PROBLEM 1 – DICE

The people of ACMamia have a tradition of rolling a six sided die to determine who will choose where to dine that evening.

One person rolls the die, the other calls “odd” or “even”. If the caller guesses correctly, then he or she gets to choose the restaurant; otherwise, the person throwing does. Hopefully, it is obvious that “odd” wins when 1, 3 or 5 is rolled, and “even” with 2, 4 or 6!

ACMamians also have a tradition of repeating the same call (either “odd” or “even”) several times in succession.

Your task is to check the recorded rolls and determine how many times each person wins.

INPUT FORMAT

Input for this problem consists of a sequence of one or more scenarios. Each scenario contains 3 lines.

- The first line contains, in order, the names of two people, and one of the words “odd” or “even” (in lowercase), separated by a space. The first named person will always throw, and the second named person will always call either “odd” or “even”, as indicated. In ACMamia, a name is a non-empty sequence of up to 20 (inclusive) letters (any of which may be uppercase or lowercase).
- The second line will be an integer, N, $1 \le N \le 255$, representing the number of die throws.
- The third line contains N integers, each between 1 and 6 (inclusive), separated by single spaces, representing die throws.

The input will be terminated by a line consisting of three hash signs (#), separated by single spaces. This line should not be processed.

SAMPLE INPUT:

```
Bill Susan even
8
1 6 5 3 4 2 5 5
Sarah Tony odd
15
2 4 5 4 3 6 1 2 5 4 3 1 2 5 6
# # #
```

OUTPUT FORMAT

Output will be a sequence of lines, one for each input scenario. Each line will contain in order the following items, separated by single spaces: the name of the first person (exactly as it appears in the input), the number of times the first person won, the name of the second person (exactly as it appears in the input), and the number of times the second person won.

SAMPLE OUTPUT:

```
Bill 5 Susan 3
Sarah 8 Tony 7
```

PROBLEM 2 – ABBREVIATIONS

In this part of the world it is quite common to abbreviate words – “Uni” for “University”, and “Bas” for “Basil” are obvious examples.

In this problem, you will be given sets of words. In each set you must decide on the shortest abbreviation that will uniquely match each word within the set. In this context, an abbreviation is one or more adjacent letters from the word, starting with the first letter. So valid abbreviations for “Basil” would be “B”, “Ba”, “Bas”, “Basi” and “Basil”. The last is not strictly an abbreviation, but it is included to allow us to distinguish words such as “wrong” and “wrongly”.

INPUT FORMAT

Input for this problem consists of a sequence of one or more scenarios. Each scenario describes a set of words.

- Each set of words will start with a single integer, *N*, 1 ≤ N ≤ 100, on a line of its own, which is the number of words in the set.
- This will be followed by *N* lines, each consisting of one word. A word consists of a non-empty sequence of up to 100 (inclusive) lowercase letters. Words within a set will be unique, but not necessarily sorted.

The input will be terminated by a line consisting of a zero (`0`). This line should not be processed.

SAMPLE INPUT:

```
4
cat
dog
mouse
horse
3
ant
antelope
anteater
0
```

Output Format

For each input scenario, you must output the scenario number, starting with 1, on a line on its own, followed by one line for each word in the set, in the same order as they appear in the input. The word must be displayed, followed by a space, followed by the shortest abbreviation (as defined above, in lower case) that will uniquely match the word within the set.

Sample Output:

```
1
cat c
dog d
mouse m
horse h
2
ant ant
antelope antel
anteater antea
```

PROBLEM 3 – CONTEST

The MCA Programming Contest differs from the ACM Contest in its scoring. In the MCA, each problem is graded and, if correctly solved, scores a number of points related to its difficulty. The system will not allow a team to resubmit again a solved problem. The team with the most points wins. If teams are tied on points, the tie is broken in favour of the team with the fewer correct solutions. If teams are still tied, the tie is broken in favour of the team with the fewer total submissions. If teams are still tied, the tie is broken in favour of the team with the smaller team number.

Your task is to find out the overall team ranking according to the above rules.

INPUT FORMAT

Input for this problem consists of a sequence of one or more scenarios. Each scenario describes a contest.

- The first line consists of 2 integers separated by a single space, *T* and *P*, $1 \le T \le 10$, $1 \le P \le 10$, giving the number of teams taking part (*T*), and the number of problems used (*P*). Teams are numbered consecutively from 1, problems are lettered consecutively from uppercase A.
- The second line contains *P* integers, separated by single spaces, giving the points score for each problem, in letter order. Each score will be in the range 1 to 100 (inclusive).
- The third line consists of a single integer, *S*, $1 \le S \le 200$, giving the total number of submissions during the contest.
- There then follow *S* lines, each containing details of a single submission. The data for a submission are, in order: the team number, problem letter, followed by uppercase `A` for accepted or uppercase `R` for rejected, all separated by single spaces.

The input will be terminated by a line consisting of two zeros (`0`), separated by a single space. This line should not be processed.

OUTPUT FORMAT

For each scenario, output the scenario number, starting with 1, on a single line. Follow that by the teams and their points, each on a single line, in order of their rankings (as defined by the rules above). For each team, output the team number, followed by a single space, followed by their score.

SAMPLE INPUT:

```
2 2
5 10
2
1 A A
2 B A
2 2
5 10
3
1 B A
2 B A
1 A R
2 3
5 5 10
3
1 A A
1 B A
2 C A
3 3
5 5 5
5
1 A A
1 B R
2 B A
2 C R
3 C A
0 0
```

SAMPLE OUTPUT:

```
1
2 10
1 5
2
2 10
1 10
3
2 10
1 10
4
3 5
1 5
2 5
```

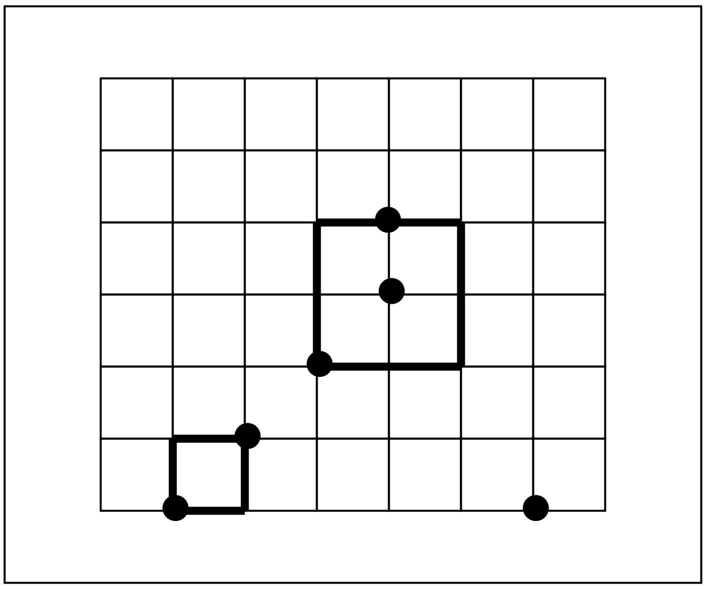
PROBLEM 4 – ORCHARDS

John wants to fence in all his trees, and happily his friend Bill has a fence building company.

Being a good friend, Bill has proposed to build the required fences for free, provided that costs are kept under control. Specifically, Bill agrees to build fences for up to three square paddocks, with sides parallel to the coordinate axes, with the condition that the side of the largest paddock(s) is as small as possible.

All paddocks must be square (have equal length sides) and paddocks must enclose all trees, considering trees as distinct points in the plane. The paddocks may intersect, or overlap on their borders, or even have 0–length sides, and a tree on a fence is considered enclosed.

A paddock may enclose only one tree, in which case its side length is 0. For example, the following diagram shows how three paddocks, of side lengths 1, 2, and 0, can be used to enclose 6 trees.



Your task is to find the minimum side of the largest paddock(s) that Bill needs to build. In the above diagram, the minimum such side is 2.

INPUT FORMAT

Input for this problem consists of a sequence of one or more scenarios. Each scenario is described by several lines.

- The first line contains, in order, the label of the scenario, a number, s, $1 \le s \le 100$, followed by the number of the trees, n, $1 \le n \le 1000$, separated by a single space.
- The first line will be followed by one or more lines containing $2 \times n$ integers, separated by single spaces or newlines, giving, in order, the coordinates of the trees, x before y, $0 \le x, y \le 1000000$. [1]

The input will be terminated by a line consisting of two zero numbers (`0`), separated by a single space. This line should not be processed.

SAMPLE INPUT:

```
1 6
1 0 2 1 3 2 4 3 4 4 6 0
2 5
1 0 2 1 3
2 5 4 6 0
3 1
1 1
4 2
1 0 5 5
0 0
```

OUTPUT FORMAT

Output will be a sequence of lines, one for each input scenario. Each line will contain in order the scenario label, followed by the minimum side of the largest paddock(s) required by the problem, separated by a single space.

SAMPLE OUTPUT:

```
1 2
2 2
3 0
4 0
```

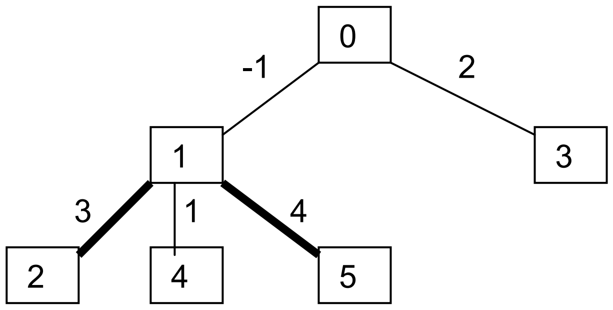
[1] Attention, for this program, input lines may contain up to 2000 characters each!

PROBLEM 5 – PATHS

A developer has designed a subdivision within a city such that all roads connect at intersections in a tree–like design. This is to prevent all petrol–head hooligans from disturbing the residents by not having any road loops for races. Only the entering intersection is connected to the rest of the city. The developer is selling off land alongside roads between adjacent intersections. A real estate agent has produced a book indicating the expected dollar profit (positive, zero or negative) that can be obtained by purchasing the land alongside each road.

Potential buyers want to maximize their profit, but prefer to buy a contiguous stretch of land alongside a simple road chain that connects two intersections of the subdivision. Your task is to write a program to determine the maximum non–negative profit that can be obtained this way, and return 0 if no such profit can be obtained.

As an example, consider the following representation of a subdivision, where road labels represent expected profits. In this scenario, the maximum non–negative profit is 7, and can be obtained alongside the road chain between the intersections #2 and #5:



INPUT FORMAT

Input for this problem consists of a sequence of one or more scenarios. Each scenario contains two or more lines.

- The first line contains an integer n, $1 \le n \le 500000$, indicating the number of intersections, including the entrance intersection, implicitly labelled 0.
- This is then followed by one or more lines, containing n-1 pairs of integers. All integers are separated by single spaces or newlines. The y–th intersection is defined by the y–th pair of integers 'x p', where $1 \le y < n$, $0 \le x < y$, $-1000 \le p \le 1000$. This pair indicates a road segment between y and a previously defined intersection, x, with a profit value p. [1]

The input will be terminated by a line consisting of one zero (`0`). This line should not be processed.

[1] Attention, for this program, input lines may contain up to 4096 characters each!

SAMPLE INPUT:

```
6
0 -1 1 3 0 2 1 1 1 4
6
0 2 0 1 0 2 0 1 1 1
5
0 1 1 -3 0 -2 1 -2
5
0 -1 1 -3 0 -2 1 -2
10
0 -1 0 -1 0 0 1 3 1 4 2 4 2 2 3
3 3 3
0
```

OUTPUT FORMAT

Output will be a sequence of lines, one for each input scenario. Each line will contain an integer, indicating the maximum non–negative profit, over all possible simple road chains connecting two intersections of the subdivision. Write zero (`0`) if no profit can be obtained.

SAMPLE OUTPUT:

```
7
5
1
0
7
```

PROBLEM 6 – RANKINGS

Consider a tournament in which each team plays the same number of games, but does not necessarily play all the opposing teams the same number of times. For every team, every game can end in a win, a draw or a loss. Tournament points are accumulated in two ways: (a) for wins and draws (there are no points for a loss) and (b) for gaining bonus points during games.

During a game, a team can accumulate a final score based on three different events (*A*, *B* and *C*), each with its own value. The final score for a game is the sum of the event values obtained, i.e., the number of *A*'s accumulated times the value for an *A*, plus the number of *B*'s accumulated times the value for a *B*, plus the number of *C*'s accumulated times the value for a *C*.

A team gains 1 tournament bonus point if it scores *J* or more *A* events in a game, regardless of the final outcome of the game. Furthermore, a team gains 1 tournament bonus point if they lose a game by no more than a final score difference of *K*.

The governing board, which is responsible for choosing the points for a win or draw, the values for events *A*, *B*, and *C*, the number *J*, and the value *K*, decided to investigate the effect of changing those parameters on the team rankings.

Your task is to determine the highest possible ranking for each team (so that as few as possible teams do strictly better), by changing the values of the above parameters, subject to the following constraints:

- a win must be worth between 2 and 5 points,
- a draw must be worth at least 1 and be worth (strictly) less than a win,
- the values for *A*, *B* and *C* are in the range 1 to 5,
- *J* is in the range 1 to 5,
- *K* is in the range 1 to 10.
- All quoted ranges include both endpoints.

INPUT FORMAT

Input for this problem consists of a sequence of one or more scenarios. Each scenario describes a tournament.

- The first line of a scenario contains two integers, n, m, $1 \le n \le 20$, $1 \le m \le 20$, separated by a single space, where n is the number of teams and m the number of games for each team.

- Each of the following lines of the scenario describe a game in the following format, with items separated by single spaces:

  $T_1\ A_1\ B_1\ C_1\ T_2\ A_2\ B_2\ C_2$

  - T_i, i=1,2, are team labels, which are sequences of 1 to 20 characters, each character being either a lower–case letter, a digit, or a minus sign (`-`).
  - A_i, B_i, C_i, i=1,2 are the number of events of type A, B, C, respectively, accumulated by team i. A_i, B_i and C_i are integers from 0 to at most 9.

The input will be terminated by a line consisting of two zeros (`0`), separated by a single space. This line should not be processed.

SAMPLE INPUT:

```
2 2
a-team 1 1 1 b-team 0 3 0
b-team 0 0 3 a-team 1 1 1
3 2
bob 1 1 1 fred 0 1 1
fred 2 1 0 jane 0 1 0
jane 0 2 0 bob 0 2 0
0 0
```

OUTPUT FORMAT

For each scenario, output the scenario number, starting with 1, on a single line. Follow that by one line for each team, sorted by team name. For each team, output the team name, followed by a single space, followed by its best possible ranking.

SAMPLE OUTPUT:

```
1
a-team 1
b-team 1
2
bob 1
fred 1
jane 3
```

PROBLEM 7 – SUDOKU

The Sudoku craze is sweeping the world. Practically every newspaper now carries a daily puzzle and there are even books of puzzles (and helpful hints about how to solve them) available from several publishers. This example shows a typical Sudoku puzzle (left) and its solution (right):

	2	3	7		8	9		
					4			
	4	9					8	1
	9				7	8		6
5		8	3				2	
8	6					5	3	
			8					
		2	1		3	6	9	

6	2	3	7	1	8	9	5	4
1	8	5	2	9	4	7	6	3
7	4	9	6	3	5	2	8	1
3	9	4	5	2	7	8	1	6
2	1	6	4	8	9	3	7	5
5	7	8	3	6	1	4	2	9
8	6	1	9	4	2	5	3	7
9	3	7	8	5	6	1	4	2
4	5	2	1	7	3	6	9	8

The aim is to fill in the blank squares such that the digits 1 to 9 appear exactly once in each row, each column and each of the nine 3×3 sub-squares that are marked by shading in the above diagram. Typically, the given digits are carefully chosen and placed so that there is only one solution.

The Sudoku Problem Preparation Company wants to get in on this craze and sees it as a way of making money. Their idea is to publish a series of puzzles with substantial prizes offered to the submitters of the correct answer. They would charge an entry fee and the perception would be that this would fund the prizes.

However, they have a different idea – they intend to produce puzzles with several different solutions and then, when all the answers come in, they will claim that no-one has won the prize and produce an answer that differs from all the submissions. Given that they expect lots of submissions, they obviously want to choose puzzles with several different solutions. This is where you come in.

Write a program that will read in details of several Sudoku puzzles and determine how many different solutions there are for each of them. None of the given puzzles will admit more than 10000 different solutions.

Input Format

Input for this problem consists of a sequence of one or more scenarios. Each scenario contains 10 lines.

- The first line contains a puzzle label, which is an integer, *n*, $1 \le n \le 100$.
- The puzzle is described on the next 9 lines, each containing a string of 9 digits in the range 0 to 9, including the limits. Zeroes will denote the blanks that need to be filled in, other digits will denote themselves.

The input will be terminated by a line consisting of a single zero (`0`). This line should not be processed.

Output Format

Output will be a sequence of lines, one for each input scenario. Each line will contain in order the puzzle label and the number of ways of solving that puzzle (0 if none), separated by a single space.

Sample Input:

```
10
023708900
000004000
049000081
090007806
000000000
508300020
860000530
000800000
002103690
20
003708900
000004000
049000081
090007806
000000000
508300020
860000530
000800000
002103690
0
```

Sample Output:

```
10 1
20 2
```

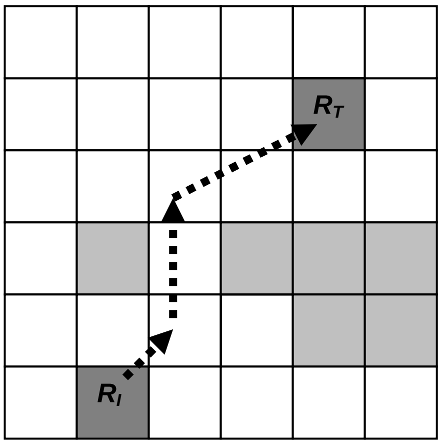
PROBLEM 8 – ROBOTS

A popular Acmian strategy game consists of moving a robot on a terrain with fixed obstacles. The objective is to move the robot from a given initial position to a given target position.

In our case, the terrain is defined by a planar rectangular map. The robot and the obstacles have rectangular shapes, with corners on integer coordinates. All rectangles are proper, i.e., each side is at least 1 unit long. All obstacles and robot positions are completely within the rectangle bounding the map. The initial and target positions may overlap each other, but do not overlap existing obstacles. Obstacles themselves may overlap. Here, boundaries are not considered as belonging to the objects – thus, the robot, the obstacles and the map may share some of their boundaries.

The robot has a fixed shape, cannot rotate, but can move by translations in any direction, within the map borders, as long as it avoids collisions with the obstacles and with the map limits. The robot is allowed to slide alongside the borders of the obstacles or of the map.

The following diagram gives an example of this game, where R_I is the initial robot position, R_T the target position, the other shaded areas are obstacles, and the dotted line suggests the shortest robot path.



Your task is to write a program that determines the length of a shortest robot path on such a map with obstacles, if such a path exists.

For this problem, where necessary, use **`double`** floating point numbers, floating point additions, the `sqrt`() function, and the output display facilities, such as `printf`(), provided by your system. The final results must be displayed accurate to exactly 2 decimals. For example, in the above diagram scenario, report that the length of the shortest path is 5.65:

Sqrt(2) + 2 + Sqrt(5) = 1.4142… + 2.0 + 2.2360… ≈ 5.65

INPUT FORMAT

Input for this problem consists of a sequence of one or more scenarios. Each scenario is described by several lines.

- The first line defines, in order, a numeric scenario label, *l*, $1 \le l \le 100$, the number of obstacles, *m*, $1 \le m \le 10$, and the *x* and *y* coordinates of the top-right corner of the rectangle bounding the map, $1 \le x, y \le 1000$, separated by single spaces.
- The second line defines the initial and target robot positions, specifically, in order, the initial *x* and *y* coordinates of its bottom-left corner, the initial *x* and *y* coordinates of its top-right corner, and the target *x* and *y* coordinates of its bottom-left corner, separated by single spaces.
- The next lines contain $4 \times m$ integers, separated by single spaces or newlines. These integers define the *m* obstacles, by giving in order, for each obstacle, the *x* and *y* coordinates of its bottom-left corner, and the *x* and *y* coordinates of its top-right corner.
- All coordinates are bounded within the map rectangle, including the inferred *x* and *y* coordinates of the robot's top-right corner target position.

The input will be terminated by a line consisting of four zeros (`0`), separated by single spaces. This line should not be processed.

OUTPUT FORMAT

Output will be a sequence of lines, one for each input scenario. Each line will start with the scenario label, followed by the length of the shortest path, accurate to exactly 2 decimals, separated by a single space. If no such path exists, print the character minus (`-`) in place of the path length.

SAMPLE INPUT:

```
10 3 6 6
1 0 2 1 4 4
1 2 2 3 3 2 5 3 4 1 6 3
11 4 6 6
1 0 2 1 4 4
1 2 2 3 3 2 5 3 4 1 6 3 3
4 4 5
12 3 6 6
1 0 3 1 4 4
1 2 2 3 3 2 5 3 4 1 6 3
0 0 0 0
```

SAMPLE OUTPUT:

```
10 5.65
11 6.41
12 -
```

PROBLEM 9 – POLYCUBES

A Grant's cube is a 3×3×3 cube made of (usually wooden) *blocks*, also known as *polycubes*. Each of the blocks is made up of a number of unit cubes, joined on faces. In the standard Grant's puzzle there are 3 *tetracubes* (blocks made of 4 units), and 3 *pentacubes* (blocks made of 5 units), making a total volume of 27 units. However, larger or smaller blocks are allowed in this problem.

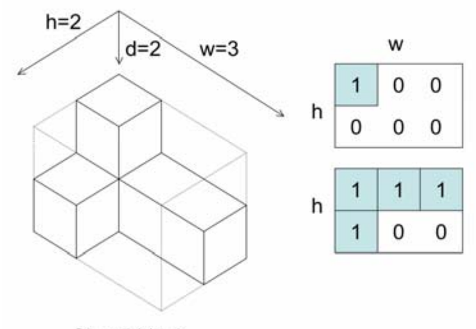
Your task is to determine if a *candidate set of blocks* can fit together to form a Grant's cube. Finding a solution may require rotation of some of the blocks from their initial orientations, and it is possible that no solution exists for a given set of blocks (that they can't form a cube at all). It does not matter whether the blocks can be physically brought together when determining if a solution exists. As long as they can be oriented so as to completely fill the 3×3×3 space without overlapping, that is enough for this problem.

INPUT FORMAT

The input file describes one or more scenarios. Each scenario is described on *k*+1 lines, where *k* is the number of blocks forming a candidate set. The first line contains a label, which is a string of 1 to 5 digits, and the number *k* of blocks in the set, 1 ≤ *k* ≤ 6, separated by single spaces.

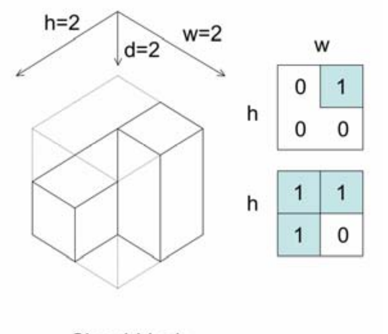
Each of the next *k* lines encodes a single block. Each such line begins with three dimensions, *d*, *h*, *w*, 1 ≤ *d*, *h*, *w* ≤ 3, representing the smallest rectangular solid that the block fits into, followed by a list of *d*×*h*×*w* binary digits, all items separated by single spaces. These binary digits are logically arranged in *d* groups of *h* groups of *w* integers, where a 1-digit means that the corresponding cell in the rectangular solid is part of the encoded block, whereas a 0-digit means that the corresponding cell is not part of the block. For an explanatory example, see the following diagram, which describes two blocks and their encodings:

```
2 2 3 1 0 0 0 0 0 1 1 1 1 0 0
```



Size 5 block

```
2 2 2 0 1 0 0 1 1 1 0
```



Size 4 block

The input will be terminated by a line consisting of two zeros (`0`), separated by a single space. This line should not be processed.

SAMPLE INPUT:

```
01 6
2 2 3 1 0 0 0 0 0 1 1 1 1 0 0
2 2 3 1 0 0 0 0 0 1 1 1 0 0 1
2 2 3 0 1 1 0 0 0 1 1 0 1 0 0
2 2 2 0 1 0 0 1 1 1 0
2 2 2 0 0 1 0 1 1 1 0
2 2 2 1 0 0 0 1 1 1 0
001 3
1 3 3 1 1 1 1 1 1 1 1 1
3 1 3 1 1 1 1 1 1 1 1 1
3 3 1 1 1 1 1 1 1 1 1 1
002 3
1 3 3 1 1 1 1 1 1 1 1 1
1 3 3 1 1 1 1 1 1 1 0 0
2 3 3 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 1 1
003 3
1 3 3 1 1 1 1 1 1 1 1 1
1 3 3 1 1 1 1 1 1 1 0 0
2 3 3 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 0 1
0 0
```

OUTPUT FORMAT

For each scenario, i.e., for each input candidate block set, the program should output a single line of text. The line of text should consist of the scenario label, followed the word "YES" (in uppercase) if there is at least a solution, otherwise "NO" (in uppercase), separated by a single space.

SAMPLE OUTPUT:

```
01 YES
001 YES
002 YES
003 NO
```