2012/2013 SOUTHERN CALIFORNIA REGIONAL
ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 1
Cap and Trade

California Assembly Bill 32, which goes into effect on January 1, 2013, requires companies to reduce carbon emissions or to pay a fee to offset greenhouse gases. (Technically, the latter is to purchase credits from businesses that are not emitting their allotment of greenhouse gases.) The allotment goes down each year until goal numbers are met. For those businesses that do not reduce emissions, annual costs would continue to rise as the goals are lowered. All businesses, including colleges and universities, could be impacted, depending on their sizes.

Implementing measures that reduce greenhouse gases can be expensive and take time to implement. Colleges and universities that are large enough to be impacted by this bill have made estimates of what the budget impact would be for next year if they do not reduce carbon emissions.

Swamp County College is one of the impacted colleges. It would like to explore what the costs would be if distributed on a per-unit basis as a fee.

Your team is to write a program that takes assumptions about likely costs and anticipated enrollments and reports the ranges of fee increases that would be needed to cover the assumed costs. The fee is to be calculated by credit unit taken and by full-time equivalent student (FTE). The number of units per FTE will vary by campus.

Input to your program will be one or more lines of text of up to eighty characters. Each line will have the following information, separated by commas, with no leading white space:

- College or University identifier (string that does not contain a comma)
- Total number of anticipated enrolled units for a year for all anticipated enrollments ($90 \leq$ units $\leq$ 1,000,000)
- Number of units considered a "full time" load ($9 \leq$ full-time load ≤ 18)
- Minimum annual assumed cost for emissions (integer dollars)
- Maximum annual assumed cost for emissions (integer dollars)

Output will consist of a single line for each college or university, in the same order as the input, formatted beginning in the first column of the line as follows:

- College or University identifier, followed by a single space
- The string "Fees per Unit: "
- The minimum cost per unit, rounded to the nearest dollar with a leading dollar sign, followed by a hyphen ("-"), followed by the maximum cost per unit, rounded to the nearest dollar, with a leading dollar sign
- A single space, followed by the string "Full Time: "
- The minimum cost for a full-time student, rounded to the nearest dollar, with a leading dollar sign, followed by a hyphen ("-"), followed by the maximum cost per unit, rounded to the nearest dollar, with a leading dollar sign.

Rounding is to occur after the per-unit and per-FTE values are calculated.

Sample Input

```
SCC,90000,15,485000,967000
UCX,150000,15,1500000,6000000
```

Problem 1
Cap and Trade (continued)

Output for the Sample Input

```
SCC Fees per Unit: $5-$11 Full Time: $81-$161
UCX Fees per Unit: $10-$40 Full Time: $150-$600
```

2012/2013 SOUTHERN CALIFORNIA REGIONAL
ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 2
Component Testing

The engineers at ACM Corporation have just developed some new components. They plan to spend the next two months thoroughly reviewing and testing these new components. The components are categorized into several different classes, depending on their complexity and importance. Components in different classes may require different number of reviewers, whereas components in the same class always require the same number of reviewers.

There are also several different job titles at ACM Corp. Each engineer holds a single job title. All engineers holding a given job title have the same limit on the number of components that they can review. Note that an engineer can be assigned to review any collection of components and will be able to complete the task, regardless of which classes the components belong to. An engineer may review some components of the same class, and others from different classes, but an engineer cannot review the same component more than once.

Can the engineers complete their goal and finish testing all components in two months? Your team is to develop a program to answer this question.

There will be multiple test cases in the input. The first line of each test case contains two integers n and m, separated by whitespace, where n is the number of component classes and m is the number of engineer job titles. n and m will be in the range 1–10,000 inclusive. Each of the next n lines contains two integers j and c indicating that there are j components in this class and that each component requires at least c different reviewers. j is in the range 1–100,000 inclusive, c is in the range 0–100,000 inclusive. This is followed by m lines, each containing two integers k and d separated by whitespace, indicating that there are k engineers with this job title and that each engineer may be assigned to review at most d components. k is in the range 1–100,000 inclusive, d is in the range 0–100,000 inclusive. Input will be terminated by the end-of-file.

For each test case, print a single line containing "Yes" if it is possible for the engineers to finish testing all of the components or "No" otherwise. No leading or trailing whitespace is to appear on an output line.

Sample Input

```
3 2
2 3
1 2
2 1
2 2
2 3
5 2
1 1
1 3
1 1
1 3
1 1
1 20
1 4
```

Problem 2
Component Testing (continued)

Output for the Sample Input

```
Yes
No
```

2012/2013 SOUTHERN CALIFORNIA REGIONAL
ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 3
BeepBeep

Swamp County College is running an engineering contest. Each competing team is given a bucket of parts with which to design and build a mini robot that has to maneuver around a square game board.

Your team's job is to determine the location of the robot when it stops at various grid point locations on the board. The most promising items in the bucket appear to be three old WiFi enabled smart phones. A quick check of the GPS reception in the contest location shows that GPS doesn't work reliably. Sound looks promising but the first tests are pretty disappointing: it doesn't seem possible to synchronize the clocks of the phones close enough, and even if you could, there seems to be an unpredictable delay between commanding a beep and the beep being emitted.

Luckily you have access to the awesome ACM Digital Library, and quickly find a paper by Peng, Shen, and Zhang that makes things clear. What the phones can do reliably is record sounds. These phones, like most phones, take 44,100 samples per second. The key is to use the recorder to determine when the beep is actually emitted rather than when it is commanded.

We can put two of the phones, $P1$ and $P3$, at known fixed locations and the third phone, $P2$, on the robot. After determining the distances between phones $P1$ and $P2$ ($P12$) and phones $P2$ and $P3$ ($P23$) we can intersect the distance circles to find the location of the robot. The equation of a circle of radius r centered at (h, k) is:

$$(x - h)^2 + (y - k)^2 = r^2$$

The robot moves around a board divided into 100 by 100 grid lines as shown in Figure 1. Reflective markers are placed at the intersections of the grid lines. The robot stops on one of these markers after each move.

See Figure 2 for a time diagram of the sound events. Note that there is no vertical scale in Figure 2.

A beep is emitted from $P1$ and received by $P1$ at $T1$, at $P2$ at $T2$, and $P3$ at $T3$. Some time after $T2$, $P2$ emits a beep that is received by $P2$ at $T4$, at $P3$ at $T5$, and $P1$ at $T6$.

The known quantities are $P13$, the distance between the fixed phones, and the differences between the events at each phone: $I3 = (T5 - T3)$, $I2 = (T4 - T2)$, and $I1 = (T6 - T1)$. We need to find $P12$ and $P23$. We can see, for example, that $I1 = I2$ + twice the time it takes sound to go from $P1$ to $P2$.

The known fixed locations for $P1$ and $P3$ need to be at least a few samples away from the board grid and located so as to make sure the distance circles always intersect in two points, only one of which is on the board square.

We will label the grid lines 0–99, locate $P1$ at $(-20, -20)$, and locate $P3$ at $(-20, 119)$. We don't know the spacing of the grid lines but it is at least 5 cm and not greater than 10 cm. To determine the speed of sound we will start the robot at $(50, 50)$.

Your program is to read a series of lines terminated by end of file. Each line will have three integers separated by whitespace. Each integer will be greater than zero and less than 100,000. The integers represent the number of samples between the beeps at $P1$, $P2$, and $P3$ for each location of the robot.

For each line in the input print the location x and y coordinates to the nearest integer. The correct answer to the first line will be

```
50 50
```

Print no spaces before the x coordinate, one space between x and y, and no spaces between y and a newline. Print no sign on x or y.

Problem 3
BeepBeep (continued)

Sample Input

```
1297 13 391
1361 87 455
1374 110 478
1267 77 571
1600 36 1008
919 99 472
389 23 87
1008 64 100
1462 100 288
1727 27 920
1770 96 589
2202 20 992
```

Output for the Sample Input

```
50 50
49 50
49 49
60 25
99 0
40 0
0 0
0 50
25 75
99 35
58 83
99 99
```

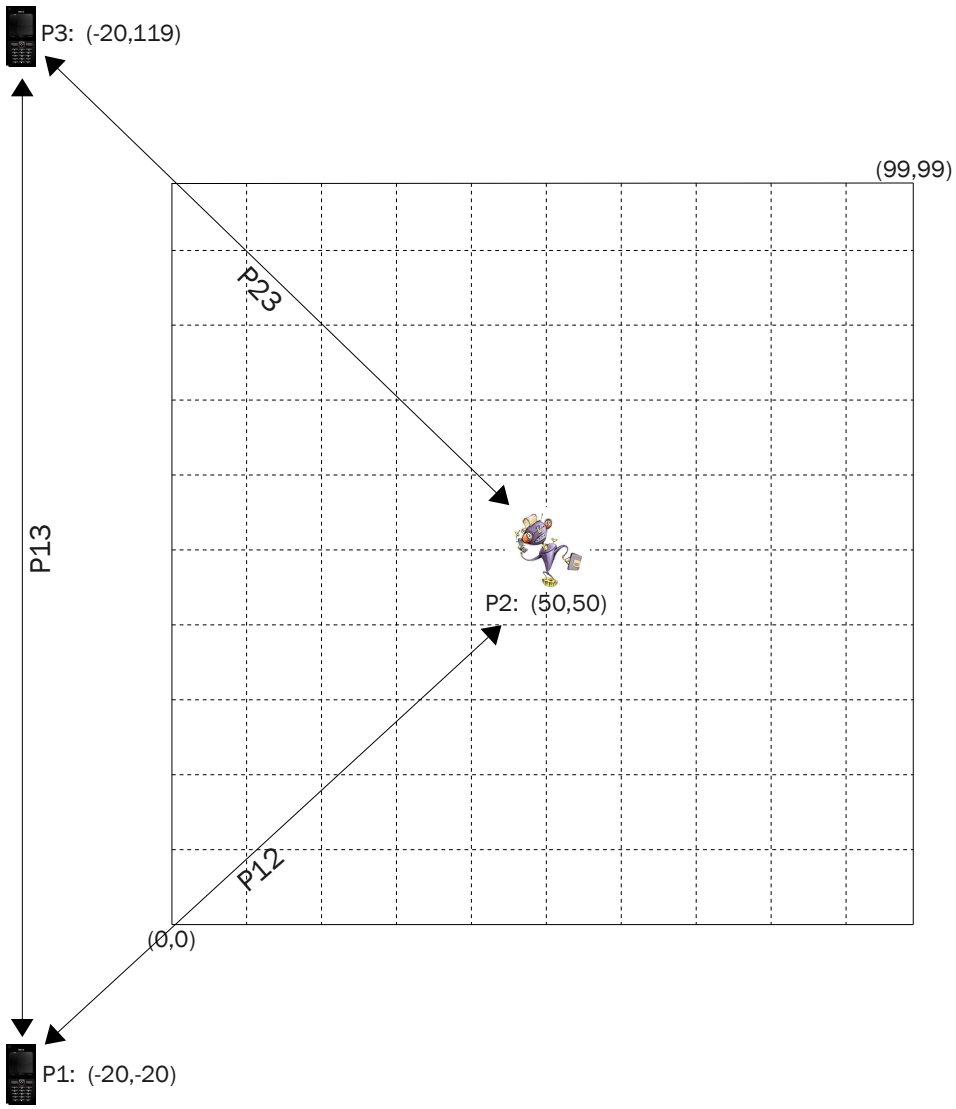

Figure 1. Fixed phones and initial position of robot.

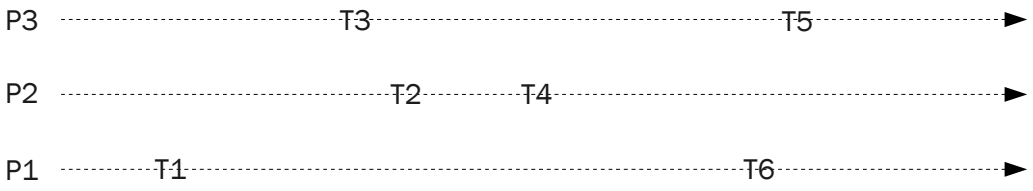

Figure 2. Independent time lines showing times of beeps received at each phone.

2012/2013 SOUTHERN CALIFORNIA REGIONAL
ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 4
Fifty Coats of Gray

A contractor is planning to bid on interior painting for a series of modest housing projects. These projects typically involve several homes or apartments that are all built using just a few floor plans, and have basic drywall walls and ceilings, with no particular architectural features like crown molding or cathedral ceilings. The contractor would like to find a quicker way to estimate how much paint it will take to paint the walls and ceilings for each job.

A gallon of paint will cover 350 to 400 square feet if walls are plain, flat, and light-colored. It will cover less if walls are textured or a dark color, which could require additional coats of paint.

The plan for these housing units is to paint the four walls and the ceiling. Of course, no paint is needed for window and door openings. All walls, ceilings, windows, and doors are rectangular, and all walls are the same height in any given room. All rooms will be painted the same color.

The contractor will provide you with information about the dimensions of the rooms, the windows and doors for each floor plan, and the number of domiciles for each floor plan. Your team is to write a program that will tell him how much paint he should include in his bid.

Input to your program will be a series of integer values on multiple lines representing one or more floor plans. Values will be separated from each other by one or more spaces. The values will represent the following:

- Number of domiciles with this floor plan (1–100 inclusive)
- Assumed square feet that a gallon of paint will cover in this floor plan (100–400 inclusive)
- Number of rooms in each domicile (1–9 inclusive)
  For each room:
  - Three integers, representing the height, length and width of the room in feet (h, w, l): height will range from 8–20 feet; length and width will range from 3–50 feet
  - The number of windows and doors in the room (1–9 inclusive)
  - One or more pairs of values that represent the dimensions of windows and doors, in feet

The last floor plan will be followed by the end-of-file.

For each floor plan, your program is to print the integer number of gallon paint cans needed for the job. The value is to be printed on a single line without leading or trailing whitespace.

Sample Input

```
50 350 3
8 10 8 2 6 3 3 3
8 10 8 2 6 3 3 3
8 20 8 3 6 3 5 3 5 3
20 400 2
8 16 12  2  7 3  3 5
8 7 5    2  7 3  2 3
```

Output for the Sample Input

```
178
41
```

2012/2013 SOUTHERN CALIFORNIA REGIONAL
ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 5
Shift Differential

For hourly employees in union shops, there are various things that can adjust pay. One of these is called "Shift Differential." Shift differential is a cash add-on to an hourly rate to compensate an employee for working outside of standard business hours. Usually there are two categories: one for the evening or "swing" shift and one for the overnight or "graveyard" shift. Swing and graveyard shift hours and the rules for payment are defined in the collective bargaining agreement.

Example: Assume a base hourly rate of $15/hour.

Swing Shift Differential: $1.25	Swing Shift Hourly Rate: $16.25
Graveyard Shift Differential: $2.00	Graveyard Shift Hourly Rate: $17.00

Your team is working for a company that pays shift differential. They would like you to calculate the shift differential pay owed to their hourly employees.

Rules for the company's collective bargaining agreements:

1. Specific times of day are defined as "swing" and "graveyard" periods. The graveyard period starts when the swing period ends. These periods are defined as contiguous four to eight hour blocks defined by start and end times.
2. Employees are assigned to regularly scheduled eight-hour base shifts. The base shift times may include hours in the swing and/or graveyard periods.
3. An employee is defined as regularly working graveyard shift if at least four hours of his/her base shift fall in the graveyard period. This is true even if the other four hours fall in the swing period.
4. An employee is defined as regularly working swing shift if at least four hours of his/her base shift fall in the swing period, unless the employee is in the graveyard shift category.
5. An employee that is not defined as regularly working swing or graveyard shifts is defined as regularly working day shift.
6. An employee may be assigned to different work hours on any given day, and those hours may span more than one shift period.
7. An employee who regularly works graveyard shift will receive the graveyard shift differential for all hours worked.
8. An employee who regularly works swing shift will receive the swing shift differential for all hours worked, except for any hours that fall in the graveyard period. The employee will receive the graveyard shift differential for those hours.
9. An employee who regularly works day shift will receive shift differential for any hours worked during the defined swing or graveyard periods.
10. If overtime (work for more than eight hours) occurs during hours when a person is receiving shift differential, the shift differential is paid at the overtime rate. Time worked from eight up to twelve hours in a shift is paid at 1.5 times the usual rate ("time-and-a-half"). Time above twelve hours is paid at double the usual rate. No employee will work more than sixteen consecutive hours on a given day.

Shift differential is often paid as a separate check after the work is completed and time reported. The company you are working for follows this practice.

Your team is to write a program that calculates the weekly shift differential payments owed to a group of hourly employees based on a given collective bargaining agreement. Each employee will work five days during the week.

Problem 5
Shift Differential (continued)

Input to your program is as follows:

- The first line defines the swing and graveyard period parameters. The fields are: Swing Shift Start Time *(HH:MM)*, Swing Shift End Time *(HH:MM)*, Graveyard Shift End Time *(HH:MM)*, Swing Shift Differential Pay *(dollars and cents)*, and Graveyard Shift Differential Pay *(dollars and cents)*, all separated by commas.
- Second line: Employee ID *(string of up to forty non-comma characters)*, employee base shift start time *(HH:MM)*, employee base shift end time *(HH:MM)*, all separated by commas.
- Next five lines: Work Date *(YYYYMMDD)*, Actual Start Time *(HH:MM)*, and Actual End Time *(HH:MM)*, all separated by commas.

Eighth and subsequent lines will repeat lines 2–7.

All input will start at the beginning of the line and there will be no whitespace except possibly embedded in employee IDs. Input is terminated by end-of-file.

Output will be a list, one per line, of the employee IDs who are owed shift differential, a single space, and the amount of the differential owed. The amount owed is to be printed in dollars and cents, rounded to the nearest cent. It is guaranteed that at least one person will be owed shift differential. No leading or trailing whitespace is to appear on an output line.

Times are given using a twenty-four hour clock. Midnight will be shown as 00:00. You do not need to be concerned about handling transitions to and from Daylight Savings Time.

Sample Input

```
18:00,00:00,06:00,1.25,2.00
Wendell Bipken,08:00,16:00
20120109,08:00,16:00
20120110,08:00,16:00
20120111,08:00,18:00
20120112,08:00,18:00
20120113,08:00,19:00
Breanna Shock,08:00,16:00
20120109,12:00,20:00
20120110,10:00,18:00
20120111,10:00,18:00
20120112,08:00,16:00
20120113,11:00,19:00
Theodora Reeghbuckle,18:00,02:00
20120109,18:00,02:00
20120110,18:00,02:00
20120111,18:00,02:00
20120112,16:00,02:00
20120113,18:00,04:00
```

Output for the Sample Input

```
Wendell Bipken 1.88
Breanna Shock 3.75
Theodora Reeghbuckle 68.00
```

2012/2013 SOUTHERN CALIFORNIA REGIONAL
ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 6
The Perks of Planning Ahead

Having just downloaded the latest multi-player RPG, *"SkyFall Ex—The Creed of the Guild,"* a group of gamers are trying to plan how to maximize their character development over the course of the game.

Game characters advance by acquiring "perks." One perk is earned for every three levels the character earns, and the maximum number of levels is capped, so only a limited number of perks can be acquired. By reading the manual and early reviews of the game, the players have come up with a score for how desirable they believe each perk will be in the final stages of the game.

However, players can't simply grab the most valuable perks. The more advanced perks can only be taken if a game character qualifies by way of having earlier chosen certain prerequisite perks. For example, the perk "Sharpshooter 2" can only be taken after having selected "Sharpshooter 1," and the perk "Master Trader" can only be selected if the character already has "Merchant 2" and "Diplomat 1." Your team is to write a program to select the most valuable set of perks that can be acquired in the game world.

Input will contain multiple data sets. Each data set begins with a line containing a positive integer P, the maximum number of perks that a player can acquire. A zero value for this indicates the end of the input.

The second line of the data set contains from one to twenty-six integers in the range 1–100 separated by one or more spaces, giving the estimated value of the perks. Perks are identified by single upper-case alphabetic characters. The first integer in this line is the value of perk A, the second is the value of perk B, and so on.

This is followed by zero to twenty-six lines identifying the requirements for taking a perk. Each line will contain two or more uppercase alphabetic characters, separated by one or more spaces. The first character indicates the perk being described. The remaining characters indicate perks that must already belong to the character before this perk can be taken. For example, the line

```
B A C
```

would specify that perk B can be taken only if a character already has both perk A and perk C. Perks will not have mutual prerequisites: in this example, neither perk A nor perk C will require perk B. The end of the series of requirements, and of the data set, is indicated by a line containing only a period.

For each data set, your program is to print a single line containing the identifiers of the perks making up the most valuable attainable set. These are to be printed in ascending alphabetical order with no leading, intervening, or trailing whitespace.

If there is a tie among multiple sets for most value, print the one whose output line would come earliest in an alphabetical ordering.

Problem 6
The Perks of Planning Ahead (continued)

Sample Input

```
3
10 10 90 30 75 30
B A
C B
E D A
F D E
.
2
100 45 60 50 5
A C D
D C
B E
.
0
```

Output for the Sample Input

```
ADE
CD
```

2012/2013 SOUTHERN CALIFORNIA REGIONAL
ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 7
Loadmaster

(This problem uses an interactive server.)

After a disaster, a rapid response is often necessary to airlift relief supplies into the affected area. The American Catastrophe Management Corporation has developed a set of cargo containers that fit into the seats of passenger aircraft. Each of these containers can be loaded onto and/or relocated within a plane in one minute. There are many fine programs that calculate cargo configurations when all the cargo weights are known prior to loading. Your team is to write a program that loads a specified aircraft as truckloads of containers arrive.

An airplane has two important loading limits: weight and balance. The takeoff weight (plane + fuel + cargo), or *TOW*, cannot exceed the airplane's specified maximum takeoff weight, or *MTOW*. In the interest of loading speed, your program will be given a threshold minimum takeoff weight, expressed as a percentage of MTOW, that should be achieved before considering the plane sufficiently loaded. Weights are specified as a whole number of pounds.

If the balance (center of gravity, or *CG*) of a plane falls outside a specified range along the longitudinal axis, the plane becomes aerodynamically unstable. The lateral CG of a traditional passenger plane is not a concern because the seats are close to the longitudinal axis; any side-to-side imbalance is corrected by trim adjustments. The minimum and maximum CG points are expressed as inches from the *datum*. The datum is an arbitrary point along the longitudinal axis. See Figure 1. Often, the datum is a point in front of the aircraft so that CG computations don't involve negative numbers. The longitudinal CG of an airplane is determined from Equation 1:

$$\mathrm{CG} = \frac{\mathrm{TotalMoment}}{\mathrm{TotalWeight}} \tag{1}$$

where a Moment, M, is computed as $M = Arm \times Weight$. *Arm* is the distance from the datum, in inches, and *Weight* is in whole-number pounds. For example, Table 1 shows the center of gravity for a fueled plane with cargo weights A, B, C, and D at their respective arms. Note that the fueled plane is expressed as a weight with its arm at the CG of the unloaded plane.

Item	*Weight (MTOW = 2200)*	*Arm*	*Moment*	*CG (+35 to +44)*
Airplane	1340	37	49580	
A	352	35	12320	
B	212	72	15264	
C	240	48	11520	
D	50	92	4600	
Total	*2192*		*93284*	*42.5*

Table 1. Sample loading and CG computation.

Your program must produce a safely loaded plane (TOW $\leq$ MTOW *and* minCG $\leq$ CG $\leq$ maxCG) when *one or more* of the Ready Conditions specified below are met:

a. TOW $\geq$ threshold/100.0 $\times$ MTOW
b. all seats are loaded with containers
c. no additional trucks arrive after D_{max} minutes of having nothing to load (the need for relief exceeds the desire for efficient loading)
d. loading the lightest remaining unloaded cargo container would push TOW > MTOW.

Problem 7
Loadmaster (continued)

Program Interaction

Your program must converse with a server, issuing commands to standard output and receiving responses from standard input. Your program starts the conversation by querying the airplane configuration. After receiving the configuration from the server, your program then repeatedly reads a timed event and responds to it. The first command is "C", a single character followed by end-of-line. The server replies with the airplane configuration:

MTOW W_{fueled} A_{fueled}
threshold D_{max}
minCG maxCG P
a_1
...
a_P

The first configuration line contains MTOW, W_{fueled}, and A_{fueled}, separated by whitespace. MTOW is the maximum takeoff weight, W_{fueled} is the weight of the fueled plane (plane + fuel), and A_{fueled} is the moment arm of the fueled plane. The second line contains threshold and D_{max}, where threshold is the integer percentage of MTOW for minimum loading. D_{max} is the maximum idle delay to wait before deciding that no more cargo is arriving. Once your program decides not to load any further containers, D_{max} is the maximum number of minutes allowed to reshuffle the contents within the plane to achieve balance. The third line contains minCG and maxCG, the minimum and maximum center of gravity arms (in inches) and P, the number of passenger seats in the airplane. Following the third line are P additional lines, each specifying the loading arm of each seat, expressed as whole-number inches from the datum. Implicit in the order of the seat arms is the seat number.

After the configuration, your program reads an event from the server. An event is of the form

t [C
w_1
...
w_C]

where t is the current step time in minutes. If a truck arrives, C will be present, followed by w_1 through w_C weights, expressed as whole-number pounds. Implicit in the ordering of the cargo weights is the cargo container number. Each truck starts again with its own items 1–C.

Problem 7
Loadmaster (still continued)

Your program responds to an event with one of the following commands.

Command String	*Discussion*
L c_i p_j	Load cargo item i into seat position j.
M p_i p_j L c_m p_n	Move cargo from seat p_i to seat p_j *before* loading cargo item c_m into seat p_n.
M p_i p_j	Move cargo from seat p_i to seat p_j. In the interest of loading speed, your program can perform a move without load only when the current truck was emptied and no additional truck has arrived yet.
I	Idle. Do nothing.
A p_i p_j	Cease loading items onto the plane, but make final adjustments. The Adjust command is almost the same as Move, but Adjust prevents the server from forcing Loads or MoveLoads while unloaded cargo items remain. Because no cargo items are being loaded, your program may move a cargo container to and from the aisle (position 0) to perform swaps. Once the A command is issued, your program may issue only the A or R commands.
R	Ready. At least one of the Ready Conditions specified above has been met.

It is very important that your program flush standard output after issuing a command to the server.
e.g.

```
  C    fputs("C\n",stdout); fflush(stdout);
C++    cout << "C\n" << flush;
Java   System.out.println("C"); System.out.flush();
```

Some important considerations:

- Only one plane can be loaded at a time (different planes require separate execution runs).
- Containers can only be loaded from one truck at a time.
- The number of passenger seats on an airplane varies from 4 to 853.
- The maximum number of cargo containers in a truckload is 100.
- Once a container is loaded, it cannot be unloaded to achieve safe weight or balance.
- Although some planes carry passengers on more than one level, the vertical relationship of seats to the datum is ignored, and thus not specified.
- It will always be possible to achieve one or more of the Ready Conditions.

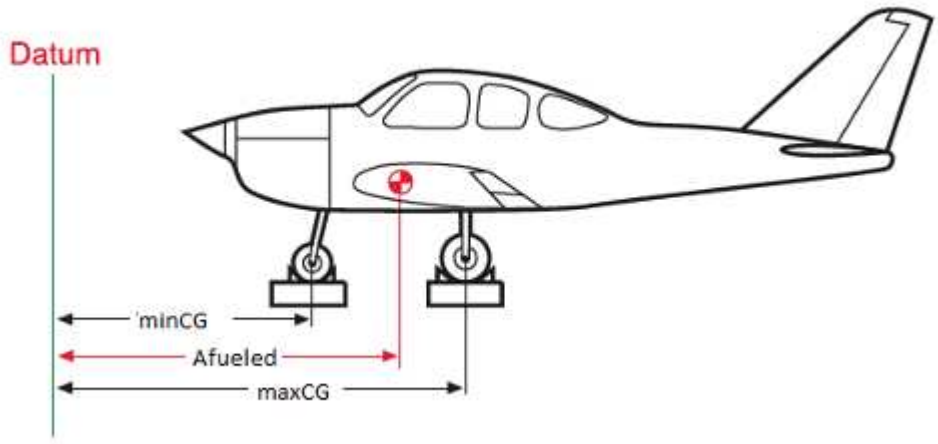

Figure 1. Diagram showing Datum and Center of Gravity parameters.

Problem 7
Loadmaster (still continued)

Sample Transcript

A sample transcript follows. Client commands are offset for clarity. The comments to the right of the transcript must not appear in your client commands and will not occur in the server responses; they are for description only.

Server	*Client*	*Comments*
	`C`	(Query configuration)
`2200 1340 37`		(MTOW W_{fueled} A_{fueled})
`90 3`		(threshold percentage and D_{max})
`35.0 44.0 4`		(minCG maxCG and P = 4 passenger seats)
`35`		(a_1 = 35.0 inches)
`72`		(a_2)
`48`		(a_3)
`92`		(a_4. This is the end of the configuration.)
`1 2`		(The first event will always be a truck arrival at t = 1.)
`240`		(w_1 = 240 pounds)
`50`		(w_2)
	`L 1 1`	(program loads cargo item 1 (240 lbs.) to seat position 1 at 35 inches)
`2`		(event at time = 2, no truck can arrive while unloaded containers remain)
	`L 2 3`	(cargo item 2 (50 lbs.) to seat position 3 at 48 inches)
`3`		(event at time = 3, nothing to load, waiting for a truck)
	`I`	(do nothing while waiting)
`4`		(event at time = 4, nothing to load, still waiting for a truck)
	`I`	
`5 2`		(at time = 5, another truck arrives with two containers)
`352`		(w_1 = 352 pounds)
`212`		(w_2)
	`M 3 2 L 1 3`	(move cargo from seat 3 to 2, load cargo item 1 (352 lbs.) to seat 3)
`6`		
	`M 2 4 L 2 2`	(move cargo from seat 2 to 4, load cargo item 2 (212 lbs.) to seat 2)
`7 1`		(at time = 7, another truck arrives with one container)
`500`		(w_1 = 500 pounds)
	`R`	(at least one Ready Condition satisfied)

Judged Responses

An ill-formed command produces a PRESENTATION ERROR. If the server has to wait longer than one second (real time) for your client to issue a command, the judged response is TIME LIMIT EXCEEDED. Any other problem with your client will result in a response of WRONG ANSWER.

Problem 7
Loadmaster (still continued)

Hints for Testing

You may construct a series of simulated events manually. Execute your client program with you, the programmer, acting as the server. Let standard input come from your command line environment. Supply correct responses to you program, keeping track of cargo containers and where they are loaded.

The server process used by the judges is available to the contestants. You may use the server in two ways. In the absence of a working client program, you, the programmer, can act as the client, typing input to the server. Use the following command:

`test7` *configurationFile transcriptFile*

To test your program and its conversation with the server, use the command:

`test7` *configurationFile transcriptFile sourceFile*

where:

- *configurationFile* is described below
- *transcriptFile* is a file that captures the server's inputs and outputs for later analysis. The server's inputs are your client's commands. Client commands are offset by two tab characters to distinguish the client vs. server dialog.
- *sourceFile* is the source code to your client. When specified, the `test7` script will call the compile script before executing your program.

You can produce a simulation to the server by supplying a configuration file in the following format:

The first three lines contain whitespace separated fields: line one contains MTOW, W_{fueled}, and A_{fueled}; line two contains threshold and D_{max}; and line three contains minCG, maxCG, and P.

The following P lines contain moment arm locations, expressed as whole number inches, for each of the P seats. The first 3 + P lines are exactly the information presented in response to a configuration query.

Following the airplane configuration information are truck arrival events of the form:

t_a C
w_1
...
w_C

where t_a is the arrival time in minutes of a truck at the head of the line for loading. The first arrival event must be at $t_a = 1$. C is always ≥ 1. C lines follow with one cargo weight per line. Subsequent arrival events must obey the condition $t_{a+1} \geq t_a + C$. The configuration for the above transcript is in the file *sample7.in*, and is repeated here. The comments to the right of the configuration file entries must not appear in the actual file; they are here for information only.

```
2200 1340 37      (MTOW W_fueled A_fueled)
90 3              (thresholdPercentage and D_max)
35.0 44.0 4       (minCG maxCG and P = 4 passenger seats)
35                (a_1 = 35.0 inches)
72                (a_2)
48                (a_3)
92                (a_4)
1 2               (at time = 1, a truck arrives with two containers, w_1 = 240, w_2 = 50)
240
50
5 2               (at time = 5, another truck arrives with two containers, w_1 = 352, w_2 = 212)
352
212
7 1               (at time = 7, another truck arrives with one container, w_1 = 500)
500
```

2012/2013 SOUTHERN CALIFORNIA REGIONAL
ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 8
Hypotenuse Numbers

The Pythagorean theorem states that the square of the length of the hypotenuse of a right triangle is equal to the sum of the squares of the other two legs of the triangle. This can be represented by the equation:

$$Z^2 = X^2 + Y^2$$

where Z is the length of the hypotenuse of the triangle and X and Y are the lengths of the other legs. If we restrict X, Y, and Z to be integers greater than zero, which numbers can be the length of a hypotenuse of a right triangle?

A famous result from number theory states that an integer can be the length of a hypotenuse of a right triangle if and only if it has at least one prime factor of the form $4k + 1$, where k is an integer greater than zero. Your team is to write a program that will test integers to determine if they are possible hypotenuse lengths.

Your program is to read a series of lines terminated by end of file. Each line will consist of an integer Z, $0 < Z \leq 2{,}147{,}483{,}647$.

For each line in the input your program is to print a line containing the input number, a single space, and either "yes" if the number can be a hypotenuse or "no" if it cannot be. No whitespace is to appear before the number or between the answer and the newline.

Sample Input

```
5
6
7
10
2147483647
2147483645
2147483642
2147483627
1927738275
1927738291
```

Output for the Sample Input

```
5 yes
6 no
7 no
10 yes
2147483647 no
2147483645 yes
2147483642 no
2147483627 yes
1927738275 yes
1927738291 no
```

2012/2013 SOUTHERN CALIFORNIA REGIONAL
ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 9
Destination North America

Twenty-one slots at the ICPC World Finals are reserved for North American teams. Currently, each of the eleven regions is guaranteed that its winner will advance to the World Finals; the other slots are allocated by ICPC Headquarters. Generally, advancing a second or third team is based on fuzzy logic that includes things like the number of teams participating from a region and/or regional growth.

The 2011/2012 North American World Finals Teams participated in an Invitational contest prior to the World Finals. The problem set for this contest was more World Finals-like in difficulty than typical regional contests. (A link to the problem set is on the Regional Facebook page.) The group who organized this has proposed replacing the current allocation methodology with a new process, as follows:

> The top five ranked schools from each region would be invited to send a team to the North American Semi-Finals contest—a total of 55 teams. This would be a multi-day event that includes speakers and activities, similar to but shorter than the World Finals.
>
> The winners of the Semi-Finals would advance to the World Finals as follows:
> - The top-ranked school from each region would advance.
> - The remaining slots would awarded based on placement at the Semi-Finals, irrespective of region.

The goal of this proposal is to create a Semi-Finals that is a destination in and of itself; and, advances stronger North American teams to the World Finals.

The implications of this proposal include the following:
- The team that wins a Regional may not advance to the World Finals (assuming it places lower at the Semi-Finals than another team from its region).
- A single region may send five teams to the World Finals (if they all rank high enough overall).
- The 51st-ranked team may go to the World Finals (if the teams from its region rank 51st–55th at the Semi-Finals).

Your team is to write a program that takes as input the ranking of 55 teams at the Semi-Finals and produces as output a list of the 21 teams that would be invited to the World Finals if this process were in place.

Input to your program will consist of one or more sets of data. Each data set represents the results of a semi-final contest in rank order. Each data set consists of a contest identifier, followed by a comma, followed by 55 team identifiers separated by commas. Team identifiers consist of a single alphabetic character regional identifier, a hyphen, and a team name that may include white space. No team identifier will span more than one line, although the input for a given contest data set will span multiple lines. Lines will not exceed 80 characters and will always break at a comma. A new data set always begins on a new line.

For each input set, your program is to print the contest identifier on a line, followed by the 21 team identifiers of the advancing teams, one per line, in the same order as the teams appear in the input list. No leading or trailing whitespace is to appear on an output line.

NOTES:
1. ACM-ICPC has determined that this proposal could not be implemented without a restructuring of the North American region. This means that, if it or something similar is implemented, it will take several years to complete.
2. The sponsoring entity is going to try to continue to hold the Invitational.

Problem 9
Destination North America (continued)

Sample Input

```
2011-2012,A-team 1,A-team  2,A-team 3,A-team  4,A-team   5,B-team 1,B-team  2,
B-team 3,B-team  4,B-team   5,C-team 1,C-team  2,C-team 3,C-team  4,
C-team   5,D-team 1,D-team  2,D-team 3,D-team  4,D-team   5,E-team 1,
E-team  2,E-team 3,E-team  4,E-team   5,F-team 1,F-team  2,F-team 3,F-team  4,
F-team   5,G-team 1,G-team  2,G-team 3,G-team  4,G-team   5,H-team 1,H-team  2,
H-team 3,H-team  4,H-team   5,J-team 1,J-team  2,J-team 3,J-team  4,
J-team   5,K-team 1,K-team  2,K-team 3,K-team  4,K-team   5,L-team 1,
L-team  2,L-team 3,L-team  4,L-team   5
```

Output for the Sample Input

```
2011-2012
A-team 1
A-team  2
A-team 3
A-team  4
A-team   5
B-team 1
B-team  2
B-team 3
B-team  4
B-team   5
C-team 1
C-team  2
C-team 3
D-team 1
E-team 1
F-team 1
G-team 1
H-team 1
J-team 1
K-team 1
L-team 1
```

2012/2013 SOUTHERN CALIFORNIA REGIONAL
ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 10
Work Excess

Many government employees are paid "once a month."

Months do not have the same number of work days, and a given month may have a different number of work days from year to year. It is typical for an agency to assign monthly pay periods that do not necessarily align with the start and end dates of calendar months. This usually results in pay periods with 21 or 22 work days, based on working Monday through Friday.

Some employees have work weeks that do not span Monday to Friday (e.g., medical personnel, law enforcement, computer operations, and facilities personnel). These employees may work more or fewer days in any given pay period than those on the standard work week. At a minimum of once a year, the work excess/work deficit days are reconciled, and employees are either compensated or required to "pay back" time. As can be seen in Table 1 below, a work excess for a pay period is shown as a positive number of days, and a work deficit is shown as a negative number of days.

Monday - Friday	Sun - Thu	Tu-Sat	W-Sun	Th-M	F-Tu	Sat - W
Month 1: January 1 - January 30	Number of Days Worked					
21 working days (M-F)	22	20	21	22	22	22
Work Excess/Deficit	+1	-1	0	+1	+1	+1
Month 2: January 31 - February 28						
21 working days (M-F)	21	21	20	20	21	21
Work Excess/Deficit	0	0	-1	-1	0	0
Month 3: March 1 - March 30						
22 working days (M-F)	22	22	22	21	20	21
Work Excess/Deficit	0	0	0	-1	-2	-1

Table 1. Table showing work excess/deficit examples for three pay periods.

You work for such a government agency. They would like your team to develop a program that will produce the work excess for all work schedules in a list of monthly pay periods. Each pay period is named for the month that the majority of the pay period falls in.

Input to your program will be a series of lines representing single pay periods. Each line will contain the start and end dates for a pay period (*YYYYMMDD*), and the day of the week of the start date represented as a number: 1 = Sunday, 2 = Monday, ..., 7 = Saturday. Years will range from 1980 through 2040. Values will be separated from each other by commas. Pay periods do not span calendar years.

For each pay period, your program is to print a line that contains:

- the name of the pay period month followed by a space,
- followed by the four-digit year followed by a space,
- followed by seven numbers, separated by commas without embedded whitespace. The first of the seven numbers is to be the number of working days in the pay period for Monday through Friday schedules. The next six numbers will be the work excess/work deficit total for the other six schedules, in the order shown in the example above. Plus and minus signs are to be printed only to indicate a non-zero excess or deficit, respectively.

No leading or trailing whitespace is to appear on an output line.

Problem 10
Work Excess (continued)

Sample Input

```
20060101,20060130,1
20060301,20060330,4
```

Output for the Sample Input

```
January 2006 21,+1,-1,0,+1,+1,+1
March 2006 22,0,0,0,-1,-2,-1
```