贺华南理工大学 60 周年校庆

“TCL 通讯杯”华南理工大学程序设计竞赛

（2012 SCUT Collegiate Programming Contest）

试

题

册

2012 年 11 月

A. School Anniversary

Description

Today is anniversary of the founding of SCUT!

As we know，it is the sixtieth (called Jia-Zi in Chinese)birthday of SCUT.

On November 5, 60 years of school-running achievement exhibition was held in the square before Shaw Building of Humanities. Current and senior school leaders, including WANG Yingjun, PENG Xinyi, LI Lin, ZHANG Zhen-gang, PENG Shuolong, QIU Xueqing, ZHU Min, ZHANG Xichun, LIU Zhengyi, LI Botian, and HUANG Shisheng, attended the opening ceremony. Directors from all offices and divisions, teacher and student representatives participated in the ceremony. ZHU Min presided over the ceremony.

The exhibition has an area of 2,000 square meters which includes several parts respectively referring to discipline development, talent cultivation, scientific research, etc. Through the utilization of pictures, words and multimedia, it well illustrates SCUT’s extraordinary development and significant achievements in facilitating economic and social growth of China, especially of Guangdong Province.

Edited by CHANG Wei and PENG Ying

From SCUT News Network

ChiChi，a clever student in School of Computer Science（SCS），has noticed that today is Nov. 17, 2012，Sat. He want know whether or not it is a Sat. for every birthday of SCUT.Please help him.

Input

Multiple testcase.
For each testcase, there will be one integer in the line, indicating the year.
HINT: The EOF indicates the end of input.

Output

For each testcase, just print a num indicates the day of the week in one line.

Sample Input

2012

2013

Sample Output

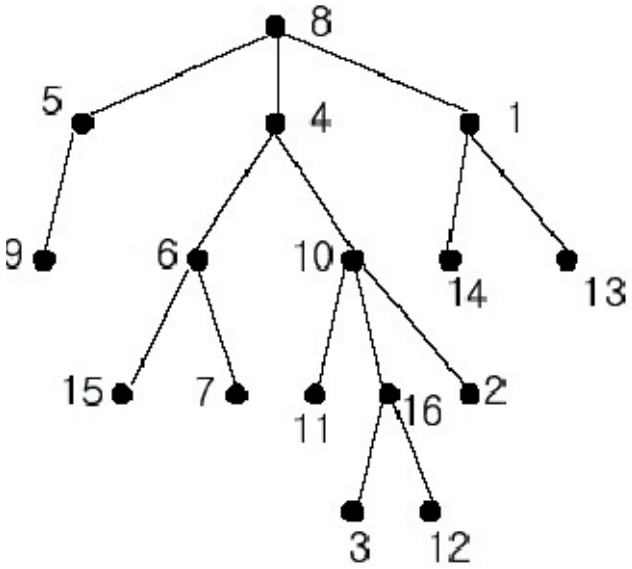
6

7

B. Nearest Common Ancestors

Description

A rooted tree is a well-known data structure in computer science and engineering. An example is shown below:



In the figure, each node is labeled with an integer from {1, 2,...,16}. Node 8 is the root of the tree. Node x is an ancestor of node y if node x is in the path between the root and node y. For example, node 4 is an ancestor of node 16. Node 10 is also an ancestor of node 16. As a matter of fact, nodes 8, 4, 10, and 16 are the ancestors of node 16. Remember that a node is an ancestor of itself. Nodes 8, 4, 6, and 7 are the ancestors of node 7. A node x is called a common ancestor of two different nodes y and z if node x is an ancestor of node y and an ancestor of node z. Thus, nodes 8 and 4 are the common ancestors of nodes 16 and 7. A node x is called the nearest common ancestor of nodes y and z if x is a common ancestor of y and z and nearest to y and z among their common ancestors. Hence, the nearest common ancestor of nodes 16 and 7 is node 4. Node 4 is nearer to nodes 16 and 7 than node 8 is.

For other examples, the nearest common ancestor of nodes 2 and 3 is node 10, the nearest common ancestor of nodes 6 and 13 is node 8, and the nearest common ancestor of nodes 4 and 12 is node 4. In the last example, if y is an ancestor of z, then the nearest common ancestor of y and z is y.

Write a program that finds the nearest common ancestor of two distinct nodes in a tree.

Input

The input consists of T test cases. The number of test cases (T) is given in the first line of the input file. Each test case starts with a line containing an integer N , the number of nodes in a tree, 2<=N<=10,000. The nodes are labeled with integers 1, 2,..., N. Each of the next N -1 lines contains a pair of integers that represent an edge --the first integer is the parent node of the second integer. Note that a tree with N nodes has exactly N - 1 edges. The last line of each test case contains two distinct integers whose nearest common ancestor is to be computed.

Output

Print exactly one line for each test case. The line should contain the integer that is the nearest common ancestor.

Sample Input

2

16

1 14

8 5

10 16

5 9

4 6

8 4

4 10

1 13

4

6 15

10 11

6 7

10 2

16 3

8 1

16 12

16 7

5

2 3

3 4

3 1

1 5

3 5

Sample Output

4

3

C. ACM Ranking

Description

In the SCUT/ICPC，the competition is more and more fierce. Now we will show some records the competitors handed in, which include the time of the submission, the teams' ID, the problem's ID, and the result of their programs running. Who will be the final top three?

The ranking method of ACM is:

a) The more problem the team solves, the higher rank it will be.

b) If some teams solve the same quantity of problems, they will be ranked according to the total time they used and their punishment time. The team who used less time will get a higher rank. If there is still a tie, the team with smaller ID is ranked higher.

c) Total time and punishment time are consisted of the time whenever your team makes correct solution and your team's punishment time.

d) The time of each problem is from the competition beginning to the problem you solved is judged as correct. During this period, every erroneous running will add twenty minutes to your team as punishment time if you solve this problem finally.

Input

The first line of the input is an integer T which indicates the number of test cases.

For each test case, the first line are three integers N ($3 \le N \le 100$) , M ($0 \le M \le 1000$) and Q ($1 \le Q \le 10$), indicating the number of teams, the number of records, and the number of problems. Teams are numbered from 1 to N, and problems are numbered from 1 to Q.

Each of the next M line describes a record in the following format

XX:XX A B C where XX:XX(00:00 – 05:00) is the time of the submission, A ($1 \le A \le N$) is the team ID, B ($1 \le B \le Q$) is the problem ID, and C is 1 if the program passes all test cases or 0

otherwise. All these items will be separated by a single space. The submission time is given in nondecreasing order, and no team will submit twice in one minute.

Output

For each test case, print a line contains the ID of the top three, separated by a single space. The team ID should be printed according to the ranking.

Sample Input

2

4 6 2

00:10 2 1 0

00:10 3 1 0

00:15 2 1 1

00:31 3 1 1

00:50 1 2 1

01:00 2 2 1

3 1 1

00:01 2 1 0

Sample Output

2 1 3

1 2 3

D. Billing Tables

Description

In the world of telecommunications phone calls to different phone numbers have to be charged using different rate or different billing plan. International Carrier of Phone Communications has an antique billing table that determines which phone call has to be charged using which billing plan.

Each international phone number has 11 digits. The billing table has n lines. Each line specifies a range of prefixes of phone numbers like “7919 - 921”. This specification means that all phone numbers starting from 7919, 7920, and 7921 match this line. A billing plan name is specified for each prefix. To determine a billing plan for a call, the table is scanned from top to bottom and the first matching line determines the billing plan. If no match is found, the phone number is invalid and no billing plan is needed. A special billing plan named “invalid” (without quotes) is used as an alternative way to define invalid phone numbers. Some billing plans are used for quite differently looking phone numbers and their names may be specified on different lines in different places of the table.

ICPC’s billing table is old and contains many entries. Some of those entries may not be even used anymore. It is very hard to figure out which phone numbers each billing plan is actually used for. The ICPC’s management has reached a decision to transform this billing table into a more legible format. In this new format table consists of the lexicographically ordered list of simple prefixes (without the “-” range feature of the old format) with a billing plan name for each prefix. No prefix of this new billing table should be a prefix of any other prefix from the table. Thus, a simple dictionary lookup (binary search, for example) will be sufficient to figure out a billing plan for a given phone number. Finding all phone numbers for a given billing plan will also become quite a simple task. The number of lines in the new billing table should be minimized. Billing plan named “invalid” should not be present in the new billing table at all, since invalid phone numbers will be denoted by absence of the corresponding prefix in the new billing table.

Input

There are multiple test cases.

The first line of each case contains a single integer number n ($1 \le n \le 100$) — the number of lines in the old billing table. The following n lines describe the old billing table with one rule on a line.

Each rule contains four tokens separated by spaces — prefix A, minus sign (“-”), prefix B, and billing plan name. Prefixes contain from 1 to 11 digits each, and the billing plan name contains from 1 to 20 lower case letters.

Further, let us denote with |A| the number of digits in the prefix A. It is true that $1 \le |B| \le |A| \le 11$. Moreover, last |B| digits of prefix A form a string that is lexicographically equal or precedes B. Such pair of prefixes A and B matches all phone numbers with the first $|A| - |B|$ digits matching the first digits of A and with the following |B| digits being lexicographically between the last |B| digits of A and B (inclusive).

Output

For each case output a single integer number k — the minimal number of lines that the new table should contain to describe the given old billing table. Then write k lines with the lexicographically ordered new billing table. Write two tokens separated by a space on each line — the prefix and the billing plan name. Note, that the prefix in the new billing table shall contain at least one digit.

If all phone numbers are invalid (every phone number has no matching line or matches line with billing plan “invalid”) then the output should contain just number zero.

Sample Input

8

7919 - 921 cell

7921800 - 999 priv

1 - 1 usa

760 - 9 rsv

7928 - 29 rsv

7600 - 7899 spec

73 - 77 invalid

7 - 7 cis

Sample Output

35

1 usa

70 cis

71 cis

72 cis

76 rsv

77 spec

78 spec

790 cis

7910 cis

7911 cis

7912 cis

7913 cis

7914 cis

7915 cis

7916 cis

7917 cis

7918 cis

7919 cell

7920 cell

7921 cell

7922 cis

7923 cis

7924 cis

7925 cis

7926 cis

7927 cis

7928 rsv

7929 rsv

793 cis

794 cis

795 cis

796 cis

797 cis

798 cis

799 cis

E. A Very Simple Hash Function

Description

The following string hash function is widely used in computer programming:

$$\text{Hash}(s) = \sum_{i=1}^{\text{length}(s)} \text{ascii}(s_i) \times a^{i-1}$$

Where ascii(s[i]) is the ASCII value of the ith character of the input string (from left to right) and a is an integer parameter.

Usually we use decimal representation as a string representation of an integer. However we use a different string representation for non-negative integers here. The string representation of x S(x) is defined recursively as follows:

$$S(x) = \begin{cases} "()" & x = 0 \\ "("+ S(0) + S(1) + ... + S(x-1) + ")" & x > 0 \end{cases}$$

"+" stands for string concatenation. For example S(3) is "(()(()))(()(()))" (without the quotes).

Your task is to compute the hash value of the string representation of a non-negative integer.

Input

There are multiple test cases. Each test case is a line containing two non-negative integers x and a(the parameter of hash function).

Output

For each test case you should compute the hash value for S(x), with the given a. Since the result may be huge, you only need to output the answer MOD 100000000 in one line

Sample input:

1 1

2 2

Sample Output:

162

10428

F. Bulb

Description

It is lctear's birthday today. Liulike has bought her a big birthday cake. The cake is special for it has many colored little bulbs equally arranged in a circle on the edge. There are two colors: blue and green – lctear's favorite colors :). In one unit of time, some of the lovely bulbs will change their colors, simultaneously.

Careful lctear found that a bulb changes its color only when the bulb anticlockwise next to it is green. E.g., if there are 6 bulbs and the initial colors are "GBGGBB" read anticlockwise ('G' denotes green, 'B' denotes blue), they will change to "GGBGBG" after one unit of time.

After several units of time, the colors of the bulbs are very different from the initial ones. Knowing Liulike is a talented programmer, lctear asked him if we can find out the initial colors of the bulbs, given the current state and the number of units of time from beginning.

Liulike thought for a while, and told his girl with a smile: we cannot know the initial state of colors, since there are often many possibilities, yet we can work out the number of different possibilities which can change into current state after *K* units of time.

Now the problem is given to you :-). Note two circles of colors are considered the same state if one can be gotten from the other by rotating it some number of positions.

Input

The first line is the number of test cases.

For each test case, the first line contains two integers: *N* and *K(1<=N<=200, 1<=K<=10000)* – the number of bulbs and the number of units of time from the beginning, followed by one line containing a string – the current state of bulbs.

Output

For each test case, output one line containing one integer: number of different possibilities of initial state. Note it can be a very big integer.

Sample Input

2

6 1

GGBGBG

6 3

GGBGGB

Sample Output

2

3

G. A Mincost-Maxflow Problem

Description

Given a flow network, that is, a directed graph $G = (V, E)$ with source $s \in V$ and sink $t \in V$, where edge $(u, v) \in E$ has capacity $c(u, v) \geq 0$ and cost $h(u, v)$. You are required to send a maximal amount of flow from s to t while the cost must be minimal.The formal definition of the problem is to minimize the total cost of the flow:

$$\sum_{(u,v)\epsilon E} f(u, v) \times h(u, v)$$

with the constraints:

Capacity constraints: $0 \leq f(u, v) \leq c(u, v)$ for all edge (u,v).

Flow conservation:
$\sum_{(u,v)\epsilon E} f(u, v) = \sum_{(v,u)\epsilon E} f(v, u)$ for every node u ($u \neq s$ and $u \neq t$).

Maximum flow condition: $\sum_{(s,u)\epsilon E} f(s, u) - \sum_{(u,s)\epsilon E} f(u, s)$ should be maximal.

Input

The first line of the input file contains a single integer K ($1 \leq K \leq 10$), the number of test cases, followed by the input data for each test case. The first line of each test case consists of two integers N and M ($2 \leq N \leq 100$, $0 \leq M \leq 5000$) giving the number of the vertexes and number of the edges respectively. The next line contains two integers s,t($0 \leq s,t \leq N-1$ and $s \neq t$) stand for the source vertex and sink vertex.Then M lines follow. Each line contains four integers p,q,c,h (0 <= p, q < N, $p \neq q$,0 <= c <= 200, -200 <= h <= 200), indicating there is an edge from vertex p to vertex q with capability c and the cost is h for per unit flow. It is possible that the graph contains more than one edge between two vertexes.

Output

For each test case print a line with the minimal cost.

Sample Input

1

2 1

0 1

0 1 1 1

Sample Output

1

H.Super Restaurant

Description

Moe Maid Cafe is one of the best restaurant in Neko city and Xuxu plans to have lunch and supper in Moe Maid Cafe every day in the next week. For each meal he selects one and only one combo in menu. Every combo has a positive integer score. The score of a meal plan is the product of scores of 14 selected combos of the plan. Of course, these combos are not free. Xuxu doesn't want to spent more than K cents and he wants to know the sum of scores of all possible meal plans.

Input

There serval test case. The first line of a test case is an integer N (1 <= N <= 20000) indicating the number of different combos in menu. The following N lines each contains two integers S (1 <= S) and P (1 <= P <= 10000) indicating the score and price(in cents) of a combo. Then an integer M (1 <= M <= 100000) and M integers Ki (0 <= Ki) in the next line indicate M cost limitations.

Output

For each Ki in a test case you should output the sum of scores of meal plans which costs no more than Ki cents in one line, separated by one space. Since the answer may be huge you only need to output the answer MOD 104857601.

Sample input:

1

1 10

3 100 1000 10000

Sample output

0 1 1

18

I. Exchange

Description

You are taking part in a large project to automate operations.Different resources and commodities are traded on this exchange via public auction. Each resource or commodity is traded independently of the others and your task is to write a core engine for this exchange — its order book. There is a separate instance of an order book for each traded resource or commodity and it is not your problem to get the correct orders into order books. The order book instance you will be writing is going to receive the appropriate orders from the rest of exchange system.

Order book receives a stream of messages. Messages are orders and requests to cancel previously issued orders. Orders that were not cancelled are called active. There are orders to buy and orders to sell. Each order to buy or to sell has a positive size and a positive price. Order book maintains a list of active orders and generates quotes and trades. Active order to buy at the highest price is the best buy order and its price is called bid price. Active order to sell at the lowest price is the best sell order and its price is called ask price. Ask price is always lower than bid price, that is, buyers are willing to pay less than sellers want to receive in return.

A current quote from the order book contains current bid size, bid price, ask size, and ask price. Here bid and ask sizes are sums of the sizes of all active orders with the current bid price and the current ask price correspondingly.

A trade records information about transaction between buyer and seller.

Each trade has size and price. If an order to buy arrives to the order book at a price greater or equal to the current ask price, then the corresponding orders are matched and trade happens — buyer and seller reached agreement on a price. Vice versa, if an order to sell arrives to the order book at a price less or equal to the current bid price, then trade happens, too. For the purpose of order matching, order book works like a FIFO queue for orders with the same price (read further for details).

When an order to buy arrives to the order book at a price greater or equal to the current ask price it is not immediately entered into the order book. First, a number of trades is generated, possibly reducing the size of incoming order. Trade is generated between incoming buy order and the best order to sell. If there are multiple best orders (at the ask price), then the order that entered the order

book first is chosen. Trade is generated at the current ask price with the size of the trade being equal to the smaller of the sizes of two matching orders. Sizes of both matching orders are reduced by the size of the trade. If that reduces the size of sell order to zero, then it becomes inactive and is removed from the order book. If the size of incoming buy order becomes zero, then the process is over — incoming order becomes inactive. If the size of incoming buy order is still positive and there is another sell order to match with, then the process continues generating further trades at the new ask price (ask price can increase as sell orders are traded against and become inactive). If there is no sell order to match with (current ask price became greater than incoming buy order price), then incoming buy order is added to the order book with its remaining size.

For incoming sell order everything works similarly – it is matched with buy orders from the order book and trades are generated on bid price.

On incoming cancel request the corresponding order is simply removed from the order book and becomes inactive. Note, that by the time of the cancel request the quantity of the corresponding order might have been already partially reduced or the order might have become inactive. Requests to cancel inactive order do not change anything in the order book.

On every incoming message the order book has to generate all trades it causes and the current quote (bid size, bid price, ask size, ask price) after processing of the corresponding message, even when nothing has changed in the order book as a result of this message. Thus, the number of quotes the order book generates is always equal to the number of incoming messages.

Input

The first line of the input file contains a single integer number n ($1 \le n \le 10\,000$) – the number of incoming messages that the order book has to process. The following n lines contain messages. Each line starts with a word describing the message type – BUY,SELL, or CANCEL followed after a space by the message parameters.

BUY and SELL denote an order to buy or to sell correspondingly, and are followed by two integers q and p ($1 \le q \le 99\,999$, $1 \le p \le 99\,999$) – order size and price. CANCEL denotes a request to cancel previously issued order. It is followed by a single integer iwhich is the number of the message with some preceding order to buy or to sell (messages are numbered from 1 to n).

Output

Write to the output file a stream of quotes and trades that the incoming messages generate. For every trade write TRADE followed after space by the trade size and price. For every quote write QUOTE followed after space by the quote bid size, bid price, minus sign (“-”), ask size, ask price (all separated by spaces).

There is a special case when there are no active orders to buy or to sell in the order book (bid and/or ask are not defined). This case is treated as follows. If there is no active order to buy, then it is assumed that bid size is zero and bid price is zero. If there is no active order to sell, then it is assumed that ask size is zero and ask price is 99 999. Note, that zero is not a legal price, but 99 999 is a legal price. Recipient of quote messages distinguishes actual 99 999 ask price from the special case of absent orders to sell by looking at its ask size.

See example for further clarification.

Sample Input

11

BUY 100 35

CANCEL 1

BUY 100 34

SELL 150 36

SELL 300 37

SELL 100 36

BUY 100 38

CANCEL 4

CANCEL 7

BUY 200 32

SELL 500 30

Sample Output

QUOTE 100 35 - 0 99999

QUOTE 0 0 - 0 99999

QUOTE 100 34 - 0 99999

QUOTE 100 34 - 150 36

QUOTE 100 34 - 150 36

QUOTE 100 34 - 250 36

TRADE 100 36

QUOTE 100 34 - 150 36

QUOTE 100 34 - 100 36

QUOTE 100 34 - 100 36

QUOTE 100 34 - 100 36

TRADE 100 34

TRADE 200 32

QUOTE 0 0 - 200 30

J.Sticks

Description

Tim took sticks of the same length and cut them randomly until all parts became at most 50 units long. Now he wants to return sticks to the original state, but he forgot how many sticks he had originally and how long they were originally. Please help him and design a program which computes the smallest possible original length of those sticks. All lengths expressed in units are integers greater than zero.

Input

Each case contains of 2 lines. The first line contains the number of sticks parts after cutting, there are at most 64 sticks. The second line contains the lengths of those parts separated by the space.

Output

The output should contains the smallest possible length of original sticks, one per line.

Sample Input

9

5 2 1 5 2 1 5 2 1

4

1 2 3 4

Sample Output

6

5

23

K.滑雪

Description

Michael 喜欢滑雪百这并不奇怪， 因为滑雪的确很刺激。可是为了获得速度，滑的区域必须向下倾斜，而且当你滑到坡底，你不得不再次走上坡或者等待升降机来载你。Michael 想知道载一个区域中最长底滑坡。区域由一个二维数组给出。数组的每个数字代表点的高度。下面是一个例子

1 2 3 4 5

16 17 18 19 6

15 24 25 20 7

14 23 22 21 8

13 12 11 10 9

一个人可以从某个点滑向上下左右相邻四个点之一，当且仅当高度减小。在上面的例子中，一条可滑行的滑坡为 24-17-16-1。当然 25-24-23-...-3-2-1 更长。事实上，这是最长的一条。

Input

输入的第一行表示区域的行数 R 和列数 C(1 <= R,C <= 100)。下面是 R 行，每行有 C 个整数，代表高度 h，0<=h<=10000。

Output

输出最长区域的长度。

Sample Input

5 5

1 2 3 4 5

16 17 18 19 6

15 24 25 20 7

14 23 22 21 8

13 12 11 10 9

Sample Output

25