2014/2015 SOUTHERN CALIFORNIA REGIONAL
ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 1
Spreadsheet Shuffle

Your company has just been acquired by Giganto Corporation, and your team has been assigned to help convert some experimental data to the standard Giganto format. The data to be converted are stored in spreadsheets. The first row of each spreadsheet has titles, and the remaining rows have data. Each field contains a string of one or more alphanumeric or space characters. No data fields are missing or empty. A single space is a valid data value. The data have been saved as comma-separated values.

Another team has analyzed the files and added two rows to the beginning of each file which are the instructions for how to convert the file. The resulting files have the following format:

- The first row (line) contains the new description for each column.
- The second row has an integer in each field which indicates what column that field will be moved to. Input and output columns are numbered left-to-right starting at 1. A '3', for example, in column 2 means make column 2 in the input be column 3 in the output. A '0' means delete this column. Each output column number will be specified exactly once. There will never be an output file with no output columns. No integer except '0' will appear more than once. The greatest number in the row is the number of columns in the output and hence the number of fields in each output row.
- The third row is the old header which will not appear in the new file.
- The fourth and subsequent rows contain the data.

Input lines are at most 80 characters long. Input is terminated by end-of-file.

Your program is to print the converted spreadsheet as comma-separated values, one row per line. Print the new header line followed by the data lines in new column order. Spaces are significant. No extra characters are to appear on an output line.

Sample Input

```
result,input,PI, ,test,Y
4,3,2,0,1,0
col a,col b,col c,col d,col e,col f
a2,b2,c2,d2,e2,f2
43, b3 or not , ,d3,e3,f3
```

Output for the Sample Input

```
test,PI,input,result
e2,c2,b2,a2
e3, , b3 or not ,43
```

2014/2015 SOUTHERN CALIFORNIA REGIONAL
ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 2
Karate Hex

Karate Hex is a game played by two players who alternate playing offense and defense in multiple rounds. In each round the player playing defense holds a hexagonal board with both hands while the player on offense, with a single well-targeted kick, tries to break the board in as many fragments as possible. The number of fragments that the board breaks into determines the score earned by the offensive player.

Boards of different sizes may be used in different rounds but all boards are in the shape of a regular hexagon that is made out of smaller hexagonal cells. Each of the six corners of the board is painted black and the rest of the cells are white. For example, Figure 1 shows a board of size 5.

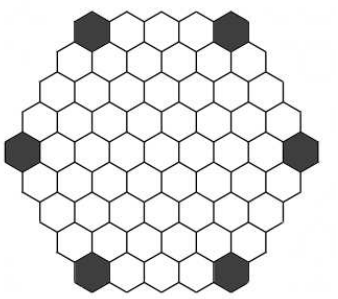


Figure 1. Karate Hex board of size 5.

It is a little bit of a challenge to represent hexagonal boards in a rectangular coordinate system, such as a text file. One way to do it is to use spaces between characters on the same row and to offset every other row by one character. The size 5 board above can be encoded in a text file like this:

```
    B W W W B
   W W W W W W
  W W W W W W W
 W W W W W W W W
B W W W W W W W B
 W W W W W W W W
  W W W W W W W
   W W W W W W
    B W W W B
```

The boards have grooves around the cells so that they will always break in such a way that the cells remain whole. Figure 2 shows what a broken board of size 4 might look like.

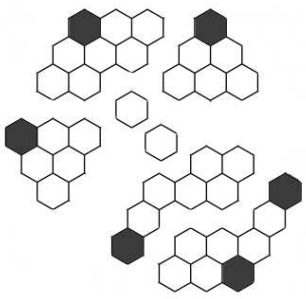
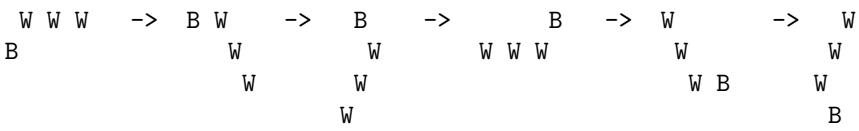


Figure 2. Broken Karate Hex board of size 4.

Problem 2
Karate Hex (continued)

Note that the orientation of a board fragment is not always obvious. With 60-degrees rotation we can generate a total of six possible orientations:



Judging in Karate Hex tournaments is often complicated because board fragments are flying all over the place and may be either lost or mixed up with fragments from other games. The International Mixed Martial Arts And Board Games Federation (IMMAABGF) has commissioned a computer program to help with the judging of the game. Given a pile of board fragments the program must determine whether they together form a regulation Karate Hex board.

The input to the program will consist of the different fragments separated by single blank lines. (Note that the text lines defining a fragment may start with space characters. Depending on the number of lines needed to define a fragment, the first line may start with an odd or even number of space characters, but the formatting of the rest of the lines will be consistent with the first one.) The last fragment will be followed by the end-of-file.

Your program is to print a single line with only the word "yes" or the word "no", depending on whether or not the fragments can be assembled into a regulation Karate Hex board. No leading or trailing whitespace is to appear on the output line.

Regulation IMMAABGF boards have sizes between 2 and 6, inclusive. You can also assume that the number of fragments in the input is not more than 12, as the official Karate Hex rules state that boards that break into more than 12 fragments are considered defective and the round is replayed.

Problem 2
Karate Hex (still continued)

Sample Input

```
  B W W
 W W W
W W

  B
 W W
W W W

W

W

W W B
 W W
  W

B W W
     W W
      W W

       B
      W
 W W W
W W B
```

Output for the Sample Input

```
yes
```

2014/2015 SOUTHERN CALIFORNIA REGIONAL
ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 3
Locked Treasure

A group of n ($1 \leq n \leq 30$) bandits hid their stolen treasure in a room. The treasure needs to be locked away until there is a need to retrieve it. Since the bandits do not trust each other, they wanted to ensure that at least m ($1 \leq m \leq n$) of the bandits must agree in order to retrieve the treasure. They have decided to place multiple locks on the door such that the door can be opened if and only if all the locks are opened. Each lock may have up to n keys, distributed to a subset of the bandits. A group of bandits can open a particular lock if and only if someone in the group has a key to that lock. Given n and m, how many locks are needed such that if the keys to the locks are distributed to the bandits properly, then every group of bandits of size at least m can open all the locks, and no smaller group of bandits can open all the locks? How many keys will each bandit need to have?

For example, if $n = 3$ and $m = 2$, only three locks are needed—keys to lock 1 can be given to bandits 1 and 2, keys to lock 2 can be given to bandits 1 and 3, and keys to lock 3 can be given to bandits 2 and 3. No single bandit can open all the locks, but any pair of bandits can open all the locks.

The first line of input contains a positive integer indicating the number of cases to follow. Each case is specified by the two integers n and m on one line, separated by a single space.

For each case, print a line containing the number of locks needed, a single space, and the number of keys each bandit will need. No leading or trailing whitespace is to appear on an output line.

Sample Input

```
3
3 2
5 1
5 3
```

Output for the Sample Input

```
3 2
1 1
10 6
```

2014/2015 SOUTHERN CALIFORNIA REGIONAL
ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 4
Rank Order

Your team has been retained by the director of a competition who supervises a panel of judges. The competition asks the judges to assign integer scores to competitors—the higher the score, the better. Although the event has standards for what score values mean, each judge is likely to interpret those standards differently. A score of 100, say, may mean different things to different judges.

The director's main objective is to determine which competitors should receive prizes for the top positions. Although absolute scores may differ from judge to judge, the director realizes that relative rankings provide the needed information—if two judges rank the same competitors first, second, third, ... then they agree on who should receive the prizes.

Your team is to write a program to assist the director by comparing the scores of pairs of judges. The program is to read two lists of integer scores in competitor order and determine the highest ranking place (first place being highest) at which the judges disagree.

Input to your program will be a series of score list pairs. Each pair begins with a single integer giving the number of competitors N, $1 < N < 1000$. The next N integers are the scores from the first judge in competitor order. These are followed by the second judge's scores—N more integers, also in competitor order. Scores are in the range 0 to 100,000 inclusive. Values are separated from each other by one or more spaces and/or newlines. No input line will exceed 80 characters. The last score list pair is followed by the end-of-file.

For each score pair, print a line with the integer representing the highest-ranking place at which the judges do not agree. If the judges agree on every place, print a line containing only the word "agree". No leading or trailing whitespace is to appear on an output line.

Sample Input

```
4 3 8 6 2
15 37 17 3
8 80 60 40 20 10 30 50 70 160 100 120 80
20 60 90 135
```

Output for the Sample Input

```
agree
3
```

2014/2015 SOUTHERN CALIFORNIA REGIONAL
ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 5
Speed Trap

The Swamp County Sheriff's Department has stepped up its speed law enforcement, but it has a problem: years of changing speed signs, lack of signage repairs, and downright bad planning have led to contradictory speed markings. Many tickets issued are unenforceable because the cited driver was obeying the last posted speed limit rather than the intended limit. Your team is to write a program that will determine and report which streets have inconsistent speed markings.

Consider the map in Figure 1, as described by the sample input. A driver on 2nd Street approaching Cherry Avenue from the south sees a posted speed limit of 50 MPH, and turns right (east) onto Cherry; according to the marked signs, that driver can continue along at up to 50 MPH without exceeding the limit. However, a driver approaching the intersection of 2nd Street and Cherry Avenue from the north or west must obey a marked 30 MPH speed limit. The eastbound section of Cherry Avenue between 2nd Street and 1st Street has conflicting speed limit markings.

One other problem stymies Swamp County Deputy Sheriffs. Some road segments have portions without any posted speed limits. This usually occurs at roads inbound from neighboring counties or where pavement begins. Segments with significant lengths of unknown speed limits must be reported. In Figure 1, southbound 1st Street towards Ash Avenue lacks explicit Swamp County speed markings for a significant distance, and must be reported.

There are some important considerations for your solution:

- There is a short threshold distance that a street can have contradictory or missing speed limits, typically at intersections or where drivers can see the upcoming speed signs. For contradictory and unknown speed conditions less than or equal to a specified threshold, do not report any problem. For example, in Figure 1, southbound 1st Street south of Cherry Avenue does not have conflicting speed limits. The 10-foot portion that could have a speed limit of either 30 MPH or 50 MPH is less than the 50-foot threshold distance specified in the sample input.
- A road segment that has both conflicting AND unknown speed limits is considered conflicted.
- U-turns are illegal in Swamp County. The county does not have any cul-de-sacs.
- Swamp County traffic judges have decreed that each segment must be analyzed for all possible paths into that segment, including circling, to determine the speed limit.

Input to your program is a street map described as road segments. Each segment has beginning and ending points identfied by two numbers. One-way streets have only one segment between two points. Two-way streets have two separate segments with reversed beginning and end points. All numbers in the input are integers.

The first line of input is the threshold distance, in feet. Each remaining line contains a single road segment starting with three numbers separated by whitespace: beginning point number, end point number, and the length of the segment in feet. If any speed signs are present on the segment of road, each sign specification follows as whitespace then a *distance*,*limit* pair. The distance is expressed as feet from the beginning point of the segment. The limit is in miles per hour. Consecutive signs on a road segment are always listed in ascending distance from the beginning point. A road segment may have 0 or more pairs. Road segments are not guaranteed to be sorted. There will be at most 100 road segments.

For any street segments that have a speed conflict or an unknown limit, your program is to print a line containing the beginning point, exactly one space, the end point, exactly one space, then a single character for the problem. The lower case letter "c" denotes a conflict, and the lower case letter "u" denotes an unknown speed limit. Display problem segments in the order supplied in the input. No leading or trailing whitespace is to appear on an output line.

Problem 5
Speed Trap (continued)

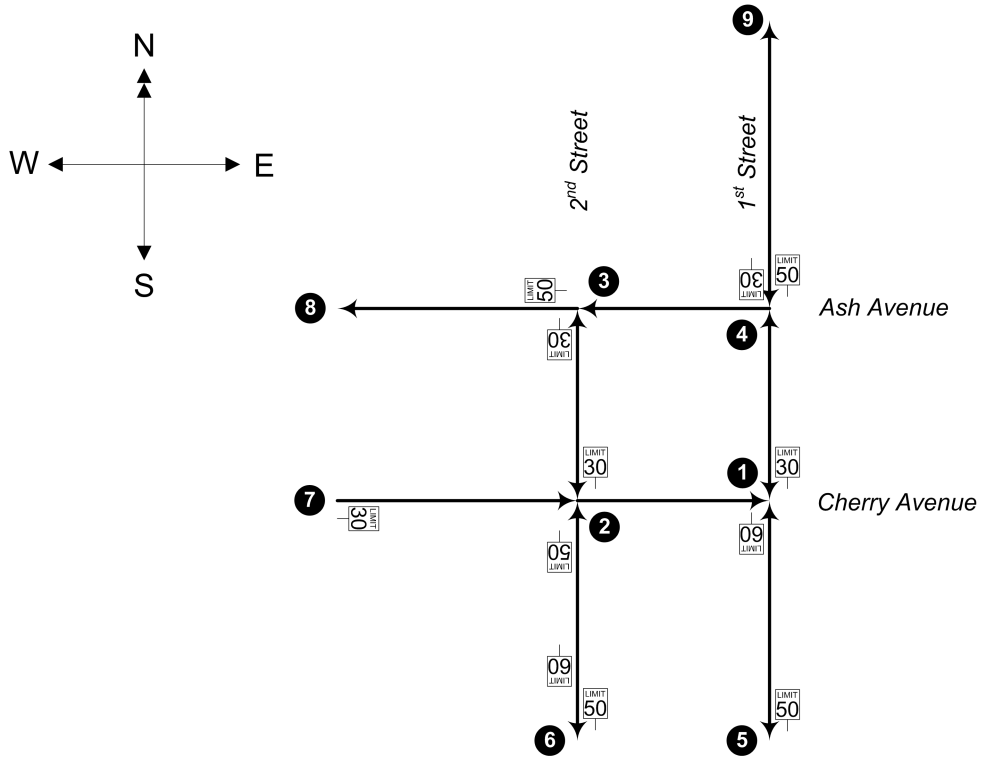


Figure 1. The street map described by the Sample Input.

Sample Input

```
50
1 5 500 10,60
1 4 400 10,30
2 6 500 10,50 300,60
2 3 400 10,30
2 1 400
3 2 400 10,30
3 8 500 10,50
4 1 400
4 3 400
4 9 600 10,50
5 1 500 0,50
6 2 500 10,50
7 2 500 0,30
9 4 600 580,30
```

Output for the Sample Input

```
2 1 c
9 4 u
```

2014/2015 SOUTHERN CALIFORNIA REGIONAL
ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 6
Bargain-Hunting Frequent Flyer

After many years, Wing-and-a-Prayer Airlines has decided to revamp its frequent-flyer bonus plan. Starting next year, they are going to switch from awarding miles based on flight distance to points based on fares paid. While this makes sense for the airline, as they will now be rewarding the people who spend the most, bargain hunters who made a habit of looking for extremely low fares that would earn many miles are disappointed.

One intrepid traveler wants to take maximum advantage of the existing program while he still can. He spends time on the WP Airlines web site looking for the most round-about routings for both one-way and round-trip travel that he can get for a given fare. However, the airline doesn't post the miles that would be earned for a given routing. He wants your team to write a program that will accept a routing as a series of airports and determine the total great-circle distance that he would fly.

Input to your program will be in two parts. The first part is a list of up to 3,000 airports that WP Airlines serves, one per line. Each line consists of the three-letter airport code, followed by four fields giving the latitude (degrees, minutes, seconds, and the letter "N" for North or "S" for South) and four more fields giving the longitude (degrees, minutes, seconds, and the letter "E" for East or "W" for West). Fields are separated from each other by single spaces. Numeric values may contain leading zeroes. This list will end with a line containing only three hyphens.

The remainder of the input is a series of routings given as lists of airport codes, one per line. Each routing is a series of airport codes separated by single spaces. The first airport code on the line is the point of departure. The remaining airport codes are the points where our traveler will make a connection. The last airport code on the line is the final destination (which might match the point of departure for a round-trip). The last routing will be followed by the end-of-file.

For each routing, your program is to print the total great-circle distance, with the final total rounded to the nearest integer. Treat the Earth as a sphere with a radius of 3958.75 miles.

No flight segment between two airports will be longer than 9,000 miles. No two airports will be at exactly the same longitude. No individual routing will contain more than thirty airports.

Sample Input

```
MAD 40 28 20 N 03 33 39 W
CMN 33 22 04 N 07 35 16 W
RAK 31 36 24 N 08 02 10 W
JFK 40 38 23 N 73 46 44 W
HNL 21 18 57 N 157 55 36 W
NRT 35 45 53 N 140 23 11 E
YYZ 43 40 38 N 79 37 50 W
FRA 50 01 35 N 08 32 35 E
LAS 36 04 49 N 115 09 08 W
LAX 33 56 33 N 118 24 29 W
---
LAS LAX HNL LAX LAS
LAX YYZ FRA CMN
LAX MAD RAK
```

Problem 6
Bargain-Hunting Frequent Flyer (continued)

Output for the Sample Input

```
5578
7526
6494
```

2014/2015 SOUTHERN CALIFORNIA REGIONAL
ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 7
Dump TISC

Titanic Industries is about to retire some aging process controllers that run on forty-year-old hardware. Unfortunately the source code has been lost. Your team is to write a program that will read a load module and produce a formatted dump. The TISC manual that describes the hardware architecture and load module format is attached.

Input to your program is a load module as described in the manual. The numbers are separated by whitespace and the module is terminated by end of file.

Your program is to print a representation of all 1000 (0–999) words of memory, one word per line. Each line starts with the four digit decimal location (0000–0999), a single space, and the four digit decimal value of the word in that location. If the word is a legal TISC instruction, print a single space followed by the instruction exactly as seen in the Op Code Summary section of the manual with no trailing spaces. To save space, if the value of a word is the same as the previous word, print a line containing only the string "same as above", then skip printing until a location has a different value. Always print the value of the last location (0999).

Sample Input

```
2800 14 2800 15 1100 14 1200 15 4127 8 2100 16 0 0 0 0 0 1 2 3 3 4 4 4 4
```

Output for the Sample Input

```
0000 2800 store regd
0001 0014
0002 2800 store regd
0003 0015
0004 1100 load rega
0005 0014
0006 1200 load regb
0007 0015
0008 4127 bif inca decb bnz
0009 0008
0010 2100 store rega
0011 0016
0012 0000 halt
same as above
0017 0001
0018 0002
0019 0003
same as above
0021 0004
same as above
0025 0000 halt
same as above
0999 0000 halt
```

TISC Manual

Principles of Operation

TISC is a decimal machine. Each word is four decimal digits and is unsigned. The range that can be represented is from 0 to 9999. 9999 + 1 yields 0 and 0 - 1 yields 9999. Memory consists of 1000 four digit words. The high order digit of the PC or a memory address is ignored.

The system has three registers (A, B, and C), an I/O register (D) and the program counter (PC). The A and B register may be incremented or decremented by one. When the D register is loaded from memory, the memory value is sent to the output latches, which are connected to the instrument being controlled. When the D register is stored into memory, the value is obtained from the input latches, also connected to the instrument. Successive stores from D will not, in general, obtain the same value. When a branch instruction results in a branch, the address of the instruction following the branch is placed in the C register. This is useful for subroutine linkage.

Instructions consist of two words: INSTR and ADDR. The high order digit of INSTR is the op code. PC contains the address of INSTR and ADDR is at PC+1. INSTR may be at an odd or even address.

HALT	0000	Put ADDR into PC and halt the processor.		
	ADDR	When the CONTINUE button is pressed, execution will resume at ADDR		
LOAD	1r00	Load register from ADDR		
	ADDR	r:	1	register A
			2	register B
			4	register C
			8	register D
STORE	2r00	Store register into ADDR		
	ADDR	r:	1	register A
			2	register B
			4	register C
			8	register D
BIF	4abc	increment/decrement and branch		
	ADDR	a:	0	leave A unchanged
			1	increment A by 1
			2	decrement A by 1
		b:	0	leave B unchanged.
			1	increment B by 1
			2	decrement B by 1
		c:	0	do not branch
			1	branch to ADDR if a = 0
			3	branch to ADDR if a != 0
			5	branch to ADDR if b = 0
			7	branch to ADDR if b != 0
			8	always branch to ADDR

Branch conditions are evaluated after any increments and decrements are done. If a branch is taken, the address of the instruction following the branch is placed in register C.

Other op codes, or r, a, b, or c values other than those specified above are illegal and may cause undefined operations. On the prototype, they may cause the instruction decoder to catch fire.

Load Module Format

The code to be loaded and run consists of a series of possibly signed 1 to 4 digit decimal numbers. Memory is set to all zeroes and then the code will be loaded starting at location 0. A negative number, for example -237, resets the loading address for the next positive number to its negative, in this case 237. Program execution starts at 0.

TISC Op Code Summary

0000 halt
1100 load rega
1200 load regb
1400 load regc
1800 load regd
2100 store rega
2200 store regb
2400 store regc
2800 store regd
4000 bif never
4001 bif az
4003 bif anz
4005 bif bz
4007 bif bnz
4008 bif always
4010 bif incb never
4011 bif incb az
4013 bif incb anz
4015 bif incb bz
4017 bif incb bnz
4018 bif incb always
4020 bif decb never
4021 bif decb az
4023 bif decb anz
4025 bif decb bz
4027 bif decb bnz
4028 bif decb always
4100 bif inca never
4101 bif inca az
4103 bif inca anz
4105 bif inca bz
4107 bif inca bnz
4108 bif inca always
4110 bif inca incb never
4111 bif inca incb az
4113 bif inca incb anz
4115 bif inca incb bz
4117 bif inca incb bnz
4118 bif inca incb always
4120 bif inca decb never
4121 bif inca decb az
4123 bif inca decb anz
4125 bif inca decb bz
4127 bif inca decb bnz
4128 bif inca decb always
4200 bif deca never
4201 bif deca az
4203 bif deca anz
4205 bif deca bz
4207 bif deca bnz
4208 bif deca always
4210 bif deca incb never
4211 bif deca incb az
4213 bif deca incb anz
4215 bif deca incb bz
4217 bif deca incb bnz
4218 bif deca incb always
4220 bif deca decb never
4221 bif deca decb az
4223 bif deca decb anz
4225 bif deca decb bz
4227 bif deca decb bnz
4228 bif deca decb always

2014/2015 SOUTHERN CALIFORNIA REGIONAL
ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 8
Deep Pockets Soccer

Professional soccer teams have eleven players on the field: one *goalkeeper*, and ten field players categorized as *defenders*, *midfielders*, and *forwards*. Different playing styles utilize different systems, but common sense and tradition dictate that a valid system has one goalkeeper, at least three defenders, at least three midfielders, and at least one forward. Thus, both 1–4–4–2 (1 goalkeeper, 4 defenders, 4 midfielders, 2 forwards) and 1–4–5–1 are valid team systems, but 1–9–0–1 is not.

A famous billionaire has just purchased a franchise in the National League of Soccer (NLS) and wants to invest in a team capable of winning the championship. Money is no object and he has given the team manager a blank check to buy eleven new starting players. The manager is free to use any valid team system, but he must buy all eleven players and no more.

Unfortunately, NLS regulations put a limit on how much money each team can spend on new players every year, so the manager must fit into a budget of N thousand dollars. Unable to just go and buy all the most expensive players on the market, the manager decided to buy the team that maximizes the formula

$$Z = S - I \tag{1}$$

where S is defined as the sum of the prices of all eleven players, and I is the *imbalance* of the team defined as:

$$I = |g + d - m| + |m - f| \tag{2}$$

where:
g is the price of the goalkeeper,
d is the total price of all defenders,
m is the total price of all midfielders, and
f is the total price of all forwards.

Your team is to write a program that will provide the manager with information about the optimal team configuration (that maximizes Z) given the players currently available and the annual dollar limit.

Input to the program will be a series of lines of not more than 80 columns. The first line contains the number of players K ($K \leq 40$) currently available on the market. The next K lines each describe an available player. The player information lines will contain three fields: a single character describing the position ("G" for goalkeeper, "D" for defender, "M" for midfielder, or "F" for forward), the price of the player in integer thousands of dollars (not to exceed 300,000), and the name of the player (a string of printable ASCII characters, excluding whitespace). Fields will be separated from each other by one or more spaces. No player will be listed more than once.

The remaining input lines will each have a dollar limit N, also given as an integer value representing thousands of dollars. For each value of N your program should find the best possible team for that budget. Should there be multiple team configurations that are optimal, choose the one that costs the least. N will not exceed one million.

The output should have one line for each input value of N. Each output line should have two numbers separated by a space: the Z score of the best team followed by the cost S. If no team can be assembled within the given budget, then Z and S should be printed as zeroes. No leading or trailing whitespace is to appear on an output line.

Problem 8
Deep Pockets Soccer (continued)

Sample Input

```
17
G 1500 G1
G 2000 G2
D 5000 D1
D 2300 D2
D 4000 D3
D 5600 D4
D 1000 D5
D 1400 D6
M 8000 M1
M 3000 M2
M 4000 M3
M 4000 M4
M 5000 M5
M 6000 M6
F 10000 F1
F 5000 F2
F 8000 F3
36000
36500
50000
100000
```

Output for the Sample Input

```
0 0
24400 36200
47800 49900
56800 60900
```

2014/2015 SOUTHERN CALIFORNIA REGIONAL
ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 9
Anagram Pyramids

Back in the 20th century anagram puzzles were often found in the back pages of many newspapers. Although they never reached the level of popularity of Sudoku puzzles, they were still a reliable means for killing a few minutes. One type of such puzzles was the anagram pyramid, which consisted of a sequence of words stacked on top of each other. The word at the bottom had N letters, the second-to-last word had $N - 1$ letters, the third-to-last word had $N - 2$ letters, and so on. Each word was formed by removing one letter from the previous word and shuffling the remaining letters. Here is one example of an anagram pyramid:

```
PIN
SNIP
PAINS
PIANOS
```

Mr. Zino Ponzi, a retired financier, was reminiscing of the good old times one day when he got the idea of creating a few anagram puzzles to share with his friends at the Ziggurat Retirement Home. Although he loved constructing anagram pyramids by hand, often he would get stuck unable to think of a suitable word in the middle of the pyramid. Finally, he decided to hire a student programming team to write a program to make things easier. He didn't want to kill all the fun of doing it by hand, so he only wanted the program to tell him whether an anagram pyramid was possible for a pair of top and bottom words.

Input to your program will consist of a dictionary of N words ($N < 1{,}000{,}000$), followed by M pairs ($M < 100$) of top and bottom words in the following format:

N
word1
...
word$N$
M
top1 bottom1
...
top$M$ bottom$M$

All input will begin in the first column. Words in the top and bottom word pairs will be separated from each other by single spaces. The top word will be shorter than the bottom word. The top and bottom words will appear in the dictionary. No word will be longer than thirty letters. Words will contain only upper-case letters A through Z.

The output is to consist of M lines, one for each pair of top and bottom words, with each line containing only the word "yes" or "no" depending on whether an anagram pyramid was possible using the given input dictionary. No leading or trailing whitespace is to appear on an output line.

Problem 9
Anagram Pyramids (continued)

Sample Input

```
8
SPAIN
PIANOS
PEN
SNIP
PAINS
SNIPER
PIN
PINE
2
PIN PIANOS
PEN SNIPER
```

Output for the Sample Input

```
yes
no
```