11034 Ferry Loading IV

Before bridges were common, ferries were used to transport cars across rivers. River ferries, unlike their larger cousins, run on a guide line and are powered by the river's current. Cars drive onto the ferry from one end, the ferry crosses the river, and the cars exit from the other end of the ferry.

There is an l-meter-long ferry that crosses the river. A car may arrive at either river bank to be transported by the ferry to the opposite bank. The ferry travels continuously back and forth between the banks so long as it is carrying a car or there is at least one car waiting at either bank. Whenever the ferry arrives at one of the banks, it unloads its cargo and loads up cars that are waiting to cross as long as they fit on its deck. The cars are loaded in the order of their arrival; ferry's deck accommodates only one lane of cars. The ferry is initially on the left bank where it broke and it took quite some time to fix it. In the meantime, lines of cars formed on both banks that await to cross the river.

Input

The first line of input contains c, the number of test cases. Each test case begins with l, m. m lines follow describing the cars that arrive in this order to be transported. Each line gives the length of a car (in centimeters), and the bank at which the car arrives ('`left`' or '`right`').

Output

For each test case, output one line giving the number of times the ferry has to cross the river in order to serve all waiting cars.

Sample Input

```
4
20 4
380 left
720 left
1340 right
1040 left
15 4
380 left
720 left
1340 right
1040 left
15 4
380 left
```

```
720 left
1340 left
1040 left
15 4
380 right
720 right
1340 right
1040 right
```

Sample Output

```
3
3
5
6
```

630 Anagrams (II)

One of the preferred kinds of entertainment of people living in final stages of XX century is filling in the crosswords. Almost every newspaper and magazine has a column dedicated to entertainment but only amateurs have enough after solving one crossword. Real professionals require more than one crossword for a week. And it is so dull - just crosswords and crosswords - while so many other riddles are waiting out there. For those are special, dedicated magazines. There are also quite a few competitions to take part in, even reaching the level of World Championships. Anyway - a lot.

You were taken on by such a professional for whom riddle solving competing is just a job. He had a brilliant idea to use a computer in work not just to play games. Somehow anagrams found themselves first in the line. You are to write a program which searches for anagrams of given words, using a given vocabulary, tediously filled with new words by yours employer.

Input

The structure of input data is given below:

```
<number of words in vocabulary>
<word 1>
..............
<word N>
<test word 1>
................
<test word k>
END
```

`<number of words in vocabulary>` is an integer number $N < 1000$. `<word 1>` up to `<word N>` are words from the vocabulary. `<test word 1>` up to `<test word k>` are the words to find anagrams for. All words are lowercase (word END means end of data - it is NOT a test word). You can assume all words are not longer than 20 characters.

Output

For each `<test word>` list the found anagrams in the following way:

```
Anagrams for: <test word>
<No>) <anagram>
...............
```

`<No>` should be printed on 3 chars.

In case of failing to find any anagrams your output should look like this:

```
Anagrams for: <test word>
No anagrams for: <test word>
```

Sample Input

```
8
atol
lato
microphotographics
rata
rola
tara
tola
pies
tola
kola
aatr
photomicrographics
END
```

Sample Output

```
Anagrams for: tola
  1) atol
  2) lato
  3) tola
Anagrams for: kola
No anagrams for: kola
Anagrams for: aatr
  1) rata
  2) tara
Anagrams for: photomicrographics
  1) microphotographics
```

755 487–3279

Businesses like to have memorable telephone numbers. One way to make a telephone number memorable is to have it spell a memorable word or phrase. For example, you can call the University of Waterloo by dialing the memorable TUT–GLOP. Sometimes only part of the number is used to spell a word. When you get back to your hotel tonight you can order a pizza from Gino's by dialing 310–GINO. Another way to make a telephone number memorable is to group the digits in a memorable way. You could order your pizza from Pizza Hut by calling their "three tens" number 3–10–10–10.

The standard form of a telephone number is seven decimal digits with a hyphen between the third and fourth digits (e.g. 888–1200). The keypad of a phone supplies the mapping of letters to numbers, as follows:

A, B, and C map to 2
D, E, and F map to 3
G, H, and I map to 4
J, K, and L map to 5
M, N, and O map to 6
P, R, and S map to 7
T, U, and V map to 8
W, X, and Y map to 9

There is no mapping for Q or Z. Hyphens are not dialed, and can be added and removed as necessary. The standard form of TUT–GLOP is 888–4567, the standard form of 310–GINO is 310–4466, and the standard form of 3–10–10–10 is 310–1010.

Two telephone numbers are equivalent if they have the same standard form. (They dial the same number.)

Your company is compiling a directory of telephone numbers from local businesses. As part of the quality control process you want to check that no two (or more) businesses in the directory have the same telephone number.

Input

The input will consist of one case. The first line of the input specifies the number of telephone numbers in the directory (up to 100,000) as a positive integer alone on the line. The remaining lines list the telephone numbers in the directory, with each number alone on a line. Each telephone number consists of a string composed of decimal digits, uppercase letters (excluding Q and Z) and hyphens. Exactly seven of the characters in the string will be digits or letters.

Output

Generate a line of output for each telephone number that appears more than once in any form. The line should give the telephone number in standard form, followed by a space, followed by the number of times the telephone number appears in the directory. Arrange the output lines by telephone number in ascending lexicographical order. If there are no duplicates in the input print the line:

```
No duplicates.
```

Sample Input

```
12
4873279
ITS-EASY
888-4567
3-10-10-10
888-GLOP
TUT-GLOP
967-11-11
310-GINO
F101010
888-1200
-4-8-7-3-2-7-9-
487-3279
```

Sample Output

```
310-1010 2
487-3279 4
888-4567 3
```

700 Date Bugs

There are rumors that there are a lot of computers having a problem with the year 2000. As they use only two digits to represent the year, the date will suddenly turn from 1999 to 1900. In fact, there are also many other, similar problems. On some systems, a 32-bit integer is used to store the number of seconds that have elapsed since a certain fixed date. In this way, when 2^{32} seconds (about 136 Years) have elapsed, the date will jump back to whatever the fixed date is.

Now, what can you do about all that mess? Imagine you have two computers C_1 and C_2 with two different bugs: One with the ordinary Y2K-Bug (i.e. switching to $a_1 := 1900$ instead of $b_1 := 2000$) and one switching to $a_2 := 1904$ instead of $b_2 := 2040$. Imagine that the C_1 displays the year $y_1 := 1941$ and C_2 the year $y_2 := 2005$. Then you know the following (assuming that there are no other bugs): the real year can't be 1941, since, then, both computers would show the (same) right date. If the year would be 2005, y_1 would be 1905, so this is impossible, too. Looking only at C_1, we know that the real year is one of the following: 1941, 2041, 2141, etc. We now can calculate what C_2 would display in these years: 1941, 1905, 2005, etc. So in fact, it is possible that the actual year is 2141.

To calculate all this manually is a lot of work. (And you don't really want to do it each time you forgot the actual year.) So, your task is to write a program which does the calculation for you: find the first possible real year, knowing what some other computers say (y_i) and knowing their bugs (switching to a_i instead of b_i). Note that the year a_i is definitely not after the year the computer was built. Since the actual year can't be before the year the computers were built, the year your program is looking for can't be before any a_i.

Input

The input file contains several test cases, in which the actual year has to be calculated. The description of each case starts with a line containing an integer n ($1 \leq n \leq 20$), the number of computers. Then, there is one line containing three integers y_i, a_i, b_i for each computer ($0 \leq a_i \leq y_i < b_i < 10000$). y_i is the year the computer displays, b_i is the year in which the bug happens (i.e. the first year which can't be displayed by this computer) and a_i is the year that the computer displays instead of b_i.

The input is terminated by a test case with $n = 0$. It should not be processed.

Output

For each test case, output the line "`Case #`*k*`:`", where k is the number of the situation. Then, output the line "`The actual year is` *z*`.`", where z is the smallest possible year (satisfying all computers and being greater or equal to $u = \max_{i=1}^{n} a_i$). If there is no such year less than 10000, output "`Unkown bugs detected.`".

Output a blank line after each case.

Sample Input

```
2
1941 1900 2000
2005 1904 2040
2
1998 1900 2000
1999 1900 2000
0
```

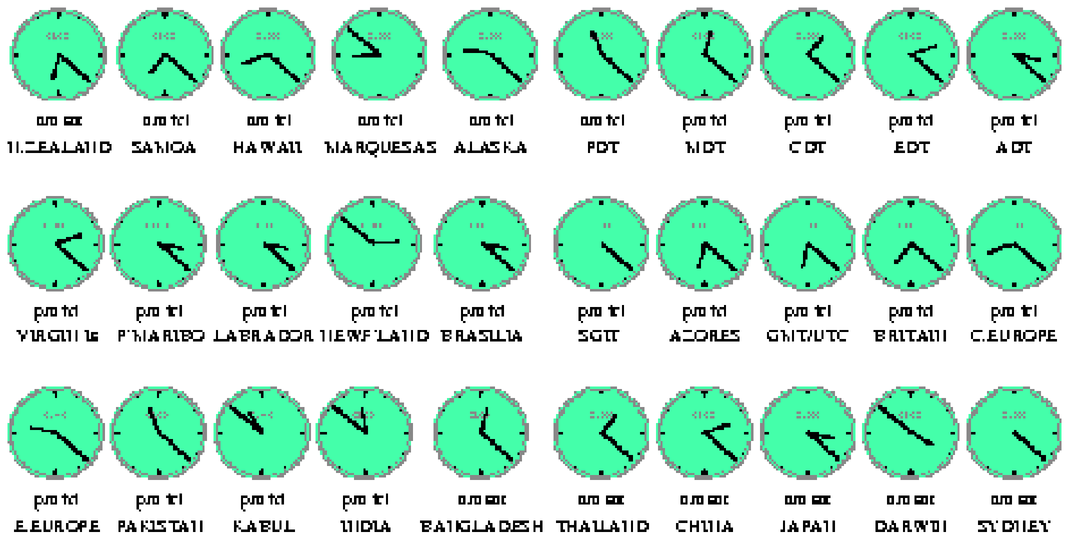
Sample Output

```
Case #1:
The actual year is 2141.

Case #2:
Unknown bugs detected.
```

10371 Time Zones

Prior to the late nineteenth century, time keeping was a purely local phenomenon. Each town would set their clocks to noon when the sun reached its zenith each day. A clockmaker or town clock would be the “official” time and the citizens would set their pocket watches and clocks to the time of the town — enterprising citizens would offer their services as mobile clock setters, carrying a watch with the accurate time to adjust the clocks in customer’s homes on a weekly basis. Travel between cities meant having to change one’s pocket watch upon arrival.



However, once railroads began to operate and move people rapidly across great distances, time became much more critical. In the early years of the railroads, the schedules were very confusing because each stop was based on a different local time. The standardization of time was essential to efficient operation of railroads.

In 1878, Canadian Sir Sanford Fleming proposed the system of worldwide time zones that we use today. He recommended that the world be divided into twenty-four time zones, each spaced 15 degrees of longitude apart. Since the earth rotates once every 24 hours and there are 360 degrees of longitude, each hour the earth rotates one-twenty-fourth of a circle or 15^o of longitude. Sir Fleming’s time zones were heralded as a brilliant solution to a chaotic problem worldwide.

United States railroad companies began utilizing Fleming’s standard time zones on November 18, 1883. In 1884 an International Prime Meridian Conference was held in Washington D.C. to standardize time and select the Prime Meridian. The conference selected the longitude of Greenwich, England as zero degrees longitude and established the 24 time zones based on the Prime Meridian. Although the time zones had been established, not all countries switched immediately. Though most U.S. states began to adhere to the Pacific, Mountain, Central, and Eastern time zones by 1895, Congress didn’t make the use of these time zones mandatory until the Standard Time Act of 1918.

Today, many countries operate on variations of the time zones proposed by Sir Fleming. All of China (which should span five time zones) uses a single time zone - eight hours ahead of Coordinated Universal Time (known by the abbreviation UTC - based on the time zone running through Greenwich at 0^o longitude). Russia adheres to its designated time zones although the entire country is on permanent Daylight Saving Time and is an hour ahead of their actual zones. Australia uses three time zones - its central time zone is a half-hour ahead of its designated time zone. Several countries in the Middle East and South Asia also utilize half-hour time zones.

Since time zones are based on segments of longitude and lines of longitude narrow at the poles, scientists working at the North and South Poles simply use UTC time. Otherwise, Antarctica would be divided into 24 very thin time zones!

Time zones have recently been given standard capital-letter abbreviations as follows:

UTC	Coordinated Universal Time
GMT	Greenwich Mean Time, defined as UTC
BST	British Summer Time, defined as UTC+1 hour
IST	Irish Summer Time, defined as UTC+1 hour
WET	Western Europe Time, defined as UTC
WEST	Western Europe Summer Time, defined as UTC+1 hour
CET	Central Europe Time, defined as UTC+1
CEST	Central Europe Summer Time, defined as UTC+2
EET	Eastern Europe Time, defined as UTC+2
EEST	Eastern Europe Summer Time, defined as UTC+3
MSK	Moscow Time, defined as UTC+3
MSD	Moscow Summer Time, defined as UTC+4
AST	Atlantic Standard Time, defined as UTC-4 hours
ADT	Atlantic Daylight Time, defined as UTC-3 hours
NST	Newfoundland Standard Time, defined as UTC-3.5 hours
NDT	Newfoundland Daylight Time, defined as UTC-2.5 hours
EST	Eastern Standard Time, defined as UTC-5 hours
EDT	Eastern Daylight Saving Time, defined as UTC-4 hours
CST	Central Standard Time, defined as UTC-6 hours
CDT	Central Daylight Saving Time, defined as UTC-5 hours
MST	Mountain Standard Time, defined as UTC-7 hours
MDT	Mountain Daylight Saving Time, defined as UTC-6 hours
PST	Pacific Standard Time, defined as UTC-8 hours
PDT	Pacific Daylight Saving Time, defined as UTC-7 hours
HST	Hawaiian Standard Time, defined as UTC-10 hours
AKST	Alaska Standard Time, defined as UTC-9 hours
AKDT	Alaska Standard Daylight Saving Time, defined as UTC-8 hours
AEST	Australian Eastern Standard Time, defined as UTC+10 hours
AEDT	Australian Eastern Daylight Time, defined as UTC+11 hours
ACST	Australian Central Standard Time, defined as UTC+9.5 hours
ACDT	Australian Central Daylight Time, defined as UTC+10.5 hours
AWST	Australian Western Standard Time, defined as UTC+8 hours

Given the current time in one time zone, you are to compute what time it is in another time zone.

Input

The first line of input contains N, the number of test cases. For each case a line is given with a time, and 2 time zone abbreviations. Time is given in standard a.m./p.m. format with midnight denoted '`midnight`' and noon denoted '`noon`' (12:00 a.m. and 12:00 p.m. are oxymorons).

Output

Assuming the given time is the current time in the first time zone, give the current time in the second time zone.

Note: that Standard am/pm notation means times are in this order: midnight, 12:01 a.m., ...11:59 a.m., noon, 12:01 p.m. ...11:59 p.m, midnight, and so on

Sample Input

```
4
noon HST CEST
11:29 a.m. EST GMT
6:01 p.m. CST UTC
12:40 p.m. ADT MSK
```

Sample Output

```
midnight
4:29 p.m.
12:01 a.m.
6:40 p.m.
```

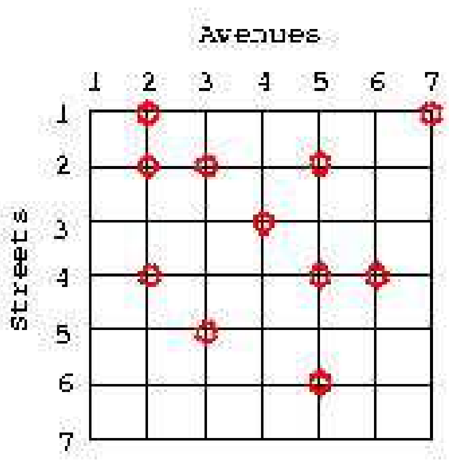
855 Lunch in Grid City

Grid city is a city carefully planned. Its street-map very much resembles that of downtown Manhattan in New York. Streets and avenues are orderly setup like a grid.

A group of friends living in Grid city decide to meet for lunch. Given that the group is dynamic, that is, its size may grow or shrink from time to time, they follow a written rule to determine the meeting point. It states that the meeting point is the one that minimizes the total distance the all group has to walk from their homes upto that point. If there is more than one candidate point, the rule imposes that the meeting point is the one corresponding to the smaller number for street and avenue. For large groups, this rule naturally avoids the usual long discussions that take place before aggreeing on a possible meeting point. For simplicity, consider that each person lives at a corner formed by a street and an avenue. You can also assume that the distance between two corners along one street or avenue is always one unit.

Your task is to suggest to the group their best meeting point (corner between a street and an avenue).

As an example, the following figure illustrates one such Crid city and the location of 11 friends. For this scenario, the best meeting point is street 3 and avenue 4. You can assume that streets and avenues are set and ordered as illustrated in this figure.



Please note that if we add another friend located at, say street 3 and avenue 5, making a total of 12 friends, then we would have two candidate meeting points, pairs (3,4) and (3,5). The rule clearly defines that street 3 and avenue 4 is the meeting point.

Given the size of a grid representing the Grid city and the locations of each person of the group of friends, your task is to determine the best meeting point following the rule of the group, as stated above.

Input

The first line of the input contains the number T of test cases, followed by T input blocks.

The first line of each test case consists of three positive numbers, the number of streets S, the number of avenues A (where $S \leq 1000$ and $A \leq 1000$), and the number of friends F (where $0 < F \leq 50000$). The following F input lines indicate the locations of the friends. A location is defined by two numbers, a street and an avenue, in this order.

Output

The output for each test case must list the best meeting point formatted as follows:

```
(Street: 3, Avenue: 4)
```

Each test case must be on a separate line.

Sample Input

```
2
2 2 2
```

```
1 1
2 2
7 7 11
1 2
1 7
2 2
2 3
2 5
3 4
4 2
4 5
4 6
5 3
6 5
```

Sample Output

```
(Street: 1, Avenue: 1)
(Street: 3, Avenue: 4)
```

11572 Unique Snowflakes

Emily the entrepreneur has a cool business idea: packaging and selling snowflakes. She has devised a machine that captures snowflakes as they fall, and serializes them into a stream of snowflakes that flow, one by one, into a package. Once the package is full, it is closed and shipped to be sold.

The marketing motto for the company is “bags of uniqueness.” To live up to the motto, every snowflake in a package must be different from the others. Unfortunately, this is easier said than done, because in reality, many of the snowflakes flowing through the machine are identical. Emily would like to know the size of the largest possible package of unique snowflakes that can be created. The machine can start filling the package at any time, but once it starts, all snowflakes flowing from the machine must go into the package until the package is completed and sealed. The package can be completed and sealed before all of the snowflakes have flowed out of the machine.

Input

The first line of input contains one integer specifying the number of test cases to follow. Each test case begins with a line containing an integer n, the number of snowflakes processed by the machine. The following n lines each contain an integer (in the range 0 to 10^9, inclusive) uniquely identifying a snowflake. Two snowflakes are identified by the same integer if and only if they are identical.

The input will contain no more than one million total snowflakes.

Output

For each test case output a line containing single integer, the maximum number of unique snowflakes that can be in a package.

Sample Input

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

Sample Output

```
3
```

10315 Poker Hands

A poker deck contains 52 cards - each card has a suit which is one of clubs, diamonds, hearts, or spades (denoted C, D, H, S in the input data). Each card also has a value which is one of 2, 3, 4, 5, 6, 7, 8, 9, 10, jack, queen, king, ace (denoted 2, 3, 4, 5, 6, 7, 8, 9, T, J, Q, K, A). For scoring purposes, the suits are unordered while the values are ordered as given above, with 2 being the lowest and ace the highest value.

A poker hand consists of 5 cards dealt from the deck. Poker hands are ranked by the following partial order from lowest to highest

High Card. Hands which do not fit any higher category are ranked by the value of their highest card. If the highest cards have the same value, the hands are ranked by the next highest, and so on.

Pair. 2 of the 5 cards in the hand have the same value. Hands which both contain a pair are ranked by the value of the cards forming the pair. If these values are the same, the hands are ranked by the values of the cards not forming the pair, in decreasing order.

Two Pairs. The hand contains 2 different pairs. Hands which both contain 2 pairs are ranked by the value of their highest pair. Hands with the same highest pair are ranked by the value of their other pair. If these values are the same the hands are ranked by the value of the remaining card.

Three of a Kind. Three of the cards in the hand have the same value. Hands which both contain three of a kind are ranked by the value of the 3 cards.

Straight. Hand contains 5 cards with consecutive values. Hands which both contain a straight are ranked by their highest card.

Flush. Hand contains 5 cards of the same suit. Hands which are both flushes are ranked using the rules for High Card.

Full House. 3 cards of the same value, with the remaining 2 cards forming a pair. Ranked by the value of the 3 cards.

Four of a kind. 4 cards with the same value. Ranked by the value of the 4 cards.

Straight flush. 5 cards of the same suit with consecutive values. Ranked by the highest card in the hand.

Your job is to compare several pairs of poker hands and to indicate which, if either, has a higher rank.

Input

The input file contains several lines, each containing the designation of 10 cards: the first 5 cards are the hand for the player named 'Black' and the next 5 cards are the hand for the player named 'White'.

Output

For each line of input, print a line containing one of:

```
Black wins.
White wins.
Tie.
```

Sample Input

```
2H 3D 5S 9C KD 2C 3H 4S 8C AH
2H 4S 4C 2D 4H 2S 8S AS QS 3S
2H 3D 5S 9C KD 2C 3H 4S 8C KH
2H 3D 5S 9C KD 2D 3H 5C 9S KH
```

Sample Output

```
White wins.
Black wins.
Black wins.
Tie.
```