The 15th Guangdong Collegiate Programming Contest

（GDCPC '2017）

Problems List

广东省计算机学会主办

华南农业大学

广州市中智软件开发有限公司

Problem A: Open Box

Description

Welcome to Game Delightful Collegiate Playing Contest (GDCPC)! In this contest, we are exciting to invite everyone to play games with us. We have prepared a box which contains bunch of games. All these games are designed for extraordinary players, so we lock the box to prevent unqualified people from accessing it.

Now, the first thing you need to do is unlock the box. It is a 4-digits passcode lock, each digit could be any number between 0 and 9. In each move, you can change each digit to its immediate neighbor. For example, if current digit is 2, you can change it to either 1 or 3. Specifically, if current digit is 0, you can change it to 1 or 9, if current digit is 9, you can change it to either 0 or 8.

The correct password is **9394**, but the initial statuses of each lock are different. Given the initial status, please find out how many steps it takes to unlock it.

Input

The first line is an integer T (T<= 10000), indicate the number of test cases.
Then n lines follow. Each line contains one integer indicate the initial 4-digits setup of a lock.

Output

For each test case, output "Case #t: s", t represents the index of test cases (start from 1), s represents the number of steps to unblock the box.

Sample Input	**Sample Output**
3 0000 9999 1170	Case #1: 9 Case #2: 9 Case #3: 10

Problem B: Power Grid

Description

Power Grid is a board game playing with many cities, players try their best to travel between cities, and supply as many cities as possible with electric power.

In this game, there are n cities and m pipelines between cities. All pipelines are one direction, there may be multiple pipelines connecting the same pair of cities, or none connecting them.
At the beginning of the game, the player needs to choose a starting city to start. In each turn of the game, the player needs to move away from current city to any adjacent city via a pipeline connecting these two cities.

Sounds easy, right? Of course not! We need to take power into consideration.

At the beginning of the game, if a player chooses the i-th city to begin with, that player will gain Ei units of electric as the initial setup.

When players move from City A to City B, they have to spend some units of electric power to use a pipeline to supply City B. There may be more than one pipelines from City A to City B or none.
After players move into the i-th city, they will re-charge Ei units of electric power from the power grid of that city.

A player can visit a city many times and each time he can re-charge from that city. Players can go through the same pipeline more than once, and each time he goes, he has to spend the corresponding electronic power.

A pipeline required Wp units of electric power is allowed to go through, only when a player has no less than Wp units of electric power.

Some special rules of this game:
1. A player can visit a city more than once. Every time he does, he is able to re-charge.
2. A player can use the same pipeline as many times as he want, as long as he has enough electric power to supply the usage.

Here, GDCPC committee is interest in figuring out if it’ s possible for a player to supply all cities.

Input

The first line is an integer T, indicate the number of test cases.

The first line of each test case contains two integers n and m (1 <= n<= 16, 1 <= m<= 150), which indicate the number of cities and the number of pipelines.

The second line of each test case contains n integers, the i-th integer Ei (0 <= Ei<= 2 ^ 32 - 1) indicates how many units of electric a player can re-charge from the i-th city.

Then m lines follow, each line consists three integers u, v, w (0 <= u, v < n, 0 <= w <= 2 ^ 32 - 1), indicate there is a pipeline starting from u-th city and ending at v-th city, and the cost of power to use this pipeline is w units of electric power.

Output

For each test case, output “Case #t: s”, t represents the index of test cases (start from 1), s represents if it’s possible to supply all cities. If yes, please output “Yes”, otherwise please output “No”.

Sample Input	Sample Output

3
3 2
5 3 3
0 1 5
1 2 3
2 1
10 0
0 1 20
4 5
10 5 0 0
0 1 20
0 1 5
1 2 5
2 1 5
1 3 5
	Case #1: Yes
	Case #2: No
	Case #3: Yes

Problem C: Stockpile

Description

Stockpile is an economic board game that combines the traditional stockholding strategy of buy low, sell high with several additional mechanisms to create a fast-paced, engaging and interactive experiences.

Stock option is one of the most important concepts of stock market. For each stock option, there are two kinds of stock options, **CALL** and **PUT**.
For CALL, the user needs to pay a fixed cost C dollar when they receive the option. At the right timing, they can choose to buy K units of stocks with price P, regardless the actual market price. If the actual market price X>P, the user can buy with price P and sell it with price X in the actual market, which gain (X - P) * K- C dollar. Otherwise, the user lost C dollar as the initial cost.
For PUT, the user (also) needs to pay a fixed cost C dollar when they receive the option. At the right timing, they can choose to sell K units of stocks with price P, regardless the actual market price. If the actual market price X<P, the user can buy from the actual market with price X, and sell with P, which gain (P-X) * K- C dollar. Otherwise, the user lost C dollar as the initial cost.

For example, a user receives two options:

1) CALL, C1 = $20, P1 = $30, K2 = 10
2) PUT, C2 = $10, P2 = $20, K2 = 5

In that case, if the market price is P = $5, the user gains $45:

1) From option #1, market price P = $5 < P1 = $30, so the user lost C1 dollar which is -$20
2) From option #2, market price P = $5 < P2 = $20, so the user gain (P2 - P) * K2 – C2 = $65
3) Overall, the user got -$20 + $65 = $45

Take another example, if the market price is P = $35, the user gains $20:

1) From option #1, market price P = $35 > P1 = $30, so the user gain (P – P1) * K1 – C1 = $30
2) From option #2, market price P = $35 > P2 = $20, so the user list C2 dollar which is -$10
3) Overall, the user got $30 - $10 = $20

The last example, if the market price is P = $25, the user lost $30 since neither of the options satisfy the price requirement, so the user lost all cost C1 + C2 = $30.

Now, being a player who already receive a bunch of stock options, please figure out how much you can gain (or lost) when the market price change. So, please calculate the formula between the market price and your final gain.
Obviously, the formula contains multiple equations in the format of $b = ax + c$ (x represents the market price, b represents the investment gain). Please output them in ascending order base on

x. Take above case as example, the formula should be:

$b = -5x + 70\ (0 \le x < 20)$
$b = -30\ (20 \le x < 30)$
$b = 10x - 330\ (30 \le x)$

Input

The first line is an integer T (T <= 2000), indicate the number of test cases.

Each test case begins with a single integer n (n <= 10^6), indicate the number of options you receive.

Then n lines follow. Each line consists of one string and three integers Type Ci Pi Ki

Type could be either PUT or CALL, indicate the type of the option.

Ci indicate the cost of this option. Pi indicate the price of this option. Ki indicate the number of unit user could buy of this option.

Output

For each test case, please output “Case #t:” in the first line, t represents the test case number (starting from 1).

Then m lines follow, each line should present the equation in the format as:[Pstart, Pend] b=a*x+c.

Pstart and Pend indicate the range of this equation.

a and c indicate the parameter of the formula, see example output for more details.

Specifically:

In the last output line of each test case, Pend is infinite. In that case, use “...” to represent that.

if a = 0, please output b = c

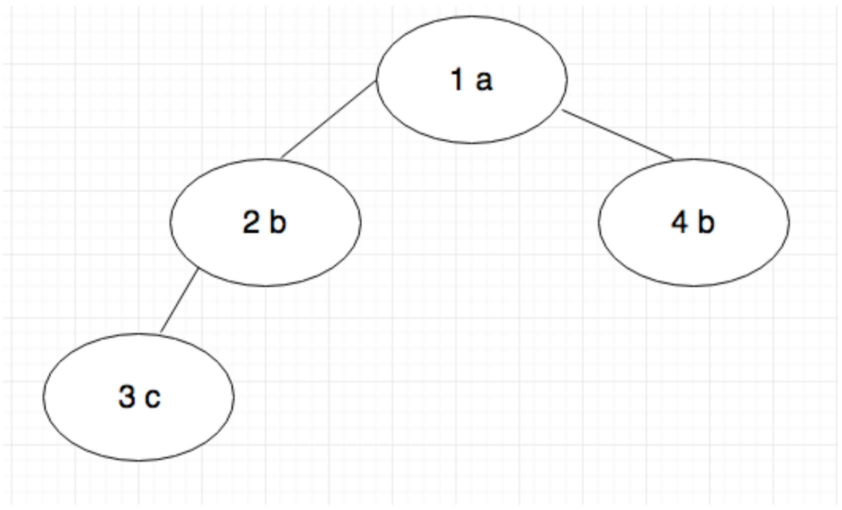
if a != 0 and c = 0, please output b = a*x

Sample Input	Sample Output
2 2 Call 20 30 10 Put 10 20 5 4 Call 32714 170443 16 Call 6846 101607 5 Call 1827 60167 31 Call 1909 124978 47	Case #1: [0,20] b=-5*x+70 [20,30] b=-30 [30,...] b=10*x-330 Case #2: [0,60167] b=-43296 [60167,101607] b=31*x-1908473 [101607,124978] b=36*x-2416508 [124978,170443] b=83*x-8290474 [170443,...] b=99*x-11017562

Problem D: Boggle Game

Description

Boggle game is a famous game to play with words. Usually people play this on a 2D matrix. But this is GDCPC, we are doing something special.



In this problem, we play the game on a tree with n nodes. Each node of this tree has an individual ID (starting from 1), and has one lowercase character on it.

We are defining a bunch of concept here to help playing this game:

1. **PathString**: From a node of this tree, keep going to parent nodes until it reaches the root of the tree, it forms a string. For example, starting 3rd node, Path String is “cba”.
2. **PathStringSet**: From a node of this tree, Path String Set includes all prefix of the Path String of this node. For example, starting from 3rd node, you can get Path String “cba”, so Path String Set of this node is { “c”, “cb”, “cba” }
3. **DistinctPathStrings**: The total different strings of all Path String Set in this tree. For example, there are 6 different Path Strings in this tree: “a”, “b”, “c”, “ba”, “cb”, “cba”
4. **WeightDistinctValues**: Define the value of a string is the sum of the ASCII value of all characters in the string. For example, value(“ba”) = 98 + 97 = 195. Now let’s sort all DistinctPathStrings by lexicographic order. In the sample above, SortedDistinctPathStrings = { “a”, “b”, “ba”, “c”, “cb”, “cba” }.

   $$\text{WeightDistinctValues} = \sum (rank(i) * value(\text{SortedDistinctPathStrings } [i]))$$

   rank(i) indicate the index of the string. For example, in this case:

   $$\text{WeightDistinctValues} = 1 * \text{value}(a) + 2 * value(b) + \cdots + 6 * value("cba")$$

5. **LCP**: Given two strings, LCP means the common prefix of these two strings. For example, LCP(“abcd” , “abef”) = “ab”
6. **Pairwise LCP Sum**: Given k integers x1, x2, ..., xk (xi >= 1).

$$PairWiseLCPSum(x1, x2.., xk) = \sum_{1 \le i < j \le k} LCP(PathString(xi), PathString(xj))$$

7. **HemiPalindrome**: Given a string, if all odd number places form a palindrome, or all even number places form a palindrome. For example, given string “abac” , all odd number place forms “aa” which is palindrome, all even number place forms “bc” which is not palindrome. So “abac” is Hemi Palindrome. On the other hand, “abcd” is not Hemi Palindrome.
8. **HemiPalindromeCount**: From a node x of this tree, HemiPalindromeCount(x) means the number of Hemi Palindrome strings in PathStringSet(x).
9. **WeightTotalHemiPalindrome**:

$$\text{WeightTotalHemiPalindrome} = \sum_{i=1}^{n} i * HemiPalindromeCount(i)$$

10. **HemiPalindromeSet**: Given x,HemiPalindromeSet(x) includes all distinct HemiPalindrome in the group of PathStringSet(1), PathStringSet(2), ..., PathStringSet(x).
    For example:

$$HemiPalindromeSet(2) = \{"a", "b", "ba"\}$$
$$HemiPalindromeSet(3) = \{\text{"a"}, \text{"b"}, \text{"c"}, \text{"ba"}, \text{"cb"}, \text{"cba"}\}$$

11. **DeltaHemiPalindrome**:

$$\text{DeltaHemiPalindrome(i)} = |\text{HemiPalindromeSet(i)}| - |\text{HemiPalindromeSet(i} - 1)|$$

For example:

a. DeltaHemiPalindrome(1) = 1 (i.e. “a”)
b. DeltaHemiPalindrome(2) = 2 (i.e. “b” , “ba”)
c. DeltaHemiPalindrome(3) = 3 (i.e. “c” , “cb” , “cba”)
d. DeltaHemiPalindrome(4) = 0 (“b” , “ba” already included before)

12. **WeightDistinctHemiPalindrome**:

$$\text{WeightDistinctHemiPalindrome(i)} = \sum_{\text{i}=1}^{\text{n}} \text{i} * \text{DeltaHemiPalindrome(i)}$$

13. **LenHemiPalindrome**: Given x and len, for all elements of PathStringSet(x) which have length <= len, find out all HemiPalindrome among them.

For example, LenHemiPalindrome(2, 1) = 1, LenHemiPalindrome(2, 2) = 2.

Your task here is quite complicate. Please refer to the input / output description.

Input

The first line is an integer T (T <= 100), indicate the number of test cases.
Then T cases follow.
The first line of each test case consists of three integers, n, type, m. n indicates the number of nodes, m indicates the number of queries.
Then n lines follow, each line contains two integers Fa[i] and Letter[i]. Fa[i] means the parent node of i-th node, when Fa[i] == 0, it means i-th node is the root of tree. Letter[i] means the letter on that node.
Then m lines follow, each line consist the information of one query.
For each query, it starts with an integer Qtype.

If Qtype == 0, it follows with an integer k, then k integers x1, x2, ..., xk follow. Please calculate $\mathrm{PairWiseLCPSum}((x1 + \mathrm{lastans} - 1)\ \%\ n + 1, \ldots, (xk + \mathrm{lastans} - 1)\%\ n + 1)$.

If Qtype == 1, then two integers x, len follow. Please calculate LenHemiPalindrome
((x + lastans – 1) % n + 1, (len + lastans – 1) % n + 1).

Here, lastans means the result of previous query. At the beginning of the queries, lastans = DistinctPathStrings.

0 <= type, Qtype<= 1
For each test case, 1 <= n <= 100000, 1 <= x, len<= n, 1 <= x1, x2, ..., xk<= n, ‘a’ <= Letter[i] <= ‘e’ .
For all test cases, $\sum n \le 1000000,\ \sum m \le 2000000,\ \sum k \le 2500000.$
When type = 1, the number of leaf nodes<= 100
When type = 0, Qtype = 0 for all queries.

Output

For each test case, output “Case #t:”, t represents the index of test cases (start from 1).

For each query:

If type == 0, output three integers:

1. DistinctPathStrings
2. WeightDistinctValues % 10000000007
3. The sum of all m queries

If type == 1, output five integers:

1. DistinctPathStrings
2. WeightDistinctValues % 10000000007
3. WeightTotalHemiPalindrome
4. WeightDistinctHemiPalindrome
5. The sum of all m queries

Sample Input	**Sample Output**
1 5 1 3 0 a 1 b 2 c 1 b 4 c 0 3 1 3 5 1 3 5 0 3 1 3 5	Case #1: 6 4023 37 14 7

Problem E: Dungeons & Dragons

Description

Dungeons & Dragons is a structured yet open-ended role-playing game. Players are adventurers, with each member often having their own area of specialty which contributes to the success of the whole.

During the game, players need to fight against different monsters. At the beginning of the game, players are at Level 0. A player takes **s** seconds for each attack, each attack give **w** units of damages.
Players’ level will increase according their will, but the maximum level they can reach is vMax. When player reach Level i, attacking time change to **s + i * x**, attacking damage change to **w + i * y**.

When players attack boss monsters, they want to minimize the number of seconds they spend to defeat those monsters. So, when they fight against a monster with b units of health points, they will first increase their level to any Level v(0 <= v<= vMax), and then keep attacking that monster until they give at least b units of damages.
Please be aware that it takes NO time for players to increase their level (Sounds ridiculous, right? Well, don’t be so serious).

Now, giving the value of s, w, x, y, vMax, b, please figure out what’s the minimal attacking time to defeat a monster.

Take sample #1 as example.
The monster has 10 units of health points, the player need to spend 2 seconds to attack, which gives 2 units of damage. Every time the player increases their level by 1, both attacking time and damage increase by 2. The maximum level of this player is 8.

In that case, the total attacking time is shown as below.

Level	Attacking time	Damage	NO. of attack	Total time
0	2	2	5	10
1	4	4	3	12
2	6	6	2	12
3	8	8	2	16
4	10	10	1	10
5	12	12	1	12
6	14	14	1	14
7	16	16	1	16
8	18	18	1	18

Obviously, the minimal attacking time is 10 seconds.
Under similar logic, in the second example, the best strategy is increase to Level 4, attacking time changes to 6, damage change to 10. This only takes 6 seconds to defeat the monster with only one attack.

Input

The first line is an integer T (T <= 2000), indicate the number of test cases.
Then n lines follow. Each line contains 6 positive integers, s, w, x, y, vMax, b, as what it described above. (All inputs are no more than 10^9)

Output

For each test case, output “Case #t: s”, t represents the index of test cases (start from 1), s represents the minimal attacking time.

Sample Input	Sample Output

2
2 2 2 2 8 10
2 2 1 2 8 10
	Case #1: 10
	Case #2: 6

Problem F: Werewolves

Description

Werewolves of Miller's Hollow is a game that takes place in a small village which is haunted by werewolves. It is a social game that requires no equipment to play, and can accommodate almost any large group of players.

Now, please help us to write an AI to play Werewolves!
Nope, just kidding. It will be too complicate to do that, let’s have a simpler question here.

Playing Werewolves require a lot of people, so the host will need to prepare a large place for the players. But cleaning up this large space is really difficult. Thus, a robot Pita is invented to do the dirty work. Pita can explore houses and do the cleaning. It explores the room with a complicate logic.

Each house consists of four components: wall, door, room and movable objects. A house is represented as a n * m matrix which only contains 0 and 1. 1 indicates **un-walkablearea** (e.g. wall, or moveable objects like beds, cabinets), 0 indicate **walkablearea** (e.g. room, part of a door).

If a grid is “1”, it identified as wall if:
It’s on the first / last row, or first / last column, or
It directly connected with a wall
Otherwise, this grid identified as movable object.

Rooms are separated by doors. Each door must include walkable area (the 0s) and door frame (the 1s). According to analyze, door dimensions could be either 2 * 4, 2 * 5, 4 * 2 or 5 * 2.
Thus, in the matrix, a door could be represented as either one of the follow:

```
11    11    100001   1000001
00    00    100001   1000001
00    00
00    00
00    00
00    11
11
```

Furthermore, there are more criteria to identify a door:

1. It’s impossible to install a door on a desk or other movable objects, so a door need to **directly connect with a wall**;

2. A door need to connect two walkable area, so **in both sides of a door, it has to be walkable area**;
3. It need some space to install door hinges, so **in both side of a door, at least one side of the frame need to connect to walkable area.**

For Example:
In first sample, there are two 2 * 4 areas in 2nd – 3rd row, the 1s in these two rows are not connected to wall, so those aren’t doors.
In second sample, there is one 2 * 4 area in 3rd – 4th rows, but there is not enough walkable area in 5th row, so it’s not a door.
In third sample, neither corner of each side connects to walkable area, so there is no door.
In fourth sample, there is a valid door in 3rd – 4th rows.

All other connected 0s are identified as different rooms. From the last sample, there are three rooms in total.

Now let’s label the rooms which different IDs. We identify the upper left corner is (0, 0). One room gets a smaller ID than the other if the upper left corner of this rooms is closer to (0, 0).
In the last sample, the ID of each room is identified as follow (* indicate the doors):

```
----------------------------------------------------------------------
-00000000000000000000000000000000000000000000000000000000000000000000-
-00000000000000000000000000000000000000000000000000000000000000000000-
---*****-----------------------------------------------*****----------
---*****-----------------------------------------------*****----------
-111111111----1111111111111-2222222222--22222222222222222222222222222-
--11111111----1111111111111-22222222222222222222222222222222222222222-
-11111111111111111111111111-22222222222222222222222222222222222222222-
-11111111111111111111111111-22222222222222222222222222222222222222222-
-1111--11111111111111111111-22222222222222222222222222222222222222222-
-1111--11111111111111111111-22222222222222222222222222222222222222222-
-1111----111111111111111111-22222222222222222222222222222222222222222-
-1111----111111111111111111-22222222222222222222222222222222222222222-
-11111111111111111111111111-22222222222222222222222222222222222222222-
-11111111111111111111111111-22222222222222222222222222222222222222222-
-11111111111111111111111111-22222222222222222222-2222222222-222222222-
-11111111111111111111111111-2222222222--22222222-2222222222-222222222-
----------------------------------------------------------------------
```

The main task of Pita, is to minimize the travel distances between different rooms. If two different rooms are connected by a door, the distance between them is 1.
Assuming there are n rooms in a house, now let’s define:

1. Dij. Which indicates the distance between room i and j;
2. Li. Which indicates max(Dij) (0 <= j < n).

Your task here is to:

1. Find out how many rooms are there in the house;
2. Find out min(Li) (0 <= i< n). To simplify the problem, you just need to tell us the ID of the room has minimal Li. Since there are multiple rooms meet the requirement, please output the minimal ID among them.

Input

The first line is an integer T (T<= 100), indicate the number of test cases.
Then n test cases follow.
First line of each test case contains two integers n and m (1 <= n, m <= 1000), indicate the size of the matrix.
Then m lines follow, each line contains a string with n characters, which only contains 0 and 1. Indicating the grids of this matrix.

It’s guarantee:

1. The house has only one way to separate rooms.
2. The total number of rooms in each house is not more than 100.

Output

For each test case, output “Case #t: r id”, t represents the index of test cases (start from 1), r represents the number of rooms, id represents the ID of the room has minimal Li.
If there is not any room in the house, please output “0 -1”

Sample Input

```
6
11 6
11111111111
10000000001
10000100001
10000100001
10000000001
11111111111
7 6
1111111
1000001
1000011
1000011
1000111
1111111
7 6
1111111
1000011
1000011
1000011
1000011
1111111
8 6
11111111
11000001
11000011
11000011
10000011
11111111
70 18
1111111111111111111111111111111111111111111111111111111111111111111111
1000000000000000000000000000000000000000000000000000000000000000000001
1000000000000000000000000000000000000000000000000000000000000000000001
1110000011111111111111111111111111111111111111111111111000001111111111
1110000011111111111111111111111111111111111111111111111000001111111111
1000000000111000000000000001000000000011000000000000000000000000000001
1100000000111000000000000001000000000000000000000000000000000000000001
```

```
1000000000000000000000000001000000000000000000000000000000000000000001
1000000000000000000000000001000000000000000000000000000000000000000001
1000011000000000000000000001000000000000000000000000000000000000000001
1000011000000000000000000001000000000000000000000000000000000000000001
1000011100000000000000000001000000000000000000000000000000000000000001
1000011100000000000000000001000000000000000000000000000000000000000001
1000000000000000000000000001000000000000000000000000000000000000000001
1000000000000000000000000001000000000000000000000000000000000000000001
1000000000000000000000000001000000000000000000001000000000010000000001
1000000000000000000000000001000000000011000000001000000000010000000001
1111111111111111111111111111111111111111111111111111111111111111111111
2 2
11
11
```

Sample Output

```
Case #1: 1 0
Case #2: 1 0
Case #3: 1 0
Case #4: 2 0
Case #5: 3 0
Case #6: 0 -1
```

Problem G: Scythe

Description

Scythe is an engine-building game set in an alternate-history 1920s period. In Scythe, each player represents a character from Eastern Europa who are attempting to earn their fortune and claim their faction's stake in the land around the mysterious Factory. Players conquer territory, enlist new recruits, reap resources, gain villagers, build structures, and activate monstrous machines.

The map of this game consists of n tiles, there are some roads between two tiles. To make this problem simpler, it’s guarantee the roads forms a tree.
There is one central tile which provides great benefit to players if they are able to reach the tile, so that tile is treated as the root of the tree.

As the player, you want to occupy some roads, to block other players from expanding their territory. However, you can’t occupy too many roads since it requires a lot of resources to do so. And you can’t occupy too little roads since it won’t be enough to block all other players.

After occupied some roads, the whole tree will be split into multiple subtrees. Your goal here is to make sure each subtree becomes a “safe piece”.
A subtree is considered as “safe”, if depth of this subtree is between given value **Dmin** and **Dmax** (inclusive).

Now, GDCPC committee want you to calculate what’s the minimal number of roads you need to occupy to achieve this.

Input

The first line is an integer T (T<= 200), indicate the number of test cases.
Then n cases follow.
Each test case start with three integer n (n<= 2000), Dmin, Dmax as what describe above.
Then n – 1 lines follow, the i-th (1 <= i< n) line contains a single integer p (0 <= p < n), which indicate there is a path between p and i.
0 is always the root of the tree.

Output

For each test case, output “Case #t: c”, t represents the index of test cases (start from 1), c represents the minimal number of roads need to occupy. If it’s impossible to do so, please output -1.

Sample Input	**Sample Output**

2
3 2 2
0
1
7 2 3
0
1
0
3
4
3
	Case #1: -1
	Case #2: 1

Problem H: Codename

Description

In Codenames, teams compete to see who can make contact with all of their agents first. Spymasters give one-word clues that can point to multiple words on the board. Their teammates try to guess words of the right color while avoiding those that belong to the opposing teams.

In fact, it’s proved that this game has significant effort to help people fall in love into each other! (Why? Don’t ask, play it!) Thus, this game attracts lots of boys and girls to play together.

There are n cute girls and m handsome boys going to play this game together. To help the GDCPC committee better coordinate this game, every girl is asked a question “How many boys you want to play with together?”
If EVERY girl want to play with the same boy, he is called “famous guy”. Now, please help to figure out AT LEAST how many famous guys are there, according to the girls’ answers.

Input

The first line is an integer T, indicate the number of test cases.
Then n cases follow.
The first line of each test case consists of two integers n and m (1 <= n <= 10^6, 1 <= m <= 10^8), which indicate how many girls and boys are going to participate in the game.
The second line of each test case consists of n integers, the i-th integer ai (0 <= ai <= m) indicate the answer of each girl.

Output

For each test case, output “Case #t: c”, t represents the index of test cases (start from 1), c represents there are AT LEAST c famous guys.

Sample Input	Sample Output

2
2 5
3 4
4 37
32 32 34 32
	Case #1: 2
	Case #2: 19

Problem I: Betrayal at House on the Hill

Description

Betrayal at House on the Hill is a tile game that allows players to build their own haunted house room by room, tile by tile, creating a new thrilling game board every time. The most exciting thing is, your character can travel between different floors in the house and create your own specified haunted house.

Now, let the game being!

Wait. We need to do some setup first. There are n floors of this haunted house, each floor has a starting tile for players to start exploring, we are putting all these tiles on a plain board.

During the game, player could travel between different floors, so they need to move their character token over and over again. Thus, if two tiles are too far away from each other, it makes players too difficult to do so.
However, if you are putting two tiles too close to each other, it will make players collide with others.

As the organizer, we decide to do follow approach to make the best placement:

1. Treat all tiles as points in the plain board
2. Find out n – 1 segments, each segment connects two different tiles
3. Segments can only intersect with each other on tiles
4. Any tile could connect to any other tile by using the segments
5. Minimize the total length of these segments

Given the position of n tiles, help us to calculate the minimal total length of these segments.

Input

The first line is an integer T (T <= 50), indicate the number of test cases.
Then T cases follow.
The first line of each test case consists of one integer n (1 <= n <= 100000), which indicate the number of tiles.
Then n lines follow. Each line consists of two integers xi, yi (0 <= xi, yi <= 1000000).

Output

For each test case, output “Case #t: d”, t represents the index of test cases (start from 1), d represents the minimal total length of the lines, keep the result in 2 digits decimal.

Sample Input	Sample Output

1
4
0 0
0 1
1 0
2 2
	Case #1: 4.24

Problem J: Carcassonne

Description

Carcassonne is a tile-based German-style board game for multiple players. The game board is a medieval landscape built by the players as the game progresses. The game starts with a single terrain tile face up and others shuffled face down for the players to draw from. On each turn a player draws a new terrain tile and places it adjacent to tiles that are already face up.

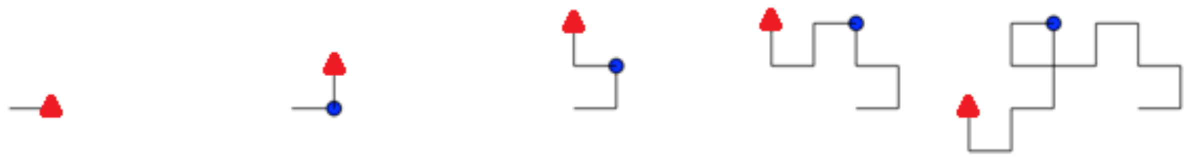
The Princess and the Dragon is an expansion of this game. A dragon figure is place on the board, and players take turns to move it over.

In GDCPC, we already place all tiles for you! But you need to move the dragon on the board yourselves.

The tiles we built is following Dragon Curve, the dragon figure is placed on the starting point of this curve.

Dragon Curve is a self-similar fractal curve. In 2D X-Y Axis, 1st degree curve is a base segment starts at (0, 0) and ends at (1, 0).
When constructing the 2nd degree curve, it crosswise rotates the 1st degree curve base on its ending point (i.e. (1, 0)), and then concatenate with the 1st degree curve. Thus 2nd degree curve is (0, 0) -> (1, 0) -> (1, 1).
When constructing the 3rd degree curve, it crosswise rotates the 2nd degree curve base on its ending point (i.e. (1, 1)), and then concatenate with the 2nd degree curve. Thus 3rd degree curve is (0, 0) -> (1, 0) -> (1, 1) -> (0, 1) -> (0, 2).
When this process continue infinitely, you can figure out the way this curve expands:
When constructing the n-th degree curve, it crosswise rotates the (n-1)-th degree curve base on its ending point, and then concatenate with the (n-1)-th degree curve.

Here is the how 1st – 5th degree curve looks like. The red triangles are the ending points of n-th degree curve, the blue dots are the ending points of (n-1)-th degree curve.



Base on the definition of dragon curve, it’ s obvious that the first half of n-th degree curve, is exactly same as (n-1)-th degree curve. Thus, we can treat Dragon curve is a singlular curve that can go infinite long.

Now let’ s go back to the game. In each turn, player will move the figure one step forward following the curve. Your task here, is to find out the position of the dragon figure after k turns. For example, in the 1st turn, it reaches (1, 0). In the 5th turn, it reaches (-1, 2). In 10th turn, it reaches (-3, 1).

Input

The first line of input contains one integer T (T <= 3000), indicate the number of test cases.
Then T lines follow.
Each test case contains one integer k (1 <= n<= 10 ^ 18), indicate the number of steps.

Output

For each test case, output “Case #t: (x, y)” , t represents the index of test cases (start from 1), (x, y) represents the coordination the dragon reaches at k-th turn.

Sample Input	Sample Output
5 1 5 10 666 11709394	Case #1: (1, 0) Case #2: (-1, 2) Case #3: (-3, 1) Case #4: (5, 21) Case #5: (2141, -703)

Hint

Some points may be reached multiple times, see 5th degree curve.

Problem K: Coup

Description

In Coup, there is political conflict in an Italian city-state, a city run by a weak and corrupt court. You need to manipulate, bluff and bribe your way to power. Your object is to destroy the influence of all the other families, forcing them into exile.

In this game, there are two import concepts: a) Faction and b) Political Distance.
Each player can be one of these three factions:

1. Neutralist
2. Reformist
3. Monarchist

At the beginning of the game, each player will be assigned one of these factions. In the game, it’ s possible for players to change their factions.

Each other player will then choose any another player as their supervisor, a value w will be assigned to this supervisor-ship, which indicate the effort for them to communicate (you know, some supervisors are really annoying and you don’ t want to deal with them). Players can only communicate with each other via their supervisors.
It's guarantee any two players can communicate with each other via their supervisors (either directly or indirectly).
Political Distance between two players is the total communication effort of these players so they can communicate.

During the game, players are more interest in the longest Political Distance between two players **if they are not in the same faction**.

To simplify the problem, you will need to handle three kinds of operations:

1. Change a player’s faction
2. Given a player, find out the longest Political Distance between this player, and any other player who are not in the same faction as the given player
3. Find out the longest Political Distance between any two players if they are not in the same faction.

Input

The first line is an integer T (T<= 100), indicate the number of test cases.
Then T cases follow.
The first line of each test case contains two integers n and m (1 <= n <= 50000, 1 <= m <= 50000), n indicates the number of players, m indicates the number of operations.

The second line of each test case contains n integers Fi (0 <= Fi <= 2), indicate the initial faction of each player.

Then n – 1 lines follow, each line contains three integers u, v, w (1 <= u, v <= n, 1 <= w <= 10000). Which means player u choose player v as supervisor, the communication effort between them is w.

Then m lines follow, each line represent one of the operations mention above:

1). Input “1 u F” (1 <= u <= n, 0 <= F <= 2), which means change player u’s faction to F

2). Input “2 u” (1 <= u <= n), please find out the longest Political Distance between **player u and any other player who are not in the same faction as player u**

3). Input “3”, please find out the longest Political Distance **between any two players if they are not in the same faction**

Output

For each test case, output “Case #t:” in the first line, t represents the index of test cases (start from 1).

For each query (i.e. operation 2 and 3), please output a single integer d which represent the longest Political Distance as mention above. Output 0 if there is no answer.

Sample Input	**Sample Output**

1
5 5
0 0 0 0 0
1 2 1
2 3 1
3 4 1
4 5 1
3
2 1
1 2 1
2 1
3
	Case #1:
	0
	0
	1
	3

Problem L: Advertising

Description

Maybe you have seen sports live, game live, etc. But you must have never seen GDCPC live before! Yes, the GDCPC committee decides to live stream the contest on their websites.

You know, when there is a video, there is an ad. The website developers want to insert ads into the live stream. Since it's the first time to serve ads, the website developers just want to use the most standard way, VAST (Video Ad Serving Template).

Basically, VAST is a XML and the video player plays ads inside it. An example looks like:

```
<VAST version="2.0">
  <Ad id="1">
    <InLine>
      <Creative AdID="601364" />
    </InLine>
  </Ad>
  <Ad id="2">
    <Wrapper>
      <VASTAdTagURI>http://demo.ad.com/vast.xml</VASTAdTagURI>
    </Wrapper>
  </Ad>
</VAST>
```

Inside a VAST, there are two kinds of ads: inline ad and wrapper ad.
For an inline ad, it contains one ID for interpret.
For a wrapped ad, it contains an URL, which wrap another VAST.

Video player interprets the VAST as following:

1. Check the ads inside the VAST sequentially
2. When encounter an inline ad, video player plays the media file inside it immediately, then move on to next ad.
3. When encounter a wrapper ad, video player need to fetch the wrapped VAST by its URL. Then interprets the wrapped VAST recursively. After the interpretation is done, move on to next ad.

Now the problem is to interpret the given VAST and output all inline ads IDs in order of appearance during interpretation.

Input

The first line is an integer T (T<= 50), indicates the number of test cases.

The first line of each test case contains one integer n indicates the number of VAST.

For each VAST:

1. First line contains an URL and an integer k, which indicates the number of ads in this VAST
2. Then k lines follow, each line could be either “id inline” or “id wrapper URL”

It’s guaranteed that in each test case

1. All VASTs have different URLs;
2. Total number of ads (both inline and wrapper) among these VAST is less than 1000;

the interpretation always succeeds.

Output

For each test case, output “Case #t: s”, t represents the index of test cases (start from 1), s represents the list of IDs when interpreting the first VAST, use space to separate the IDs.

If there is no ID when interpreting, s should be empty.

Sample Input	Sample Output
2 2 http://demo.ad.com/vast1 2 id1 wrapper http://demo.ad.com/vast2 id2 inline http://demo.ad.com/vast2 1 id3 inline 4 http://demo.ad.com/vast1 2 id1 wrapper http://demo.ad.com/vast2 id2 wrapper http://demo.ad.com/vast3 http://demo.ad.com/vast2 1 id3 wrapper http://demo.ad.com/vast4 http://demo.ad.com/vast3 2 id4 inline id5 inline http://demo.ad.com/vast4 1 id6 inline	Case #1: id3 id2 Case #2: id6 id4 id5