1a. Clock

Introduction

You have travelled back in time to a period before mobile phones, television, or even digital clocks. While it is relaxing and peaceful, you are struggling to wake up on time without an alarm. You have an analogue clock, and have invented an alarm for it. The alarm mechanism attaches to the hour and minute hands, and it rings when there is a chosen angle between the two hands.

Task

Given a list of times at which you might want to wake up, determine the angle between the hands at each of those times. The angle is measured clockwise from the hour hand to the minute hand, and is thus at least zero degrees and strictly less than 360 degrees.

Example

On the clock shown, it is 5:52. The minute hand is 136 degrees clockwise of the hour hand.

Input

The input contains multiple test cases. The first line contains T, the number of test cases. Each test case consists of a line containing two space-separated integers H and M, representing the time H:M.

Sample input

```
3
5 52
11 30
3 0
```

Output

For each test case, output a line of the form

```
Case #X: A
```

where X is the test case number, starting from 1, and A is the required angle in degrees. It is guaranteed that the answer will be an integer for all provided test cases, and A must be written as an integer (no decimal point).

Sample output

```
Case #1: 136
Case #2: 195
Case #3: 270
```

From http://poultrychamp.deviantart.com/art/Render-of-a-Clock-364390368

Constraints

It is guaranteed that in the input used to test your program:

- $1 \leq T \leq 20$
- $1 \leq H \leq 12$
- $0 \leq M \leq 59$
- The angle between the hands will be an integer number of degrees.

Note

If you are unfamiliar with analogue clocks: they have an hour hand and a minute hand (the short and long black arrows respectively in the picture). At 12:00 they both point at the 12, and they move continuously clockwise at fixed speeds. The minute hand goes around once in an hour, and the hour hand goes around once in twelve hours.

Time limit

2 seconds

1b. Clock

Standard Bank

Introduction

Having completed your alarm clock, you discovered that it has a flaw: while it rings when it is supposed to, it also rings at other times when the specified angle is reached. You've dealt with that by modifying the alarm to ring only if the angle is reached exactly at the start of a minute.

There is also a usability problem: just by looking at the mechanism, you can't immediately tell at what time your alarm will ring. You need a tool to convert an angle back to a time.

Task

Given a list of angles, determine the time at which the alarm will ring for each angle. As before, angles are measured in degrees, clockwise from the hour hand towards the minute hand. It is guaranteed that each angle will correspond to exactly one time on the clock which is a whole number of minutes.

Example

If the alarm is set with an angle of 136 degrees, it will ring at 5:52. At some point between 6:57 and 6:58 the angle will also be 136 degrees, but the alarm will not ring because it will not be at the start of a minute.

Input

The input contains multiple test cases. The first line contains T, the number of test cases. Each test case is a line containing an integer A, the angle used to set the alarm.

Sample input

```
3
136
195
270
```

Output

For each test case, output a line of the form

```
Case #X: H M
```

where X is the test case number, starting from 1, and H:M is the required time ($1 \leq H \leq 12$, $0 \leq M < 60$). Do not put any leading zeros in H or M (see the sample output).

Sample output

```
Case #1: 5 52
Case #2: 11 30
Case #3: 3 0
```

Constraints

It is guaranteed that in the input used to test your program:

- $1 \leq T \leq 20$
- $0 \leq A \leq 359$

Time limit

2 seconds

2. Sum of Two Primes

Standard Bank

Introduction

The Goldbach conjecture states that every even integer greater than 2 is the sum of two primes. (A prime is a positive integer with exactly two positive divisors.) Your friend is attempting to prove this conjecture and has asked for your help. He believes that it might be useful to know the number of ways in which a few integers can be written as the sum of two primes.

Task

Given a positive integer N, you are required to determine the number of ways it can be expressed as the sum of two primes. The sums $x + y$ and $y + x$ are considered to be the same.

Example

The number 16 can be expressed as $3+13$ and $5+11$, but no other combinations are possible. The number 3 cannot be expressed as the sum of two primes in any way.

Input

The first line of input contains an integer T, the number of test cases. The next T lines each contain a positive integer N.

Sample input

```
3
16
3
4
```

Output

For each test case, output a line of the form

```
Case #X: W
```

where X is the test case number, starting from 1, and W is the number of ways to express N as a sum of two primes.

Sample output

```
Case #1: 2
Case #2: 0
Case #3: 1
```

Constraints

It is guaranteed that in the input used to test your program:

- $1 \leq T \leq 100$
- $1 \leq N \leq 1\,000$

Time limit

1 second

3. Meeting

Standard Bank

Introduction

Standard Bank needs to schedule a meeting between a group of project managers. This is not easy, because project managers attend *many* meetings, so their calendars are already quite full. They need your help to find the best time for the meeting.

Task

Given a list of free times for each project manager, find the longest possible meeting that all of them can attend. If there is a tie, choose the earliest one.

Example

Since these project managers are so busy, they specify all times as a number of microseconds since the start of the day. Suppose there are three managers that have the following free times:

1. 0–3000, 5000–12000
2. 1000–2000, 4000–8000, 10000–12000
3. 500–7000, 9000–13000

The longest possible meeting is 2000 microseconds, either from 5000–7000 or from 10000–12000. The earlier of these is chosen, namely 5000–7000.

Input

The input contains multiple test cases. The first line contains T, the number of test cases. Each test case starts with a line containing N, the number of project managers. This is followed by N lines. The ith of these lines starts with F_i, the number of free slots that manager i has. This is followed by F_i pairs of integers $A_{i,j}$ and $B_{i,j}$, indicating the start and end of a free slot, in microseconds from the start of the day. All the integers on a line are separated by single spaces.

Sample input

```
2
3
2 0 3000 5000 12000
3 1000 2000 4000 8000 10000 12000
2 500 7000 9000 13000
2
2 0 10 1000 2000
1 10 1000
```

Output

For each test case, output a line of the form

```
Case #X: A B
```

where X is the test case number, starting from 1, and the best meeting time is from time A to time B (choosing the earliest meeting if there is a tie). If it is not possible to arrange a meeting lasting at least one millisecond, then use $A = 0$, $B = 0$.

Sample output

```
Case #1: 5000 7000
Case #2: 0 0
```

Constraints

It is guaranteed that in the input used to test your program:

- $1 \leq T \leq 20$
- $1 \leq N \leq 10$
- $0 \leq F \leq 1\,000$
- $0 \leq A_{i,j} < B_{i,j} \leq 1\,000\,000\,000$
- $B_{i,j} < A_{i,j+1}$

Time limit

2 seconds

4. Project Selection

Standard Bank

Introduction

Standard Bank's project managers finally decided on a time to meet. One of the items on the meeting's agenda is to decide which projects their team of contract workers should work on. They need your help to determine which projects should be undertaken in order to maximise the total income.

Task

Given the duration, initial value and depreciation rate of a list of projects, determine the maximum income that can be achieved by selecting and ordering the projects appropriately. Any number of projects may be undertaken, and there are no deadlines. However, the income received from a project depends on the time at which it is completed. The value depreciates linearly at a constant rate, starting from the day the team starts working on their first project. Thus, the income that will be received if a project with initial value v_i and depreciation rate of w_i per day is completed after t_i days, is:

$$p_i = v_i - w_i \times t_i$$

The team can only work on one project at a time, and a project cannot be completed more than once. The team must finish a project before starting the next one.

Example

Suppose there are three project proposals with initial value (v_i), depreciation rate (w_i) and duration (d_i) as outlined in the table below.

	v_i (R)	w_i (R/day)	d_i (days)
A	300	6	10
B	400	8	20
C	500	9	30

If project C is completed within the first 30 days, an income of R 230 will be received. Project B can then be completed within the next 20 days, but the project's value would reach zero on the day it is completed. Instead, the team can start working on project A immediately after completing project C. Project A will then be completed on day 40, which results in an additional income of R 60. The total income amounts to R 290.

Another option is to complete project A first, for an income of R 240, and then project B, for an income of R 160. This results in a total income of R 400, which is the maximum income that can be achieved.

Input

The input contains multiple test cases. The first line contains T, the number of test cases. Each test case starts with a line containing N, the number of project proposals. This is followed by N lines, each containing three space-separated integers, v_i, w_i and d_i, indicating the initial value, depreciation rate and duration of the ith project.

Sample input

```
1
3
300 6 10
400 8 20
500 9 30
```

Output

For each test case, output a line of the form

```
Case #X: M
```

where X is the test case number, starting from 1, and M is the maximum total income that can be achieved.

Sample output

```
Case #1: 400
```

Constraints

It is guaranteed that in the input used to test your program:

- $1 \leq T \leq 20$
- $1 \leq N \leq 100$
- $1 \leq v_i \leq 10\,000$
- $1 \leq w_i \leq v_i$
- $1 \leq d_i \leq \frac{v_i}{w_i}$

Time limit

5 seconds

5. Information Flow

Standard Bank

Introduction

You discovered that a mysterious group of individuals are living amongst us and are trying to gather some kind of intelligence. Each member has a number of other members to whom they will communicate any new information which they discover themselves or which they have heard from another member. You have managed to obtain a list of the members together with who each one sends information to. You do not yet know if the group has good or sinister intentions, but you are interested to find out more.

You have realised that when a member discovers information, this information might not necessarily reach every member of the group. You surmise that if there is a group member that does eventually hear all new information that the group discovers, then this group member would be the one you should contact. Of course, it is possible that the operation is completely decentralised and no such member exists.

Task

For each member of the group, you are given the members that they report information to. You must determine if there is a group member that will eventually hear of every piece of information, no matter who the first member of the group to discover it is, and if there is, report which one. If multiple members will hear everything, choose the one who appears first on the list of members.

Example

Suppose there are three group members, labelled 1, 2 and 3. Group member 1 reports to members 2 and 3 and group member 3 reports to member 1. Then group member 2 will be aware of every new piece of information.

But if member 1 were to stop reporting to member 2, then no member would hear all new information.

Input

The first line of input contains an integer T, the number of test cases. The first line of each test case contains a positive integer N, the number of group members. The next N lines each contain a number of spaces-separated integers. The i^{th} of these lines starts with a positive integer M_i, the number of members the i^{th} member reports to. This is followed by M_i further positive integers, the indices of the group members which the i^{th} member reports to.

Sample input

```
2
3
2 2 3
0
1 1
3
1 3
0
1 1
```

Output

For each test case, output a line of the form

```
Case #X: A
```

where X is the test case number, starting from 1, and A is smallest index of a member which eventually hears all new information or "`NONE`" if no such member exists. Note that the output format is case-sensitive.

Sample output

```
Case #1: 2
Case #2: NONE
```

Constraints

It is guaranteed that in the input used to test your program:

- $1 \leq T \leq 15$
- $1 \leq N \leq 40\,000$
- $1 \leq \sum_{i=1}^{N} M_i \leq 250\,000$
- $1 \leq$ each member index $\leq N$
- There will be no duplicate indices in the list of members that a given member reports to.

Time limit

5 seconds

6a. Vault Perimeter

Standard Bank

Introduction

Standard Bank wants to build new bank vaults that are monitored by video surveillance cameras with a 360° field of view. In order to maximise the size of the vaults while ensuring that the vaults are entirely covered by cameras, Standard Bank decided to build the walls of the vaults around the perimeter of the area covered by the cameras. They have already determined the layout of the cameras and want to know the length of the walls they will have to build.

Task

Given the position of each camera and the radius of the circular area covered by each camera, determine the perimeter of the area covered by the cameras. The fields of view of the cameras may, or may not, overlap. The perimeter of the overlapping parts should not be included in the answer.

Example

Figure 1 shows an example with four cameras. The cameras are located at $(-2, 0)$, $(2, 0)$, $(0, -4)$ and $(3, -5)$, respectively, and cover circles of radii 4, 4, 2 and 1, respectively. The perimeter of the area covered by the cameras is 41.89. The perimeter is drawn as a solid black line in Figure 1.

Input

The input contains multiple test cases. The first line contains T, the number of test cases. Each test case starts with a line containing N, the number of cameras. This is followed by N lines, each containing three space-separated integers, x_i, y_i and r_i, where (x_i, y_i) denotes the position of the ith camera and r_i the radius of the area it covers.

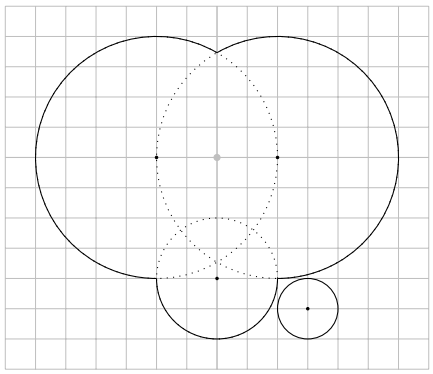

Figure 1: Example

Sample input

```
1
4
-2 0 4
2 0 4
0 -4 2
3 -5 1
```

Output

For each test case, output a line of the form

```
Case #X: P
```

where X is the test case number, starting from 1, and P is the length of the wall. P must be rounded to exactly 2 decimal places (refer to the section called *Floating-point output* in the contest manual for example code to do this). Answers that are within 10^{-2} of the correct answer will be accepted.

Sample output

```
Case #1: 41.89
```

Constraints

It is guaranteed that in the input used to test your program:

- $1 \leq T \leq 20$
- $1 \leq N \leq 500$
- $-1\,000\,000 \leq x_i, y_i \leq 1\,000\,000$
- $1 \leq r_i \leq 1\,000\,000$
- $(x_i, y_i) \neq (x_j, y_j)$ for all $i \neq j$

Time limit

5 seconds

6b. Vault Area

Standard Bank

Introduction

Standard Bank also wants to know the total floor area that will be available inside the vaults.

Task

Given the position of each camera and the radius of the circular area covered by each camera, determine the total area covered by the cameras. The field of view of the cameras may, or may not, overlap.

Example

Figure 1 shows an example with four cameras with the area covered by the cameras shaded in gray. The cameras are located at $(-2, 0)$, $(2, 0)$, $(0, -4)$ and $(3, -5)$, respectively, and have radii of 4, 4, 2 and 1, respectively. The total area covered by the cameras is 91.00.

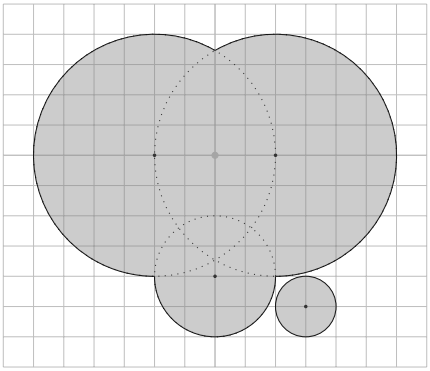

Figure 1: Example

Input

The format of the input is the same as in part a.

Sample input

```
1
4
-2 0 4
2 0 4
0 -4 2
3 -5 1
```

Output

For each test case, output a line of the form

```
Case #X: A
```

where X is the test case number, starting from 1, and A is the total area. A must be rounded to exactly 2 decimal places (refer to the section called *Floating-point output* in the contest manual for example code to do this). Answers that are within 10^{-2} of the correct answer will be accepted.

Sample output

```
Case #1: 91.00
```

Constraints

It is guaranteed that in the input used to test your program:

- $1 \leq T \leq 20$
- $1 \leq N \leq 500$
- $-1\,000\,000 \leq x_i, y_i \leq 1\,000\,000$
- $1 \leq r_i \leq 1\,000\,000$
- $(x_i, y_i) \neq (x_j, y_j)$ for all $i \neq j$

Time limit

5 seconds

7. Peg Jumping

Standard Bank

Introduction

Peg solitaire is a one-player board game that involves moving pegs on a board with holes. A valid move consists of jumping a peg over a neighbouring peg into an empty hole. The peg that was jumped over is removed from the board. The goal of peg solitaire is to remove all the pegs from the board except the peg that made the final jump.

For the IT Challenge, we will modify the game so that it is a two-player game. The goal is to remove more pegs from the board than your opponent. For each turn each player must select a peg to move and perform a sequence of valid jumps with that peg. This means that you can remove more than one peg with a single turn.

Task

The problem is an interactive task. You must write a program to play Peg Jumping against other programs. Our version is played on an $N \times N$ board. The initial board will be filled with pegs, except for N randomly placed holes. An example initial board with $N = 8$ is shown in Figure 1.

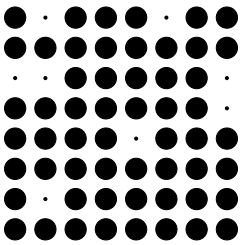

Figure 1: Example Initial Board

Each cell has a unique coordinate (R, C), designating the row and column respectively. The top-left cell is designated $(1, 1)$, the top-right cell is $(1, N)$, the bottom-left cell is $(N, 1)$ and the bottom-right cell is (N, N).

Given a cell at position (R, C) the following are its neighbours, if they exist:

- West: $(R, C - 1)$
- East: $(R, C + 1)$
- North: $(R - 1, C)$
- South: $(R + 1, C)$

The players take turns jumping a peg. For each jump the player scores a point. The game ends when there are no more valid jumps on the board. The player with the most points wins. However, if a player makes an illegal move or does not perform a move when a valid move exists, that player loses. If both players ends up with the same number of points, the game ends in a tie.

You must complete the `jump` method of the `PegJumpingPlayer` class. This method is called every turn with the current state of the board, your score and the opponent's score. The method must return the sequence of jumps of your next move. Your return must be a string describing the sequence of jumps performed by a single peg. The format of the string must be "R C [Jump sequence]", where (R, C) is the 1-based location of the peg to be moved and the jump sequence is a list of directions to be jumped in the specific order. The directions must be a combination of "N" for north, "S" for south, "E" for east or "W" for west. Your move must perform at least one jump. If you return an invalid move your program will be terminated and you will lose the game.

The state of the board is passed to your method as an array of N strings. Each string contains the information about a row of the board. The string will be N characters long. The "X" character denotes a peg and the "." character an empty space on the board.

You may also add initialisation code to the constructor, which is called at the beginning of each game with the current game's dimensions. You may add any other methods you need.

Example

Given the initial board in Figure 1, the jump "1 4 WS" will score two points. The state of the board after the jump is shown in Figure 2.

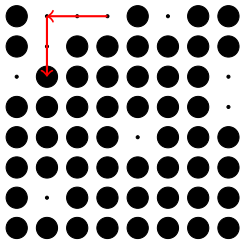

Figure 2: Example Jump

Tools

In the `peg_jumping_tools` directory on your drive are solution templates and a program for running games.

You will find a template program (in each language) on your drive, in the templates subdirectory of the `peg_jumping_tools` directory. Each template includes all the code needed to communicate with the other player;

7. Peg Jumping

Standard Bank

you need only write code to decide which move you should make next.

The program called `RunPegJumping` is a program for running games. The program can be run from a command-line interface. The graphical interface will be shown if no command-line arguments are provided.

The available command-line arguments are as follows:

- `-player1 COMMAND1` — `COMMAND1` specifies the executable command required to launch the program for player 1.
- `-player2 COMMAND2` — `COMMAND2` specifies the executable command required to launch the program for player 2.
- `-novis` — Optional. This argument disables visualisation.
- `-delay MS` — Optional. `MS` specifies the number of milliseconds delay between turns when visualisation is on.
- `-seed SEED` — Optional. `SEED` specifies the seed number used by the random number generator to generate the initial board.
- `-size N` — Optional. `N` specifies the size of the board.

The `COMMAND` parameters should specify how to launch your program. These are example commands for each language.

- C++ — `templates/pegjumping`
- Java — `java -cp templates pegjumping`
- Python — `python templates/pegjumping.py`

The following command will play a Java player against the random player, with a delay of 100 ms between moves and seed 5 used for board generation. The game will be visualised.

```
./RunPegJumping -delay 100 -seed 5
-player1 java -cp templates pegjumping
-player2 ./dummy
```

The graphical interface can be paused by pressing the space-bar. Pressing the space-bar again will continue play. Pressing any other key while the game is paused will step one move forward with each key press. The visualisation adds a delay per move; to time your program more accurately, disable the visualisation with the `-novis` argument.

Constraints

When evaluating your submission:

- $N = 16$

Timing

Your program is allowed a total of 10 seconds of processing time during each game. Time taken by your opponent or the referee does not count against your program.

Scoring

There are two available points for this problem, as well as two possible time bonuses. The first point will be awarded to any program which consistently beats a random player. The remaining point and time bonuses will be awarded in a tournament.

When you submit a solution, it will play four games against a random player — two as the first player and two as the second player. Your solution will be judged correct if it wins all the games.

A result of "wrong answer" means that your program lost at least one of the games. A result of "abnormal termination" means that your program exited (or crashed) before the game was over (unlike in non-interactive problems, exiting with an exit code of 0 before the end of the game is considered abnormal).

Once you have submitted a correct solution, you may continue to submit solutions, which will not change your score or total time. At the end of the contest the judges will take the teams' last correct submissions and place them in a tournament to determine which solution is the best. The submissions will be ranked in terms of number of wins to give an overall score for the submission.

The top three teams in this tournament will receive bonuses: the first team will receive an extra point, the second team will receive a deduction of 120 minutes from their total elapsed time, and the third team will receive a deduction of 60 minutes from their total elapsed time.

If your program does not interact validly with the server for a given game, then you will automatically lose the game. A team must interact validly in at least one game to be eligible for a bonus. In the event of a tie, all teams tied for a position will receive the corresponding bonus.

Live tournament

During the contest, the judges will run games with accepted solutions from all teams, and show these games on a screen in the judging room. The results of these games will not be used in scoring. These games may not update immediately when a new submission is accepted. The live games will not be run during the final hour of the competition.