acm International Collegiate Programming Contest 2004

South Central USA Regional Programming Contest

MisLED

Introduction:

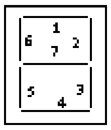
Rocky rolled over in his bed and looked at his LED alarm clock. He saw what was displayed and went back to sleep. Later, he woke up again and once again glanced at the alarm clock. Although he did not know if all the LED segments were working, he was able to determine with certainty the time. Could you?

Rocky's LED clock uses four 7-segment displays to show the time.

Each 7-segment display has seven bars on it, and displays different numbers by turning on different bars:

```
 _          _   _       _   _   _   _   _
| |    |    _|  _| |_| |_  |_    | |_| |_|
|_|    |   |_   _|   |  _| |_|   | |_|  _|
```

Using a bit to represent each segment, we can describe a display with seven bits. For example, if the segments are numbered as such:



Then the display:

```
|_
|
```

can be represented with bits:

0000111

And the numbers therefore could be represented by:

Number	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7
0	1	1	1	1	1	1	0
1	0	1	1	0	0	0	0
2	1	1	0	1	1	0	1
3	1	1	1	1	0	0	1
4	0	1	1	0	0	1	1
5	1	0	1	1	0	1	1

6	1	0	1	1	1	1	1
7	1	1	1	0	0	0	0
8	1	1	1	1	1	1	1
9	1	1	1	1	0	1	1

If a segment on the display is "burnt out", however, it will not display even though it should. For example, if the number three were to be displayed, but the segment corresponding with bit 7 were burnt out, the display would instead be:

```
 _
  |
 _|
```

Input:

The first line contains a single integer *n* indicating the number of data sets.

Each data set is a single line containing eight 7-bit strings representing the LED displays observed. The first four strings will represent the first time displayed and the next four strings will represent the second time displayed. All of the strings will be separated by a single space. Note: It is not given in the input which segments are burnt out.

Output:

For each data set, there will be exactly one line of output. This line will simply be the second time observed in the LED display, in the format HH:MM. The segments that are burnt out will be consistent between the first and second time displayed. There will only be one possible solution for each data set.

All times (for input and output) will be represented in standard (non-military) format, with no leading zeros (nine o'clock is 9:00, not 09:00).

Sample Input:

```
2
0110000 1101101 1111110 1111111 0000000 1111011 1111110 1111111
0000000 0000000 1000001 1111111 0100000 0000001 0000000 0110000
```

Sample Output:

```
9:08
12:11
```

acm International Collegiate Programming Contest 2004

South Central USA Regional Programming Contest

Relative Relatives

Introduction:

Today is Ted's 100th birthday. A few weeks ago, you were selected by the family to contact all of Ted's descendants and organize a surprise party. To make this task easier, you created an age-prioritized list of everyone descended from Ted. Descendants of the same age are listed in dictionary order.

The only materials you had to aid you were birth certificates. Oddly enough, these birth certificates were not dated. They simply listed the father's name, the child's name, and the father's exact age when the baby was born.

Input:

Input to this problem will begin with line containing a single integer *n* indicating the number of data sets. Each data set will be formatted according to the following description.

A single data set has 2 components:

1. *Descendant Count* - A line containing a single integer *X* (where 0 < *X* < 100) indicating the number of Ted's descendants.
2. *Birth Certificate List* - Data for *X* birth certificates, with one certificate's data per line. Each certificate's data will be of the format "`FNAME CNAME FAGE`" where:
   FNAME is the father's name.
   CNAME is the child's name.
   FAGE is the integer age of the father on the date of *CNAME*s birth.

Note:

- Names are unique identifiers of individuals and contain no embedded white space.
- All of Ted's descendants share Ted's birthday. Therefore, the age difference between any two is an integer number of years. (For those of you that are really picky, assume they were all born at the exact same hour, minute, second, etc... of their birth year.)
- You have a birth certificate for all of Ted's descendants (a complete collection).

Output:

For each data set, there will be *X*+1 lines of output. The first will read, "*`DATASET Y`*", where *Y* is 1 for the first data set, 2 for the second, etc. The subsequent *X* lines constitute your age-prioritized list of Ted's descendants along with their ages using the format "*`NAME AGE`*". Descendants of the same age will be listed in dictionary order.

Sample Input:

```
2
1
Ted Bill 25
4
Ray James 40
James Beelzebub 17
Ray Mark 75
Ted Ray 20
```

Sample Output:

```
DATASET 1
Bill 75
DATASET 2
Ray 80
James 40
Beelzebub 23
Mark 5
```

acm International Collegiate Programming Contest 2004

South Central USA Regional Programming Contest

Syntax Included

Introduction:

You are writing your first website and seem to be making your fair share of simple HTML syntax errors. You have decided to write an HTML parser to help you determine if your HTML code is syntactically correct based on the following condensed HTML definition:

HTML CODE	<HTML>***BODY***</HTML>
BODY	<BODY>***TEXT***</BODY>
TEXT	***STRING* | *STRING TEXT* | *TAG* | *TAG TEXT***
STRING	possibly empty string of printable characters other than '<' and '>')
TAG	***BOLD* | *ITALICS* | *LINK***
BOLD	<B>***TEXT***</B>
ITALICS	<I>***TEXT***</I>
LINK	<A HREF=***URL***>***TEXT***</A>
URL	http://***STRING***.com

Input:

The first line contains a single integer *n* indicating the number of data sets.

The following *n* lines each represent a data set and consists of up to 1000 characters. Spaces can be contained anywhere within the data set.

Note that all tags are case sensitive.

Output:

If the code is syntactically correct, the following string will be printed:

"Syntax Included"

Otherwise the following string will be printed:

"No Syntax Included"

Sample Input:

```
3
<HTML><BODY><B>Hello world!</B></BODY></HTML>
```

```
<HTML><BODY><B>Is this correct?</B></HTML>
<HTML><BODY><A HREF=http://www.go.com>link</A></BODY></HTML>
```

Sample Output:

```
Syntax Included
No Syntax Included
Syntax Included
```

acm International Collegiate Programming Contest 2004

South Central USA Regional Programming Contest

Choose Your Own Adventure

Introduction:

After reading the book *Tim and Marc Kill Kenny* about fifty zillion times, James decided he'd had it with choose-your-own-adventure stories. No matter what choices he made, it seemed like Kenny always fell down an abandoned mine shaft, got run over by a bus load of nuns, or was messily devoured by stray cats. James eventually found the page with the happy ending (where Kenny saves himself by trapping Tim and Marc between the pizza and the hungry programmers) by flipping through the book, but he can't figure out how to get there by following the rules. Luckily, he owns a C compiler...

Input:

Input to this problem will consist of a (non-empty) series of up to 100 data sets, each representing a choose-your-own-adventure story. Each data set will be formatted according to the following description, and there will be **no blank lines** separating data sets.

The first line contains a single integer *n* indicating the number of data sets.

A single data set has 2 components:

1. *Page Count* - A line containing a single integer *X*, where 1 < *X* < 100, indicating the number of pages in the story.
2. *Page List* - A sequence of *X* lines, each of which represents a page from the book. Each line has the following components separated from one another by single spaces:
    Line type - A single character indicating what type of line this is. It will represent either a "C" choice page, or an "E" end page. Page 1 is always a choice page.
    Text - A string of text surrounded by double quotes. Including the quotes, this component will not exceed 256 characters. The quotes are given for input purposes only and should not be considered part of the text. The text will not contain embedded double quotes.
    Choices - Two positive integers from 1 to *X* indicating the pages where the reader can go from this page. Only choice pages have this component.
    Ending Type - Either the text "HAPPY" or "GRISLY". There will only be one happy ending per story, and only end pages have this component.

Output:

For each story in the input:

1. Output a single line, "STORY #" where # is 1 for the first story, 2 for the second story, etc.
2. Determine the story that begins on page 1 and ends on the happy ending page. Output the text of this story, printing one "page" of text per line. Note that there is only one such story for each data set.

Sample Input:

```
2
3
C "Arrived at LSU for the contest" 2 3
E "Was devoured by sidewalk ants" GRISLY
E "Won the contest. Received glory and nachos." HAPPY
5
C "Saw a peanut" 3 5
E "Made peanut butter sandwich" HAPPY
C "Found a hammer" 4 2
E "Hit self on head with hammer, ouch!" GRISLY
E "Ate the peanut, choked on it, and died" GRISLY
```

Sample Output:

```
STORY 1
Arrived at LSU for the contest
Won the contest. Received glory and nachos.
STORY 2
Saw a peanut
Found a hammer
Made peanut butter sandwich
```

acm International Collegiate Programming Contest 2004

South Central USA Regional Programming Contest

Know When to Hold 'em

Introduction:

Poker is played with a standard deck of cards (no jokers). Every card has rank and suit. Possible card ranks in ascending order are: A,2,3,4,5,6,7,8,9,T,J,Q,K,A (Note that 'A' can be either high or low). Possible card suits in ascending order (the order of suits is for display purposes only and should not be used in determining hand or card value) are: Clubs (c), Diamonds (d), Hearts (h), Spades (s). This yields a total of 13 x 4 = 52 cards in a deck.

Poker hands consist of five cards. The possible poker hands are listed below from best to worst. Examples for each type of hand are given in bold.

- *Royal Flush* -- Ace, King, Queen, Jack, Ten; all of the same suit. Ex: **Ad Kd Qd Jd Td**
- *Straight Flush* -- Five cards with sequential rank, all of the same suit. Ace can be high or low, but not both. When there are two or more straight flushes, the straight flush with the highest ranked card wins. Ex: **Ah 5h 4h 3h 2h**
- *Four of a Kind* -- Four cards of the same rank. If there are two or more hands that qualify, the hand with the higher ranking four of a kind wins. If there are two four of a kinds with the same rank, then the one with the high card outside the four of the kind wins. Ex: **Kc 8s 8h 8d 8c**
- *Full House* -- Three cards of the same rank plus two cards sharing a different rank. When comparing full houses, the rank of the three cards determines which is higher. If the threes of a kind were equal, the rank of the pairs would decide. For example **9s 9h 9d 4d 4c** beats **As Ah 8s 8h 8d**. Ex: **Ts Th 3s 3h 3d**
- *Flush* -- Five cards of the same suit, any rank. When comparing two flushes, the highest card determines which is higher. If the highest cards are equal then the second highest cards are compared; if those are equal too, then the third highest cards, and so on. Ex: **Kc 9c 8c 5c 2c**
- *Straight* -- Five cards with sequential rank, mixed suits. Ace can be high or low, but not both. When there are two straights, the straight with the highest ranked card wins. Ex: **9s 8h 7h 6c 5d**
- *Three of a Kind* -- Three cards with the same rank. The highest three of a kind rank wins. If both three of a kinds are the same rank, then the one with the highest cards outside the three of the kind wins. Ex: **Ks 8h 6h 6d 6c**

"Texas Hold 'em" is a variation of poker that has gained popularity recently due to frequent national TV exposure. Each player receives two cards face down, known as "hole cards". Five cards, known as "the board", are then placed face up in the middle of the table. A player wins by making the best five-card poker hand possible using any combination of their two "hole cards" and the five cards on "the board". A useful skill for a Hold 'em player is to be able to tell the best possible hand an opponent might have. All the player knows are the five face up cards and his/her two hole cards. If an opponent can't make a better hand than the player, then the player knows to bet as much as possible.

Input:

The first line contains a single integer *n* indicating the number of data sets.

Each data set is a line consisting of seven two-character strings separated from each other by single spaces. Each string represents a playing card. The first character will be the card's rank; the second character will be the card's suit. The first five cards represent "the board" while the last two represent the player's "hole cards". These cards are not in any specific order.

Output:

For each line of input, output the best five-card hand an opponent might make. If there are two or more cards of different suits that can be used to form the best hand (i.e., there is a tie), display those cards' suits as '*'. The cards should be listed in descending order based on rank first (Aces always considered high for display purposes) then suit (with '*' being lowest in the suit order). On the same line, output the name of the hand after the card list. Poker hand names: ROYAL FLUSH, STRAIGHT FLUSH, FOUR OF A KIND, FULL HOUSE, FLUSH, STRAIGHT, THREE OF A KIND.

Sample Input:

```
9
9c As 3h 8d 2s Jh Ah
6s 5h 5d 4s 4h 5s 5c
Qc 3d Qd Qh 3h 3c Qs
Ah 2c 8d 6h Js 2d As
8c 3d 2c Ad Tc Kc 4d
As Js Kc Tc Qs 9s 8s
2d Kd Ad 6h 3d As Ac
Qc Qh As Qd Qs Ah Ac
2d 5h 8s Jh Kd Ks As
```

Sample Output:

```
As 5* 4* 3h 2s STRAIGHT
6s 4s 4h 4d 4c FOUR OF A KIND
A* A* Qh Qd Qc FULL HOUSE
Ah Ad Ac Js 8d THREE OF A KIND
Ac Qc Tc 8c 2c FLUSH
As Ks Qs Js Ts ROYAL FLUSH
Ad 5d 4d 3d 2d STRAIGHT FLUSH
A* Qs Qh Qd Qc FOUR OF A KIND
Kh Kd Kc Jh 8s THREE OF A KIND
```

acm International Collegiate Programming Contest 2004

South Central USA Regional Programming Contest

Cubic Rube

Introduction:

Working in a Rubik's Cube factory has always been your dream job, but you're a clumsy rube. Now you're trying to invent a new variation, a 5x5x5 Rubik's Cube! You just walked into the research lab with a working prototype and accidentally dropped it, breaking it into 2 pieces. You have one piece in your left hand and are trying to find the other. Unfortunately, the floor is littered with parts from previous attempts, making it difficult to find the missing piece. Luckily, you know how to program...

Given two solids composed of unit cubes, determine if they can be fit together to form a solid 5x5x5 cube.

Input:

The first line contains a single integer *n* indicating the number of data sets. The following lines contain the data sets.

Each data set will show side-by-side top-down views of each of the solids as a 5x5 array of numbers 0-9. Each number represents the height of the solid at that point (the number of unit cubes stacked at that point). If the height is 0, there is no solid present at that point. This is similar to providing a topographical map of each solid, and it is valid to assume that there are no gaps "under" any part of a solid that cannot be seen.

Note that each solid will be connected (i.e., a single piece) and may require translations (not just rotations) to join with a compatible solid.

Output:

For each data set, print, "Yes" if a solid 5x5x5 cube can be formed by fitting together the two halves, and print, "No" if a cube cannot be formed. Each word should appear on its own line.

Sample Input:

```
2
55551 11111
55551 11111
55551 11111
55551 11110
55552 00000
22222 33333
22222 33333
22222 33233
22222 33333
22222 33333
```

Sample Output:

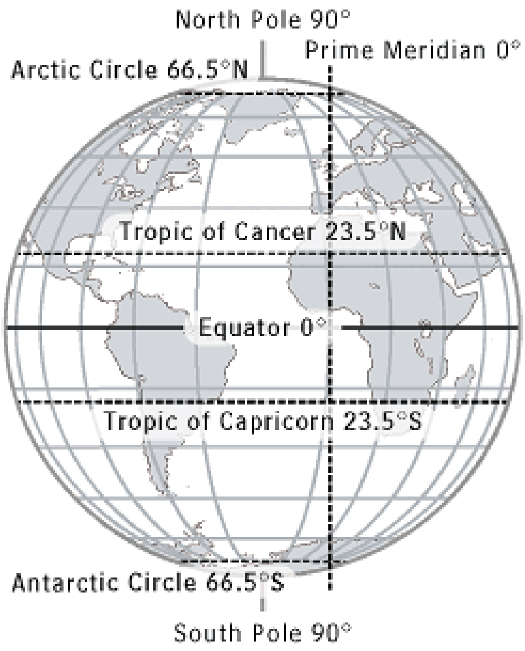
```
Yes
No
```

acm International Collegiate Programming Contest 2004

South Central USA Regional Programming Contest

As the Crow Flies

Introduction:



As president of a startup airline company, you have started a frequent flier program that rewards customers for every mile they travel. As a for-profit company, you have a vested interest in minimizing the number of frequent flier miles that a person can earn on any one trip. To get an idea of how many miles a customer could earn flying the existing network, you've decided to write a program.

Assumptions:

- A passenger's itinerary is one-way (no return flight).
- Every itinerary takes the shortest route from the departing city to the destination city.
- Frequent flier miles are counted "as the crow flies" (i.e., the shortest route across the earth's surface that connects the cities along the route).
- The earth's surface is a perfect sphere with radius 4000 miles.

Input:

The first line contains a single integer *n* indicating the number of data sets. Each data set will be formatted according to the following description:

A single data set has 3 components:

1. *Header Line* - A single line, "`X Y`", where *X* is the number of cities and *Y* is the number of flight legs

in the airline's network. Both will be positive integers less than 100.
2. *City List* - A list of cities and their locations, one city per line. The line will be of the format

   "`C LA NS LO EW`" where:

   > `C` is the name of the city (no spaces, alphabetical, first letter only upper case)
   > `LA` is the degrees of latitude where the city is located (from 0 to 90)
   > `NS` is the direction of latitude ('N'orth or 'S'outh of the equator)
   > `LO` is the degrees of longitude where the city is located (from 0 to 180)
   > `EW` is the direction of longitude ('E'ast or 'W'est of the prime meridian)
3. *Flight List* - A list of city pairs of the format "`B C`" representing different cities that are directly connected by flight legs, one pair per line. Note that "`B C`" is equivalent to "`C B`".

Note:

- Some longitude measurements can be represented in multiple ways (i.e., 180E = 180W)
- All degrees of latitude and longitude given in the input will be integers.
- The airline's network is connected (i.e., there is at least one route between any two cities).

Output:

For each data set, output the two cities that are farthest from each other (farthest in the sense that the shortest route between them is the longest of any city pair). You are guaranteed that there will be no ties. Display the city names on the same line, separated by a single space, sorted in dictionary order.

Sample Input:

```
2
6 5
Northpole 90 N 87 E
Southpole 90 S 180 W
Equatorone 0 N 45 W
Equatortwo 0 S 90 E
Equatorthree 0 S 180 E
Equatorfour 0 N 46 W
Equatorone Equatortwo
Equatortwo Equatorthree
Equatorthree Equatorfour
Northpole Equatortwo
Southpole Equatorthree
2 1
Northpole 90 N 0 E
Southpole 90 S 0 W
Southpole Northpole
```

Sample Output:

```
Equatorfour Equatorone
Northpole Southpole
```

acm International Collegiate Programming Contest 2004

South Central USA Regional Programming Contest

No Brainer

Introduction:

Zombies love to eat brains. Yum.

Input:

The first line contains a single integer *n* indicating the number of data sets.

The following *n* lines each represent a data set. Each data set will be formatted according to the following description:

A single data set consists of a line "`X Y`", where *X* is the number of brains the zombie eats and *Y* is the number of brains the zombie requires to stay alive.

Output:

For each data set, there will be exactly one line of output. This line will be "MMM BRAINS" if the number of brains the zombie eats is greater than or equal to the number of brains the zombie requires to stay alive. Otherwise, the line will be "NO BRAINS".

Sample Input:

```
3
4 5
3 3
4 3
```

Sample Output:

```
NO BRAINS
MMM BRAINS
MMM BRAINS
```