EXPRESSIONS (POL)

Let X be the smallest set defined as follows

- an empty string belongs to X,
- if A, B belong to X then both, (A) and AB, belong to X.

The elements of X are called *correctly built parenthesis expressions*.
The following strings are correctly built parenthesis expressions:

```
()(())()
((()()))
```

The expressions below are not correctly built parenthesis expressions:

```
(()))(()
())(()
```

Let E be a correctly built parenthesis expression.
The length of E is the number of single parenthesis in E.
The depth D(E) of E is defined as follows:

```
      ⎧  0                 if E is empty
D(E)= ⎨  D(A)+1            if E = (A), and A is in X
      ⎩  max(D(A),D(B))    if E = AB, and A, B are in X
```

What is the number of correctly built parenthesis expressions of length **n** and depth **d**, for given positive integers n and d?

Task

Write a program which

- reads two integers n and d from the text file PAR.IN;
- computes the number of correctly built parenthesis expressions of length n and depth d;
- writes the result into the text file PAR.OUT

Input data

The only line of the input file PAR.IN contains two integers n and d separated by a single space character, $2 \le n \le 38$, $1 \le d \le 19$.

Output data

The only line of the output file PAR.OUT should contain a single integer - the number of correctly built parenthesis expressions of length n and depth d.

Example

Input data (file PAR.IN)	Output data (file PAR.OUT)
6 2	3

There are exactly three correctly built parenthesis expressions of length 6 and depth 2:

```
(())()
()(())
(()())
```

LAZY COURSE SELECTION (USA)

When students enroll in a university, they are usually confused by the complexity of the rules for earning a degree. A degree in Computer Science, for instance, has many requirements, and each requirement can be satisfied by many courses. To complicate things further, any single course may satisfy several of the requirements at once. These rules are presented to students when they enter, and the students are left to pick the courses which will earn them a degree.

Since most good programmers are lazy, an entering Computer Science student will want to take as few courses as possible. You should write a program to help a lazy Computer Science freshman select courses to earn a degree.

Courses are numbered by consecutive integers from 1 to C (total number of available courses). Given a description of which courses satisfy which requirements, your program should recommend a set of courses so that every requirement for the degree is fulfilled by at least one recommended course. If there is any requirement left unfulfilled, then the poor student does not get a degree, and the solution is considered invalid and scores zero points. If a solution is determined to be valid, then points are awarded based on the quality of the solution: the fewer courses the student must take, the more points the solution earns.

Note that you will almost certainly not be able to find the optimum answer for some of the test cases within the given run time limit. You should try to get as good a solution as you can in time, making sure your program outputs it and exits before time runs out.

Input data

The first line of the file LAZY.IN contains two space separated integers, C and R. C is the number of courses available to the student ($1 \le C \le 200$), and R is the number of requirements for a degree ($1 \le R \le 100$).

The rest of the file contains C lines, one for each course (the first is for course number 1, up through the last for course number C). Each line has R space separated integers (the first is for requirement 1, the last for requirement R).

The r'th integer on the files c+1'st line is 1 if course c fulfills requirement r, and 0 otherwise.

Output data

Output file LAZY.OUT should contain two lines. The first line should contain a single integer N - the recommended number of courses for the student to take. The second line should contain N integers separated by single space character indicating the N recommended course numbers. Course numbers may be in any order.

Example

Input data(file LAZY.IN)	Output data(file LAZY.OUT)
4 3 0 1 0 1 1 0 0 1 1 0 0 1	2 2 3

There are 4 available courses and 3 requirements: course 1 fulfills only requirement 2, course 2 fulfills requirements 1 and 2, course 3 fulfills requirements 2 and 3 and course 4 fulfills only requirement 3.
Only two courses (#2 and #3) are recommended.

STRING MATCHING (EST)

You are given two collections of strings, collection *A* and collection *B*. Your task is to find out if there exists a non-empty string *S* that is expressible as a concatenation of members of collection *A* as well as a concatenation of members of collection *B*. If there is such string, you must find the shortest one. A member of collection may occur multiple times in a concatenation.

Input data

On the first line of file `MATCH.IN` is integer N ($0 \le N \le 100$), the number of strings in collection *A*. On the next N lines are the members of collection *A*, each on separate line. On the next line is integer M ($0 \le M \le 100$), the number of strings in collection *B*. On the next M lines are the members of collection *B*, each on separate line. All collection members consist of capital Latin letters and have length between 1 and 50 (inclusive).

Output data

Your program must write it's output to file `MATCH.OUT`. If there is no string *S*, the program must write 0 (zero) on the first and only line of the file. Otherwise, the first line must contain the length of the shortest possible *S*. The next line must contain the representation of *S* as concatenation of strings from collection *A*. The strings must be output in the order of concatenation and separated by '+' (plus) signs. The next line must contain the representation of the string as concatenation of strings from collection *B* in the same format as for strings from collection *A*. If different shortest strings are possible, you must output representation for one of them. If different representations from one collection for shortest string are possible, you must output one of them.

Evaluation

Only absolutely correct answers are accepted - if the reported string is not the shortest possible, the program scores 0 for given test. A solution that fails all tests where string *S* exists will score 0 for the whole problem.

Example 1

Input data(file `MATCH.IN`)	Output data (file `MATCH.OUT`)
1 AB 2 A BA	0

Example 2

Input data(file `MATCH.IN`)	Output data (file `MATCH.OUT`)
2 ABC CB 2 CBA BC	5 CB+ABC CBA+BC