Problem A
Prefix Sums

(Time Limit: 1 second)

Let n be a positive integer and $a_i \in \{1,2,3\}$ for all $i \in \{1,2,\ldots,n\}$. We want to perform n operations, each in one of the following two forms:

- Calculate $a_1 + a_2 + \cdots + a_k$ for a given $k \in \{1,2,\ldots,n\}$.
- Update a_i to x given $i \in \{1,2,\ldots,n\}$ and $x \in \{1,2,3\}$.

Input Format

The input begins with n, a_1, a_2, ..., a_n, where two consecutive numbers are separated either by space(s) or newline character(s). The n operations are specified in the order in which they are to be performed. In detail, we specify each operation in one of the following two ways:

- An operation of the form "sum k", where $k \in \{1,2,\ldots,n\}$, asks to calculate $a_1 + a_2 + \cdots + a_k$.
- An operation of the form "update i x", where $i \in \{1,2,\ldots,n\}$ and $x \in \{1,2,3\}$, asks to update a_i to x. Note that x may equal a_i, in which case the update does nothing.

Output Format

For each operation of the form "sum k", where $k \in \{1,2,\ldots,n\}$, output $a_1 + a_2 + \cdots + a_k$ in one line. Any operation of the other form requires no output but may affect subsequent outputs.

Technical Specification

- $n \leq 500000$.

Example

Sample Input:	Sample Output:
7 1 3 2 2 1 3 1 sum 7 update 4 1 sum 5 update 4 1	13 8 10

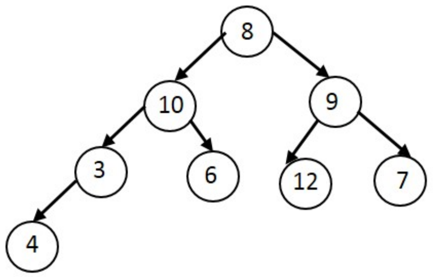
Problem A

update 2 2 update 3 2 sum 6	

Problem B
Path Sum Decision Problem

(Time Limit: 1 second)

Given a binary tree, we define a "root-to-leaf path" to be a sequence of nodes in the tree starting from the root and proceeding downward to a leaf node (a node without child). Suppose that a positive integer value, d_i, is assigned to the data field of each node n_i, for $1 \leq i \leq N$, in a tree. N is the number of nodes in the tree. Note that an empty tree contains no root-to-leaf paths. For example, the following tree has exactly four root-to-leaf paths:



The four root-to-leaf paths are:

path 1: 8-10-3-4
path 2: 8-10-6
path 3: 8-9-12
path 4: 8-9-7

For the path sum decision problem, we will be concerned with the sum of the values of such a path -- for example, the sum of the values on the 8-10-3-4 path is 8 + 10 + 3 + 4 = 25. Given a binary tree *BT* and an integer *S*. The path sum decision problem is to determine whether there exists a root-to-leaf path in the tree such that the sum of the values in the path is equal to *S*?

Write a program to solve the path sum decision problem. The input contains a binary tree *BT* and a positive integer *S*, and the output will be “True” or “False”. The binary tree is represented by a specific format described later. Given *BT* and *S*, the program outputs “True” if the tree has a root-to-leaf path such that the sum of the values along the path is equal to the given *S*. Output “False” if no such path can be found. For example, for the above binary tree, the answer will be “True” if *S* = 25; “False” when *S* = 21.

As for representation of a binary tree, it is given by format of preorder with appended “0”s. That

Problem B

is, beside the preorder representation, a “0” is appended after the node value if a node has no left child. The same is for the node does not have a right child. For example, the above binary tree is represented by the following sequence:

8 10 3 4 0 0 0 6 0 0 9 12 0 0 7 0 0

Note that each positive integer indicates the data value in the corresponding node, and a “0” indicates a termination (neither left child nor right child). Two adjacent numbers are separated by a blank space.

Input Format

The input file contains several test cases. The first line is an integer T indicating the number of test cases. The following lines denote the test cases. Each test case is composed of two lines. The first line is the sequence representing a binary tree, and the second line is a positive integer indicating the given value of S.

Output Format

For each test case, output one line of “True” if the tree has a root-to-leaf path such that the sum of the values along the path is equal to the given S. Otherwise, output “False” if no such path exists.

Technical Specifications

- N is the number of nodes in the tree, $1 \leq N \leq 20$.
- d_i is a positive integer, $1 \leq d_i \leq 99$, for $1 \leq i \leq N$.
- S is a positive integer, $1 \leq S \leq 10000$.
- There are at most 10 test cases, i.e., $T \leq 10$.

Example

Sample Input:	Sample Output:

4
6 3 0 0 2 8 0 0 0
16
6 3 0 0 2 8 0 0 0
8
10 11 0 7 0 0 12 6 0 0 5 0 0
	True
	False
	False
	True

Problem B

21
10 11 0 7 0 0 12 6 0 0 5 0 0
28	

Problem C
Roman Numerals

(Time Limit: 1 second)

Game of Thrones is an American fantasy drama television series. You may notice that there are special notations of numbers. For example, “IV” represents 4 and “XIII” represents 13. These notations are called **Roman numerals**. The numeric system represented by Roman numerals originated in ancient Rome and remained the usual way of writing numbers today. Numbers in this system are represented by combinations of letters of Latin alphabet. Roman numeral representations are based on seven symbols:

Symbol	I	V	X	L	C	D	M
Value	1	5	10	50	100	500	1000

The patterns of Roman numerals are as follows:

1. Repeating a numeral up to three times represents addition of the numbers. For example, III represents 1 + 1 + 1 = 3. Only I, X, C, and M can be repeated; V, L, and D cannot be, and there is no need to do so.
2. **Writing numerals that decrease from left to right represents addition of the numbers.** For example, LX represents 50 + 10 = 60 and XVI represents 10 + 5 + 1 = 16.
3. To write a number that otherwise would take repeating of a numeral four or more times, there is a subtraction rule. **Writing a smaller numeral to the left of a larger numeral represents subtraction.** For example, IV represents 5 - 1 = 4 and IX represents 10 - 1 = 9.

Please write a program to calculate the values of Roman numerals.

Input Format

The input contains several test cases, each of which appears on a line. Notice that there may be invalid input symbols which may break any of Roman numeral rules.

Output Format

Print out the values of input Roman numerals line-by-line. For an invalid test case, print a 0.

Problem C

Technical Specifications

- A string contains any undefined character (not one of the 7 symbols in the above table) is invalid.
- Repeating a character over 4 times is invalid.
- Value of input Roman numerals is guaranteed to be within the range from 1 to 3999.

Example

Sample Input:	Sample Output:
IV	4
IIII	0
XXIV	24
MLXAI	0
XCIX	99
CDLXXXVII	487
DCLXXXIX	689
MDCCLXXVI	1776
MCMLIV	1954
MCMXC	1990
MMXVIII	2018
MMMCMXCIX	3999

Problem D
Trinity

(Time Limit: 1 second)

Let $P = \{p_1, p_2, \ldots, p_n\}$ be a set of n points in a 2-dimensional space. We say that 3 distinct points p_i, p_j and p_k in P form a ***trinity***, denoted by $[p_i, p_j, p_k]$, if the distance between p_i and p_j is equal to the distance between p_i and p_k. The first point p_i is called the ***pivot*** of the ***trinity.***

For example, assume that there are 3 points {(2, 2), (3, 2), (2, 1)} in 2-dimensioinal space. Let p_1, p_2, and p_3 be (2, 2), (3, 2) and (2, 1), respectively. Then $[p_1, p_2, p_3]$ is a trinity, and p_1 is the pivot. Note that if we choose p_2 as the pivot, then $[p_2, p_1, p_3]$ is **not a trinity**, as the distance between (3, 2) and (2, 2) is not equal to the distance between (3, 2) and (2, 1). Also note that [(2, 2), (3, 2), (3, 3)] is not a trinity either, as the distance between (2, 2) and (3, 2) is not equal to the distance between (2, 2) and (3, 3).

Figure 1 shows 4 points, p_1, p_2, p_3 and p_4, in a 2-dimensioinal space. Assume that p_1 is the pivot, we can find three trinities. They are $[p_1, p_2, p_3]$, $[p_1, p_2, p_4]$, and $[p_1, p_3, p_4]$. The order of the second and the third points are not important. Therefore, $[p_i, \mathbf{x}, \mathbf{y}]$ and $[p_i, \mathbf{y}, \mathbf{x}]$ are considered as the same trinity.

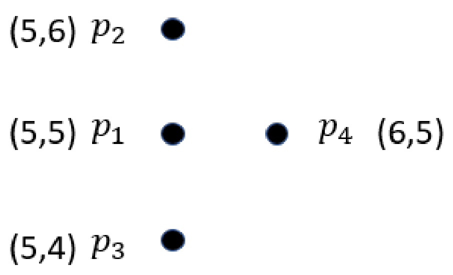


Figure 1. Four points in a 2-dimensional space.

Given a set of n points in 2-dimensional space, write a program to compute the number of trinities in the given points. For example, in Figure 1, there are 4 trinities: $[p_1, p_2, p_3]$, $[p_1, p_2, p_4]$, $[p_1, p_3, p_4]$, and $[p_4, p_2, p_3]$.

Input Format

The first line is a number indicating the number of test cases. There are at most 5 test cases.

The first line of each test case contains an integer T ($0 \le T \le 600$), representing the number of points in this test case. Each of the next T lines contains a pair of integers x, and y. Note that x and y

Problem D

are separated by a space. Both *x* and *y* are integers, and $0 \leq x, y \leq 10000$.

Output Format

For each test case, print the number of trinities on one line.

Technical Specification

Example

Sample Input:	Sample Output:

2
0
3
2 2
3 2
2 1
	0
	1

Problem E
City Surveillance System

(Time Limit: 1 second)

In Tree City, the roads are straight and connected in a tree structure, a structure without any cycles. One day, Tree City mayor decided to build a City Surveillance System for public safety. The system will have some cameras installed at the crossroads and dead ends to watch all the city roads. Crossroads and dead ends are regarded as the guarded points. Two guarded points are adjacent if they are respectively at the two ends of the same road. The camera installed at a certain guarded point can watch all the roads incident to it. The distance between any two adjacent guarded points is the same as the watching distance of a camera. The placement of the cameras has to satisfy the following two requirements: (1) each guarded point has at most one camera installed and (2) each camera must be watched at least by a camera placed at any adjacent guarded point to prevent malicious damage. For example, the placement of cameras (black guarded points) satisfies the requirements in Figure 1 but not in Figure 2.

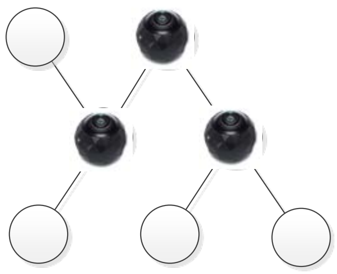


Figure 1. Satisfied placement

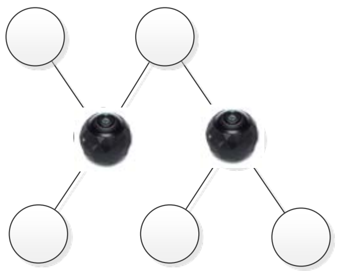


Figure 2. Unsatisfied placement

Please write a program to compute the minimum number of required cameras to watch all the city roads.

Input Format

The first line is a number indicating the number of test cases. For each test case, the first line is an integer n, $2 \le n \le 10000$, which indicates the number of guarded points. Each of the next n-1 lines consists of two integers separated by a space, which indicates the indices of two adjacent guarded points of a road.

Output Format

For each test case, output the minimum number of cameras, which satisfies the requirement of the problem.

Problem E

Technical Specifications

- There are at most 10 test cases.
- There are n guarded points, $2 \le n \le 10000$. Each guarded point has a unique index between 1 and n and its index is larger than the index of its parent except the root (index 1).

Example

Sample Input:

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

Sample Output:

```
2
3
```

Problem F
Special N-Queen Problem

(Time Limit: 3 seconds)

Given a number ***n***, it is interesting to know the number of ways to place ***n*** queens on an ***n*** by ***n*** chessboard such that each queen attacks no queens. Here a queen can attack others horizontally, vertically, left to right diagonally, and right to left diagonally. Moreover, if we pre-place an obstacle at a special position where any queen cannot be placed and pre-place one queen at a special position, then we are interested in the number of ways to place ***n* - 1** queens such that any queen attacks no queens?

For example, let ***n* = 8**, the pre-placed queen Ⓠ be at **(0,0)**, and the obstacle ⊗ be at **(1,6)**. There are only two ways to place ***n* - 1** queens such that any queen attacks no queens as follows.

	0	1	2	3	4	5	6	7
0	Ⓠ							
1					Q		⊗	
2								Q
3						Q		
4			Q					
5							Q	
6		Q						
7				Q				

	0	1	2	3	4	5	6	7
0	Ⓠ							
1						Q	⊗	
2								Q
3			Q					
4							Q	
5				Q				
6		Q						
7					Q			

Input Format

The first line is a number indicating the number of test cases. Each test case has one number ***n***, two integers indicating the position of a pre-placed queen and two integers indicating the position of an obstacle. The pre-placed queen's position and the obstacle's position are not the same.

Output Format

For each test case, please output the number of ways to place other queens such that any queen attacks no queens.

Problem F

Technical Specification

- There are at most **10** test cases.
- The value of ***n*** is at most **14**.

Example

Sample Input:	**Sample Output:**
`2`	`2`
`8 0 0 1 6`	`3`
`8 0 7 1 2`	

Problem G
Profit Maximization

(Time Limit: 1 second)

A part-time worker has ***n*** part-time job opportunities, each of which has its own clock-in time ***s***, clock-out time ***f***, and wage ***p***. The part-time worker can only do a part-time job at a time. In other words, the part-time worker cannot simultaneously do two different part-time jobs. For example, if the working hours of a part-time job ***A*** and a part-time job ***B*** are respectively from 4 (i.e., ***s*** = 4) to 8 (i.e., ***f*** = 8) and from 6 (i.e., ***s*** = 6) to 10 (i.e., ***f*** = 10), the part-time worker can only select either the part-time job ***A*** or the part-time job ***B***. Note that if the clock-out time of a job is the same as the clock-in time of another job (i.e., f_1 = 4 and s_2 = 4), these two jobs are conflict free. So, the part-time worker needs to select a set ***J*** of non-conflict part-time jobs from ***n*** part-time job opportunities to maximize his/her earning (or the total amount of wages $\sum_{j \in J} p_j$).

For example, Table 1 lists four part-time job opportunities n_1, n_2, n_3, and n_4. If a part-time worker wants to get the maximum amount of wages, he/she should do n_1 and n_4 because n_1 and n_4 form a non-conflict job set (i.e., $J = \{n_1, n_4\}$) that maximizes his/her earning. Although n_1, n_2, and n_3 are also conflict-free, the total amount of wages is only 170 if they are selected. Note that the part-time worker cannot do n_3 and n_4 at the same time because of the overlap of working hours.

Table 1. Part-time job opportunities

Job	Clock-in time (***s***)	Clock-out time (***f***)	Wage (***p***)
n_1	1	2	50
n_2	3	5	20
n_3	6	19	100
n_4	2	20	200

Given a set of part-time job opportunities, write a program to find a non-conflict job set that will maximize the total amount of wages.

Input Format

The first line is an integer *n* indicating the number of part-time job opportunities. Each of the next *n* lines has **three** parameters ***s***, ***f***, and ***p***, which indicate a job's clock-in time, clock-out time, and wage.

Problem G

Output Format

For each test case, please output the maximum amount of wages, which is obtained from a set of non-conflict part-time jobs.

Technical Specification

- There are at most **30** test cases.
- $\mathbf{1 \leq n \leq 200}$.
- $\boldsymbol{1 \leq s \leq 300}$.
- $\boldsymbol{s < f \leq 400}$
- $\boldsymbol{1 \leq p \leq 500}$.

Example

Sample Input:	Sample Output:
3 1, 2, 50 3, 5, 20 6, 19, 100 4 1, 2, 50 3, 5, 20 6, 19, 100 2, 100, 200	170 250

Problem H

Maximizing Cryptocurrency Mining Profit

(Time limit: 1 second)

A cryptocurrency, such as *Bitcoin* or *Ethereum*, is a digital asset designed to work as a medium of exchange that uses cryptography to secure its transactions, to control the creation of additional units, and to verify the transfer of assets.

There are no central authorities in a cryptocurrency system. Blockchains are commonly used for a publicly accessible ledger of transactions. For simplicity, a blockchain is a chain of blocks. Each block contains a set of transactions, as well as other information, such as digital signatures, hash values, the nonce, etc.

In a cryptocurrency system, mining is a validation of transactions. A miner computes the value of the nonce in a block so that the hash value of the block is within a predefined range. There are no known efficient algorithms to compute proper value of the nonce, and mining usually takes quite a long time to succeed. For example, it usually takes 10 minutes to generate a new block in Bitcoin. For this effort, successful miners obtain new cryptocurrency as a reward.

In this problem, assume that many transactions can be collected into a block, and the owner of the transaction can offer a bigger amount of profit so that the transaction will be selected and processed quicker.

Mathematically, let T be a set of n transactions, $T = \{t_1, t_2, \ldots, t_n\}$. The size of each transaction t_i is s_i, and its profit is p_i. A miner can collect any subset of T into a block, as long as the sum of the sizes of the selected transactions is no more than S, where S is the capacity of a block.

Write a program to select a subset of transactions in T into a block in a way that the profit of the successful mining of that block is maximized.

Input Format

The input file may contain many test cases. Each test case contains a positive integer n in a line. The second line contains n sizes $s_1, s_2, \ldots, s_n$. The third line contains n profits $p_1, p_2, \ldots, p_n$. The last line of the test case contains the capacity of a block S. They are all nonnegative integers. Furthermore,

Problem H

1. $n \leq 200$,
2. s_i, $t_i \leq 10^8$, and
3. $S < 10^9$.

The last line of the input file is 0. Your program must exit if this line is reached.

Output Format

For each test case print the maximum profit which can be achieved for a block.

Examples

Input	Output

4
6 3 4 2
30 14 16 9
10
	46

5
2 6 3 4 2
9 30 14 16 9
10
	48

0	

Problem I
Lucky Date

(Time Limit: 3 seconds)

There are 365 days in a year, except for the leap years. Some people think that a date consisting of at most two digits is a “Lucky Date”. For example, **0101** is **1/1**, which has two digits, **0** and **1**, so we call **1/1** a Lucky Date. Instead, **0328** is **3/28**, which has four digits, **0**, **2**, **3** and **8**, so we will not call **3/28** a Lucky Date. As an exception, **4/4** is a date for death and is not a Lucky Date.

Please find the number of Lucky Dates between and including two given dates, $\mathbf{d_1}$ and $\mathbf{d_2}$. If $\mathbf{d_1}$ precedes or equals $\mathbf{d_2}$, they are in the same year. Otherwise, $\mathbf{d_1}$ is in one year and $\mathbf{d_2}$ is in the next year. The interval between any two given dates is at most one year. No leap years are considered.

Input Format

The first line contains an integer $\boldsymbol{n}$, indicating the number of test cases. Each of the following $\boldsymbol{n}$ lines contains two dates $\mathbf{d_1}$ and $\mathbf{d_2}$.

Output Format

For each test case, output the number of Lucky Dates between and including the two given dates.

Technical Specification

- There are at most **20** test cases.
- $\mathbf{1 \leq \boldsymbol{n} \leq 20}$.

Example

Sample Input:	**Sample Output:**
3 1/1 1/2 4/3 4/5 12/31 1/5	1 0 1