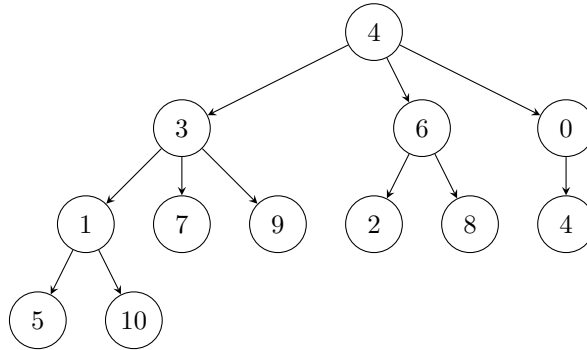


Problem A. Mostrando grafos

Input file: Standard input
Output file: Standard output
Time limit: 1 second
Memory limit: 64 megabytes

Dado un árbol $T = (V, E)$ como un conjunto de vértices $V = \{1, 2, \dots, n\}$, y un conjunto de aristas $E = \{(p, q) / p, q \in V\}$ con la condición de que existe un nodo raíz $r \in V$, y que hay $n - 1$ aristas. Por conveniencia, se considerará un grafo dirigido, y para cada elemento $(p, q) \in E$, indicará que existe arista desde p hacia q . Por ejemplo, la siguiente figura muestra un árbol de con 12 nodos.



Tu deber es recorrer el grafo y mostrarlo por niveles, es decir, primero la raíz (nodo 4), luego sus hijos (nodos 3, 6, y 0), luego los hijos de estos nodos q se encuentran en el siguiente nivel (nodos 1, 7, 9, 2, 8, y 4), y así sucesivamente. Para el ejemplo de la figura anterior, la respuesta que se busca sería la siguiente:

```
4
3 6 0
1 7 9 2 8 4
5 10
```

Input

El problema contiene varios casos de prueba. La primera línea es un entero T ($1 \leq T \leq 10^2$) que denota el número de casos de prueba. Cada caso está compuesto por dos líneas, en la primera de ellas se encuentra el número entero N indicando la cantidad de nodos que tendrá el grafo ($1 \leq N \leq 10^3$). Luego, en la segunda línea del caso de prueba se encuentran $N - 1$ pares de números p, q ($0 \leq p, q < n$, $p \neq q$) que representan las aristas del árbol en cuestión. El nodo raíz es determinado como aquel nodo que no tiene aristas entrantes, más sí aristas salientes.

Output

Para cada caso de prueba, el programa deberá imprimir “Caso #i:” (sin comillas) y a partir de la siguiente línea el grafo por niveles.

Example

| Standard input | Standard output |
|---------------------------------------|---|
| 2 2 0 1 5 0 1 1 2 2 3 3 4 | Caso #1: 0 1 Caso #2: 0 1 2 3 4 |

Problem B. Contador de direcciones IP

Input file: Standard input
Output file: Standard output
Time limit: 1 second
Memory limit: 64 megabytes

Xavier, un programador muy hábil fue contratado por una gran compañía de software llamada TORO Out of Control Technologies (TORO-OTC). Recientemente, había surgido un problema masivo con las direcciones IP de todas las computadoras de la compañía (aprox. 1000 equipos) y necesitaban con urgencia poder saber la cantidad de direcciones IP que seguían un determinado patrón.

Muy brevemente, una dirección IP (la empresa sólo trabaja con la versión 4 por el momento) es una etiqueta asignada a un dispositivo dentro de una red y está representado por un número de 32 bits. Dicho número, para ser fácilmente leible por nosotros, los humanos, es presentado como 4 números decimales (que varían de 0 a 255) separados por un punto. Por ejemplo, algunas direcciones IP son:

```
192.168.1.1   192.168.0.1   172.0.0.1
192.168.1.13  192.168.0.15  172.0.0.15
192.168.1.14  192.168.0.16  172.0.0.16
192.168.1.15  192.168.0.17  172.0.0.17
```

El problema que TORO-OTC tenía yace básicamente en saber cuantas direcciones fueron asignadas siguiendo un determinado patrón. Un patrón era una dirección IP que podía contener asteriscos (*) los cuales reemplazaban números (entre 0 y 9), pero también un asterisco puede reemplazar números en el rango de 0-255 siempre y cuando no haya ningún otro número que condicione la naturaleza del asterisco — para mayor claridad, consultar los ejemplos. Considerando las direcciones IP proporcionadas, la Tabla 1 muestra algunos ejemplos del tipo de consultas que Xavier debería de responder.

Table 1

| Patrón | Número de direcciones |
|--------------|-----------------------|
| 192.168.1.* | 4 |
| 192.168.*.15 | 2 |
| 192.*.*.* | 8 |
| 192.168.0.14 | 0 |
| 192.168.1.1 | 1 |
| 1*2.*.*.1* | 6 |
| 1*2.*.*.* | 8 |
| *.*.*.* | 12 |

Input

El problema contiene varios casos de prueba. La primera línea es un entero T ($1 \leq T \leq T$) que denota el número de casos de prueba. Cada caso inicia con una línea “Inicio” y termina con una línea “Fin” (sin comillas) y está compuesto por una lista de consultas en medio. Las consultas pueden ser de dos tipos. “Agregar IP”, donde IP tendrá el formato de una IP válida antes expuesta, y “Contar PATRON” que define el patrón sobre el cual se debe de contar las IPs almacenadas hasta el momento. Se garantiza que no se agregarán IPs repetidas para un mismo caso de prueba.

Output

Para cada caso de prueba, el programa deberá imprimir “Caso #i:” en una nueva línea (sin comillas) y a continuación las respuestas a las consultas. Cada respuesta debe de ir en una línea.

Example

| Standard input | Standard output |
|---|--|
| 2 Inicio Contar 192.168.1.1 Agregar 192.168.1.1 Agregar 192.168.1.10 Agregar 192.158.1.1 Contar 192.168.1.* Fin Inicio Agregar 172.0.0.1 Agregar 192.0.0.1 Contar 1*2.*.*.1 Contar *.*.*.* Fin | Caso #1: 0 2 Caso #2: 2 2 |

Problem C. Nombre del Problema

Input file: Standard input
Output file: Standard output
Time limit: **TIEMPO** second
Memory limit: 64 megabytes

Descripción del problema...

Input

El problema contiene varios casos de prueba. La primera línea es un entero T ($1 \leq T \leq 10^2$) que denota el número de casos de prueba. Cada caso está compuesto por dos líneas, en la primera de ellas se encuentran los números enteros N y K , siendo N el número de piedras medidas ($1 \leq N \leq 10^5$, $1 \leq K \leq 10^9$). Luego, en la segunda línea del caso de prueba se encuentran N enteros distintos a_1, a_2, \dots, a_N que representan las alturas de las piedras ($1 \leq a_i \leq 10^9$).

Output

Para cada caso de prueba, el programa deberá imprimir la longitud del mayor conjunto libre de K -múltiplos que se pueda obtener de la lista de alturas.

Example

| Standard input | Standard output |
|----------------|-----------------|
| 2 | 1 |
| 2 3 | 3 |
| 1 3 | |
| 6 2 | |
| 2 3 6 5 4 10 | |