French-Australian Regional Informatics Olympiad

10–12 March, 2006

Duration: 4 hours

4 questions

All questions should be attempted

Problem 1 Guards

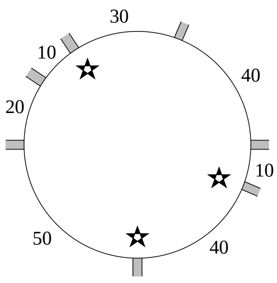
Time and Memory Limits: 3 seconds, 30 Mb

The time has come for the annual Boomerang and Bagel Throwing Olympiad, and the hosts have spared no expense. A large portion of the city has been sealed off completely to house the athletes and journalists, not to mention the barbeque makers and French pastry chefs. Amidst the hype and excitement, you have found yourself in charge of the security of this event.

Every road coming into the Olympiad grounds has been blocked off, to ensure that there are no unwanted intruders. However, the hosts have spent so much money on the opening and closing ceremonies that there is not enough money left to hire enough security guards. Your aim is to hire as few guards as possible but still keep the grounds secure.

The Olympiad grounds can be thought of as a large circular region, with roads coming in at various points on the border. The government requires that every point where a road enters the grounds must be at most k metres from a security guard, where distance is measured around the edge of the circle. Guards may only be placed at roads; they may not be placed in the regions in between.

An example is shown in the diagram below. Here there are seven roads coming into the grounds, with the distances between them marked on the diagram. Suppose that the government has declared that $k = 30$. In this case, by placing three guards at the roads marked with stars, you can ensure that every road is watched.



Your task is, given the distances between roads and the value of k, to determine the smallest number of guards that are required.

Constraints

- $1 \le n \le 1\,000\,000$, where n is the number of roads coming into the Olympiad grounds;
- $1 \le d \le 1\,000$, where d is the distance between two adjacent roads;
- $1 \le k \le 10\,000\,000$, where k is the value given by the government as described above.

Furthermore, for 30% of the available marks, the number of roads will satisfy $1 \le n \le 1\,000$.

Input

Your program must read from standard input. The first line will contain the integers n and k separated by a single space, as described above.

Following this will be n lines, each describing the distance between two adjacent roads. These distances will be given in order as one travels around the circle.

More precisely, suppose that the roads are numbered $1, 2, \ldots, n$ in clockwise order around the circle. Then the ith of these lines will be the clockwise distance between the ith and $(i + 1)$th roads, with the nth line giving the clockwise distance between the nth and first roads.

Output

Your program must write a single line to standard output. This line must contain a single integer giving the smallest number of security guards required.

Sample Input

```
7 30
30
40
10
40
50
20
10
```

Sample Output

```
3
```

Scoring

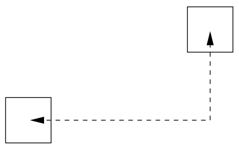
The score for each input scenario will be 100% if the correct answer is output, or 0% otherwise.

Problem 2
Escape from Civilisation

Time and Memory Limits: 3 seconds, 30 Mb

After too many years as a computer programmer, you decide it's time to retire and get in touch with your inner child. You hurriedly pack up all of your non-technological belongings (which isn't all that much) and head away from the city, away from the Internet, away from civilisation, and into the country, to live a simple life.

However, before you can be on your way, you need to write one last program. You wish to be as far away from civilisation as possible — in one of the most isolated areas known to humanity. You possess a map of width w and height h. This map tells you where the areas of civilisation can be found. The most isolated areas are those map squares that have the furthest Manhattan distance from any civilisation. The Manhattan distance between two squares is the sum of the up/down distance and the left/right distance, as illustrated below.



Your task is to determine how far the most isolated areas are from civilisation.

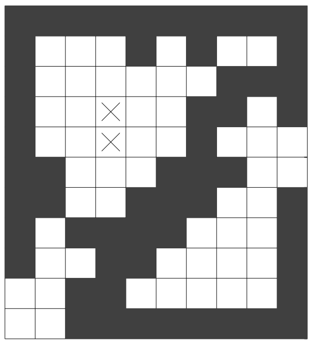
Constraints

- $1 \leq w, h \leq 1\,000$, where w is the width of the map, and h is the height of the map;
- you are guaranteed there will always be at least one map square of civilisation and one map square with no civilisation.

Furthermore, for 30% of the available marks, the dimensions of the map will satisfy $w, h \leq 50$.

Example

Consider the map below, of width 10 and height 11. Areas of civilisation are shaded. The two squares marked with an $\times$ are the most isolated squares — no other map squares are further from civilisation than these. The marked squares each have Manhattan distance 3 from the nearest civilisation, and so the answer to the problem is 3.



Input

Your program must read from standard input. The first line will contain two positive integers w and h, separated by a single space. The following h lines will each contain w integers. Each of these integers will either be a 1, denoting the presence of civilisation, or a 0, denoting no civilisation.

Output

Your program must write a single line to standard output. This line must contain a single integer, giving the Manhattan distance from civilisation to the most isolated map square.

Sample Input

```
10 11
1 1 1 1 1 1 1 1 1 1
1 0 0 0 1 0 1 0 0 1
1 0 0 0 0 0 0 1 1 1
1 0 0 0 0 0 1 1 0 1
1 0 0 0 0 0 1 0 0 0
1 1 0 0 0 1 1 1 0 0
1 1 0 0 1 1 1 0 0 1
1 0 1 1 1 1 0 0 0 1
1 0 0 1 1 0 0 0 0 1
0 0 1 1 0 0 0 0 0 1
0 0 1 1 1 1 1 1 1 1
```

Sample Output

```
3
```

Scoring

The score for each input scenario will be 100% if the correct answer is output, or 0% otherwise.

Problem 3
Belts

Time and Memory Limits: 0.1 seconds, 30 Mb

It's fashion week, and you are trying on the latest designer belts from Paris. Unfortunately none of the belts fit you, which can mean only one thing. You need more exercise.

You decide the best time to exercise is during your tram ride home. The tram runs in a straight line from school to home, and there are several stops on this line. You can get some exercise by hopping off the tram at some stop, walking to another stop further down the line, and then hopping onto the next tram that comes by. You can do this as many times as you like (tram, walk, tram, walk and so on). You may choose to start the journey walking if you wish, and you may choose to hop off a tram at some point and walk the rest of the way home.

Trams run every t minutes, with the first tram leaving school precisely when you begin your journey. Trams move at a fixed speed, and you walk at a slower fixed speed. You may only walk forwards — you may not walk backwards or walk around in circles. If you are walking from stop a to stop b and catching the next tram, you simply wait at stop b and listen to your mp3 player until the next tram comes along. If you arrive at stop b at precisely the same moment as a tram comes past, you may hop on board.

After poring through a handful of health magazines, you decide that you need to walk at least k metres during your travels home. Your task is to find the shortest possible travel time from start to finish that meets this requirement.

Constraints

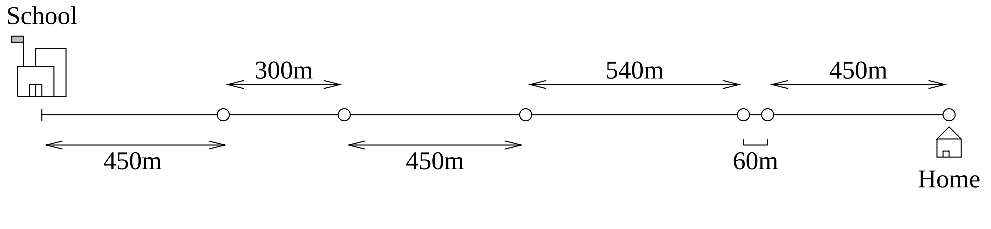
All times below are measured in milliseconds, and all distances are measured in metres.

- $30\,000 \le t \le 1\,000\,000$, where t is the time between each tram and the next;
- $1 \le m_t < m_w \le 100$, where m_t is the time it takes a tram to move one metre and m_w is the time it takes you to walk one metre;
- $1 \le k \le 10\,000$, where k is minimum distance that you must walk;
- $1 \le s \le 100$, where s is the total number of tram stops (excluding the school where your journey begins);
- No two adjacent tram stops are more than $1\,000$ metres apart;
- The last stop is at least k metres from the school (i.e., it is possible to actually get the exercise that you need).

Furthermore, for 60% of the available marks, the minimum walking distance will satisfy $k \le 2\,000$.

Example

Consider the tram line illustrated below. It begins with the school on the left hand side, after which come $s = 6$ tram stops. Distances between consecutive stops are marked on the diagram, and your house is at the very last stop.



Suppose that the council is extremely efficient, and that trams run every 30 seconds, i.e., every $t = 30\,000$ milliseconds. Trams take $m_t = 1$ millisecond to move one metre, and you take a brisk $m_w = 100$ milliseconds to walk one metre. Your fitness regime requires you to walk at least $k = 870$ metres during your journey.

An optimal strategy for getting home is as follows.

Action	Stops	Distance	Time
Tram	School–1	450m	450ms
Walk	1–2	300m	30 000ms
Wait for tram	2		300ms
Tram	2–3	450m	450ms
Walk	3–5	600m	60 000ms
Wait for tram	5		600ms
Tram	5–6	450m	450ms
			92 250ms

Your total walking distance according to this plan is 900m, which is enough since it is more than the required $k = 870$m. The total travel time is 92 250 milliseconds.

Input

Your program must read from standard input. The first line will contain the single integer t, giving the time between each tram and the next. The second line will contain the two integers m_t and m_w separated by a single space, giving the time it takes a tram to move one metre and the time it takes you to walk one metre. The third line will contain the single integer k, giving the minimum distance that you need to walk.

The fourth line of input will contain the single integer s, giving the total number of tram stops (excluding the school where the journey begins). Following this will be s lines, giving the positions of the individual stops in consecutive order from school to home. The ith of these lines will contain the integer d_i, giving the distance from the school to the ith stop. Your house is at the last of these stops.

Once more, all times are measured in milliseconds and all distances are measured in metres.

Output

Your program must write a single line to standard output. This line must contain a single integer giving the shortest time possible for the entire journey, measured in milliseconds.

Sample Input

```
30000
1 100
870
6
450
750
1200
1740
1800
2250
```

Sample Output

```
92250
```

Scoring

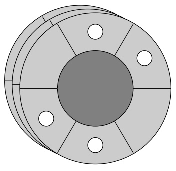
The score for each input scenario will be 100% if the correct answer is output, or 0% otherwise.

Problem 4
Lucky Wheels of Doom

Time and Memory Limits: 3 seconds, 30 Mb

It's time for your weekly trip to the fortune teller. Last week she read your palm; this week she is bringing out the Lucky Wheels of Doom.

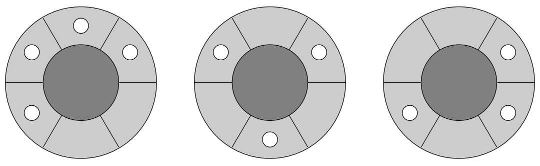
The Lucky Wheels consist of a set of w circular wheels, placed one behind another as illustrated below. The rim of each wheel is divided into s equal segments, some of which have holes cut out of them.



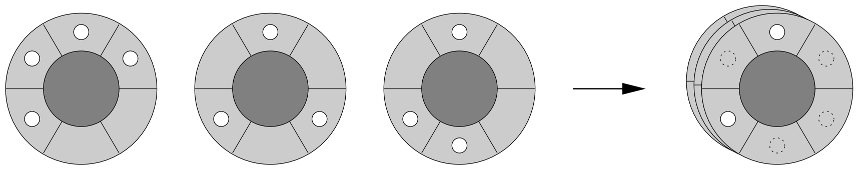
Your fortune teller will ask you to spin each wheel in turn. Once all of the wheels have been spun, she will look for holes that are lined up all the way from the front to the back (so that you can see right through the entire set of wheels). For each set of holes that are lined up, you are guaranteed a year of prosperity and bug-free code.

Fortunately you have done this many times before, and so you are able to spin each wheel precisely as far as you would like. Your task is to decide how far to spin each wheel so that many sets of holes can be lined up.

As an example, consider the set of $w = 3$ wheels illustrated below, each of which is divided into $s = 6$ segments.



By spinning the second wheel one segment clockwise and spinning the third wheel two segments clockwise, you can arrange the three wheels as shown below. When the wheels are placed one behind another, there are two sets of holes lined up all the way from front to back. This is illustrated in the final diagram on the right hand side.



You are not required to give the best possible solution. Instead you must line up as many sets of holes as you can. Your program will then compete with other programs, with the better solutions scoring more points.

Constraints

- $1 \leq w \leq 50$, where w is the total number of wheels;
- $1 \leq s \leq 50$, where s is the number of segments into which each wheel is divided.

Furthermore, for 30% of the available marks, the number of segments will satisfy $s \leq 20$.

Input

Your program must read from standard input. The first line will contain the two integers w and s separated by a single space, as described above.

Following this will be w lines, each describing a single wheel. Each of these lines will contain s integers separated by spaces, describing the segments of the wheel in clockwise order from the top. For each segment, the integer 0 represents a hole in the wheel and the integer 1 represents no hole.

Output

Your program must write $w+1$ lines to standard output. Of the first w lines, the ith line corresponds to the ith wheel from the input data, and should contain a single integer describing the number of segments the wheel should be rotated clockwise. Each output integer must be between 0 and $s-1$ inclusive.

After these w lines have been written, your program must write one additional line of output. This line must contain a single integer giving the total number of sets of holes that are aligned all the way from front to back.

Sample Input

```
3 6
0 0 1 1 0 0
1 0 1 0 1 0
1 0 0 1 0 1
```

Sample Output

```
0
1
2
2
```

Scoring

There is no particular "best solution" that you are required to achieve. Instead, your score will be determined relative to the other contestants whom you are competing against (as well as the judges' solution).

One simple solution involves examining each wheel in turn from top to bottom, and at each stage rotating the wheel to line up as many holes as possible with the wheels above it. Such a solution will be defined to score 30%.

For each input scenario, the contestant who lines up the most sets of holes will be identified. Suppose that this contestant lines up h sets of holes in total. Your score for this input scenario will then be:

- 100% if your program finds a solution also with h sets of holes lined up;
- 30% if your program finds a solution equal to the simple solution outlined above;
- 0% if your program generates an incorrect solution (e.g., you incorrectly calculate how many sets of holes are aligned, or you do not follow the output format exactly);
- otherwise determined by a linear scale according to how many sets of holes your program aligns, with the 100% and 30% marks on the scale corresponding to the solutions described above.

For example, consider an input scenario for which the simple solution aligns 26 sets of holes. If the best solution found by any contestant (or the judges) aligns 40 sets of holes, then the scoring scale for a *correct* solution would be as follows:

Holes aligned	40	36	32	28	26	24	20	16
Score	100%	80%	60%	40%	30%	20%	0%	0%