 59 60 61 64 2 3 8 7 58 57 62 63 15 14 9 10 55 56 51 50 16 13 12 11 54 53 52 49 17 18 31 32 33 34 47 48 20 19 30 29 36 35 46 45 21 24 25 28 37 40 41 44 22 23 26 27 38 39 42 43

Note

In the first example, $N = 1$, the square has size 2×2, and the answer is the matrix

$$\begin{pmatrix} 1 & 4 \\ 2 & 3 \end{pmatrix}.$$

This is exactly matrix M_1 from the statement: the traversal starts in cell $(1, 1)$ with number 1, then moves to cell $(2, 1)$ with number 2 (a step down), then to cell $(2, 2)$ with number 3 (a step right), and ends in cell $(1, 2)$ with number 4 (a step up). Thus the traversal order is "down, right, up", it starts in the upper-left corner of the square and ends in its upper-right corner, as required by the statement.

In the second example, $N = 2$, the square has size 4×4, and the answer is the matrix

$$\begin{pmatrix} 1 & 2 & 15 & 16 \\ 4 & 3 & 14 & 13 \\ 5 & 8 & 9 & 12 \\ 6 & 7 & 10 & 11 \end{pmatrix}.$$

Here the 4×4 square is divided into four 2×2 quadrants, which are traversed in the order described in the statement:

- the upper-left quadrant (cells $(1, 1)$–$(2, 2)$) is traversed in the order "right, down, left";
- the lower-left quadrant (cells $(3, 1)$–$(4, 2)$) is traversed in the order "down, right, up";
- the lower-right quadrant (cells $(3, 3)$–$(4, 4)$) is traversed in the order "down, right, up";
- the upper-right quadrant (cells $(1, 3)$–$(2, 4)$) is traversed in the order "left, up, right".

The traversal starts in cell $(1, 1)$ with number 1 and ends in cell $(1, 4)$ with number $16 = 4^2$.

In the third example, $N = 3$, the square has size 8×8; the answer is constructed by the same recursive rule as in the previous examples, but for the next order.