String Inversions

Filename: `inversions`

Suppose we have a string of *n* lowercase letters. Let *i* and *j* represent two arbitrary indices in the string. An inversion is defined as a pair of integers *i* and *j* such that $i < j$ and the character in the string at position *i* is lexicographically greater than the character at position *j*. For example, “ab” has no inversions, “ba” has one inversion, and “baba” has three inversions. Given a string, can you count the number of inversions?

The Problem:

Count the number of inversions in a given string.

The Input:

The first line of the input file begins with a single, positive integer, *t*, representing the number of test cases. Each test case contains two lines. The first contains a single integer $1 \le n \le 10^5$, denoting the length of the string. The second line contains the string itself, which consists of only lowercase letters.

The Output:

For each test case, output a single line containing “`String #i: c`” without the quotes, where `i` is the test case number, and `c` is the number of inversions in the string.

Sample Input:

```
3
5
danny
4
lior
7
natalie
```

Sample Output:

```
String #1: 1
String #2: 1
String #3: 12
```

Lazy Bob and Lime Bike

Filename: `limebike`

Lazy Bob is back and he is lazier than ever. In fact, the only time he is not lazy is when he is solving flow problems. One day, Lazy Bob was heading to class on campus but was running extremely late. He did what any UCF student would do and hopped on the nearest Lime Bike. And just like most other Lime Bike users, when he got to class he left his bicycle in a very inconvenient place: blocking an entire walkway!

UCF's campus can be modeled as a series of nodes and edges, with the buildings being the nodes and walkways (or bikeways) between them being edges.

The Problem:

Given the edge Lazy Bob blocks, determine how many unordered pairs of buildings are no longer connected (if they were before).

The Input:

The first line of input is a positive integer t, the number of days where Lazy Bob took a lime bike to class.

For each day the input is as follows. The first line has two integers $1 \leq n, m \leq 10^5$, the number of buildings and walkways, respectively. Then m lines follow, each with two positive integers $1 \leq a, b \leq n, a \neq b$, indicating that there is a walkway between buildings a and b. One line follows with a single integer $1 \leq e \leq m$, the index of the walkway Lazy Bob blocks. Walkways are indexed by the order they appear in the input (starting from ***1*** in each case).

The Output:

For each test case, output a single line saying "`Day #i: c`" without the quotes, where `i` is the test case number, and `c` is the number of unordered pairs of buildings such that one could reach one from the other before Lazy Bob left his Lime Bike but not after.

(Sample Input and Output are on the next page)

Sample Input:

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

Sample Output:

```
Day #1: 0
Day #2: 4
```

Timothy's Marbles

Filename: `marbles`

Timothy loves geometry! For his sixth birthday, he received a box with *n* marbles from his parents. He immediately began playing with them by arranging them along a circle. Timothy began wondering, how many different polygons can he make by drawing lines between the marbles? Two polygons are different if they are composed out of a different set of marbles.

The Problem:

Given *n* marbles along a circle, count the number of unique polygons that can be made. As the answer can be very large, print it modulo 1,000,000,007.

The Input:

The first line of the input file begins with a single, positive integer, *t*, representing the number of test cases. For each test case, a single line follows containing a single integer $1 \leq n \leq 10^4$ representing the number of marbles.

The Output:

For each test case, output a single line saying "`Marble Set #i: c`" without the quotes, where `i` is the test case number, and `c` is the number of polygons Timothy can make modulo 1,000,000,007.

Sample Input:

```
3
2
3
5
```

Sample Output:

```
Marble Set #1: 0
Marble Set #2: 1
Marble Set #3: 16
```

Octo-Octothorpes

Filename: `octothorpe`

There are several names for the symbol that appears when you hold the shift key and press 3 on a regular keyboard. Mathematicians commonly call it a number sign. German students sometimes call it a bicrux (meaning two crosses, one in the top left, one in the bottom right). Teenagers usually call it a hashtag. Telephone salesmen often call it a pound sign. Musical theorists and Microsoft developers call it a sharp sign. We prefer to call it an octothorpe.

Let's define a k-octothorpe. A 1-Octothorpe is simply 25 #'s and .'s in the shape of a # character as shown below:

```
.#.#.
#####
.#.#.
#####
.#.#.
```

A 2-octothorpe is the same symbol, but with all the periods replaced with a 5x5 grid of periods, and all #'s replaced by the above 5x5 1-octothorpe. Similarly, for any k > 1, a k-octothorpe is a (k-1)-octothorpe with all periods and #'s replaced as described above. For example, an 8-octothorpe (or octo-octothorpe, if you will) is created by taking a 7-octothorpe and replacing all #'s with the 5x5 1-octothorpe and all periods with a 5x5 grid of periods. A 2-octothorpe is shown below the sample output.

Please find the *n*th character of a *k*-octothorpe, if the characters are numbered starting at 1 in row-major order (that is, first numbered left-to-right on the top row, then left-to-right on the second row, et cetera).

The Problem:

Given *n* and *k*, find the *n*th character of a *k*-octothorpe.

The Input:

The first line of the input file begins with a single, positive integer, *t*, representing the number of test cases. Each test case begins starts with two integers $1 \le n, k \le 10^{18}$, representing the 1-based index of the requested character and the type of octothorpe respectively. For all test cases, it is guaranteed that a *k*-octothorpe contains at least *n* characters.

The Output:

For each test case, output a single line of the form "`Request #i: c`" without the quotes, where `i` is the number of the test case, and `c` is either "#" or ".".

Sample Input:

```
4
14 1
100 2
31 2
463 2
```

Sample Output:

```
Request #1: #
Request #2: .
Request #3: #
Request #4: #
```

Note: The requested locations are shown below in red.

```
          .#.#.
          #####
          .#.#.   ← Request 1
          #####
          .#.#.
```

```
......#.#.......#.#......
.....#####.....#####.....  ← Request 3
......#.#.......#.#......
.....#####.....#####.....  ← Request 2
......#.#.......#.#......
.#.#..#.#..#.#..#.#..#.#.
#########################
.#.#..#.#..#.#..#.#..#.#.
#########################
.#.#..#.#..#.#..#.#..#.#.
......#.#.......#.#......
.....#####.....#####.....
......#.#.......#.#......
.....#####.....#####.....
......#.#.......#.#......
.#.#..#.#..#.#..#.#..#.#.
#########################
.#.#..#.#..#.#..#.#..#.#.
#########################  ← Request 4
.#.#..#.#..#.#..#.#..#.#.
......#.#.......#.#......
.....#####.....#####.....
......#.#.......#.#......
.....#####.....#####.....
......#.#.......#.#......
```

Pizza Dog

Filename: `pizza`

Pizza Dog is a superhero dog who delivers pizza to those in need. For the holiday season, Pizza Dog decided to go on a generous mission to deliver pizza to everybody! However, since pizza is not infinite, he decided to provide each household with enough pizzas such that each family member has their own. Given the number of family members in each household and the number of pizzas that each household already owns, determine the total number of pizzas which Pizza Dog delivered.

The Problem:

Determine the number of pizzas delivered by Pizza Dog.

The Input:

The first line of the input file begins with a single, positive integer, *t*, representing the number of test cases. Each test case consists of multiple lines. A test case a single line follows containing a single integer $1 \leq n \leq 100$, the number households. *n* lines follow, each with two integers, $1 \leq a \leq 1000$, and $0 \leq b \leq 1000$, the number of family members in the house and the number of pizzas they own, respectively.

The Output:

For each test case, output a single line saying "`Neighborhood #i: c`" without the quotes, where `i` is the test case number, and `c` is the number of pizzas delivered. Make sure to double check spelling!

Sample Input:

```
2
5
5 0
5 0
5 1
4 0
1 0
3
2 4
11 0
4 4
```

Sample Output:

```
Neighborhood #1: 19
Neighborhood #2: 11
```

Quantum Network

Filename: `quantum`

Qupee is a subatomic particle living in a network of nodes. Due to the laws of quantum mechanics, we can never know where Qupee is with certainty at any point in time, since by finding its position we alter it. However, we do know some things. At time t = 0, Qupee may be at any of the nodes in the network with equal probability. At every integer time afterwards, Qupee moves to a random node that is adjacent to its current position with uniform probability. For example, if three nodes are adjacent to Qupee's current position, the probability that Qupee will move to each of these nodes is ⅓. We wish to determine the probability that Qupee has passed through node *x* by time t = *k*, inclusive.

The Problem:

Given a network of nodes, determine the probability that Qupee has passed through node *x* by time t = *k*.

The Input:

The first line of the input file begins with a single, positive integer, *t*, representing the number of queries. For each query, multiple lines follow. The first contains a four integers $1 \le n, m \le 100$, representing the number of nodes and the number of adjacencies in the network, as well as $1 \le x \le n$ and $0 \le k \le 100$, as described in the problem statement. The next *m* lines include two integers $1 \le u, v \le n, u \ne v$, meaning that nodes *u* and *v* are adjacent. It is guaranteed that through a series of adjacencies, Qupee can reach each node from every other node.

The Output:

For each query, output a single line saying "`Network #i: c`" without the quotes, where `i` is the number of the query, and `c` is the probability that Qupee passed through node *x* at time t = *k*. Find the probability as an integer fraction P/Q, and print it in the form $P*Q^{-1}$ modulo 1,000,000,007.

Sample Input:

```
3
3 2 1 3
1 2
2 3
3 2 2 2
1 2
2 3
6 6 1 0
1 2
2 3
```

```
3 4
4 5
5 6
6 1
```

Sample Output:

```
Network #1: 750000006
Network #2: 1
Network #3: 166666668
```

Note: The probabilities in the samples are equal to: ¾, 1, and ⅙, respectively.

Upvote Bots

Filename: `upvote`

Sharon frequently browses a website known as NorseHorses, a forum for rating and ranting about various viking mounts. His ultimate goal is to become the most popular user on that website. To do so, he frequently posts blogs and comments with lame jokes and overused memes in order to get upvotes. In order to get the most upvotes on the website, Sharon came up with his greatest scheme yet.

Sharon has *n* friends around the world, each having some number of bots at their disposal. If one of his friends is awake at the time Sharon posts a blog, they will use their bots to instantly grant upvotes equal to the number of bots. By doing so, he hopes that the bandwagon effect will kick in and no matter how low quality his post is, readers will continue to upvote it!

In order to be most effective, Sharon would like to know the maximum number of upvotes he can have on his blog post, assuming he posts it at an optimal time.

The Problem:

Given the number of bots owned by each of Sharon's friends, as well as the range of time they are awake, determine the maximum number of upvotes Sharon can get.

The Input:

The first line of the input file begins with a single, positive integer, *t*, representing the number of blogs Sharon will write. For each blog, input consists of multiple lines. The first of which consists of a single integer, $1 \leq n \leq 500$, the number of friends Sharon has. *n* lines follow, the *i*th of which consisting of an integer $1 \leq x \leq 10{,}000$ and two "times", *a* and *b* ($a \neq b$), indicating that the *i*th friend has *x* bots and is awake between time *a* and time *b*, inclusive. A time is formatted as four integers of the form "HH:MM", representing time in a 24-hour digital clock. See examples below.

The Output:

For each test case, output a single line saying "`Blog #i: c`" without the quotes, where `i` is the test case number, and `c` is the maximum number of upvotes Sharon can get.

(Sample Input and Output are on the next page)

Sample Input:

```
3
3
10 00:30 14:59
10 08:15 16:00
10 13:00 23:50
3
2 22:22 14:44
3 13:37 16:00
1 15:00 16:00
1
502 00:00 23:59
```

Sample Output:

```
Blog #1: 30
Blog #2: 5
Blog #3: 502
```

Note:

In the first case, if Sharon posts a blog at 14:00 (2PM), all three of his friends can upvote him.

In the second case, Sharon can post a blog at 13:37. His first and second friends will be awake, but the third friend will not, granting him 5 upvotes.

In the third case it does not matter when Sharon posts his blog because his friend is awake every minute of the day.