894 Juggling Trams

After many years of faithful service, the good city of Melbourne has decided to upgrade all of its aged green and gold trams to newer, more fashionable models. But alas, the pursuit of fashion has not been without its costs. The newer trams, whilst looking stunning, can only travel in straight lines.

But this has not deterred the Melbourne City Council. In order to deal with this problem, they have laid more tracks and built more trams, so that the city is now a vast grid of tram tracks criss-crossing throughout the city.

For simplification, consider the city to be a large grid of roads running north-south and east-west. Each road has a tram line running along it. You, the humble traveller, wish to travel south and west through this grid using the new tram system.

Trams run along each road at t-minute intervals, where t is a fixed time that has been determined by the Melbourne City Council. Thus, if you stand at any street intersection, you will see a tram travelling south every t minutes, and a tram travelling west every t minutes (although the trams will not necessarily pass by at the same times). We will assume that trams run at a constant speed, and take no time to pick up or drop off passengers.

Based on your training in city traversal optimisation theory, you wish to write a computer program that will find the fastest route from your starting point to your finishing point through the grid. You can hop on or off a tram at any intersection, and you may need to wait at some intersections for a tram to come along. You may assume that hopping on or off a tram takes no time at all. So, for instance, if a southward tram passes through an intersection at the same time as a westward tram, you may hop off the southward tram and immediately hop onto the westward tram.

Input

Input will consist of a number of data sets. The first line of each data set will be of the form

$t\ m$

where t and m are both integers. t represents the number of minutes between each tram travelling down a street, as described earlier, with $1 \leq t \leq 60$ inclusive. m represents the number of minutes it takes a tram to move from one intersection to the next, with $m > 0$. The second line of the data set will be of the form

$n\ e$

where n is the number of north-south streets and e is the number of east-west streets. Both n and e will be between 1 and 200 inclusive. North-south streets will be numbered from 1 to n, where 1 is the eastmost street and n is the westmost street. East-west streets will be numbered from 1 to e, where 1 is the northmost street and e is the southmost street. The third line will be of the form

$sx\ sy\ fx\ fy$

and will describe the start and end points of your journey. Your journey must begin at the intersection of north-south street sx and east-west street sy, and must end at the intersection of north-south street fx and east-west street fy. The fourth line will contain a single integer representing the number of minutes after midnight when your journey begins.

Following this will be a line for each north-south street of the form

$first\ k$

where $first$ is the number of minutes after midnight when the first tram leaves the eastmost intersection along that street, and k is the total number of southward trams that will travel down that street throughout the day ($k > 0$). Note that these k trams will be spaced by intervals of t minutes. A line

$first\ k$

will then be given for each east-west street, where $first$ is the number of minutes after midnight when the first tram leaves the northmost intersection along that street, and k is the total number of westward trams that will travel down that street throughout the day ($k > 0$).

Input will finish with $t = 0$ and $m = 0$.

Output

Output must be one line for each case of input. If it is possible to travel from the start point to the finish point, you must output a line of the form:

`You arrive at` hh`:`mm`.`

where hh : mm is the earliest time of day (using a 24-hour clock) at which you can arrive at your destination. Both hours and minutes must be two digits; use a leading '`0`' if necessary. You may assume that if a solution exists, the time of arrival will be before the next midnight.

If it is impossible to reach the destination, output the line: '`Impossible.`'

Sample Input

```
30 3
5 4
2 2 5 4
93
30 5
100 6
115 4
100 7
30 8
10 10
0 11
20 9
10 10
30 3
5 4
2 2 5 4
300
30 5
100 6
115 4
100 7
30 8
10 10
0 11
20 9
10 10
0 0
```

Sample Output

```
You arrive at 01:52.
Impossible.
```