Estructuras de datos

Grados de la Facultad de Informática (UCM). Grupos A, C

Examen Convocatoria Ordinaria, 17 de junio de 2021.

Normas de realización del examen

1. El examen dura 3 horas.
2. Debes programar soluciones para cada uno de los tres ejercicios, probarlas y entregarlas en el juez automático accesible en la dirección `http://exacrc`. Para la entrega el juez sólo tiene los datos de prueba del enunciado del problema.
3. Escribe comentarios que expliquen tu solución, justifiquen por qué se ha hecho así y ayuden a entenderla. Calcula la complejidad de todas las funciones que implementes.
4. En el juez te identificarás con el nombre de usuario y contraseña que has recibido al comienzo del examen. El nombre de usuario y contraseña que has estado utilizando durante la evaluación continua **no** son válidos.
5. Escribe tu **nombre y apellidos** en un comentario en la primera línea de cada fichero que subas al juez.
6. Dispones de un fichero plantilla para cada ejercicio.
7. Tus soluciones serán evaluadas por el profesor independientemente del veredicto del juez automático. Para ello, el profesor tendrá en cuenta **exclusivamente** el último envío que hayas realizado de cada ejercicio.
8. Si necesitas consultar la documentación de C++, está disponible en `http://exacrc/cppreference`.

Ejercicio 1. Intersección de dos listas ordenadas

(2 puntos) La intersección de dos listas ordenadas y sin repeticiones es la lista ordenada que contiene los elementos que tienen ambas listas. Por ejemplo, la intersección de la lista `1, 3, 4, 5, 8, 9` con la lista `2, 4, 8, 10` es la lista `4, 8`.

Queremos extender *mediante herencia* la clase `deque<int>`, que implementa las colas dobles de enteros mediante listas de nodos dinámicos, doblemente enlazados, circular y con nodo fantasma, con un nuevo método `interseccion` que recibe como argumento una lista ordenada y modifica la lista de `this` para que termine teniendo la intersección de ambas listas. Además debe dejar la lista argumento sin modificar.

En la implementación del nuevo método no pueden hacerse nuevos `new`s, ni *copiar* los enteros de un nodo a otro. También hará falta un método `print` para mostrar por pantalla los elementos de la lista, de inicio a fin.

No modifiques ni subas al juez el fichero `deque_eda.h` cuya clase `deque` debes extender.

Entrada

La entrada consta de una serie de casos de prueba. La primera línea contiene el número de casos de prueba que vendrán a continuación. Cada caso ocupa dos líneas. Cada una de estas líneas representa una de las listas, y contiene sus elementos ordenados de menor a mayor, una serie de números entre 1 y 1.000.000, seguidos de un 0, que marca el final de la descripción de la lista, sin pertenecer a ella. En cada lista, todos sus elementos son distintos.

Salida

Para cada caso de prueba se escribirá en una línea la lista modificada tras hacer la intersección de ambas listas.

Entrada de ejemplo

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

Salida de ejemplo

```
4 8

1 2 3
```

Ejercicio 2. El regreso de los enanos

(2 puntos) Con la caída de Erebor termina el reinado de Thror y los enanos liderados por Thorin se desperdigan por la tierra media. Pasados los años sin que se haya vuelto a ver al dragón Smaug, Thorin convoca a los enanos para reconquistar la montaña solitaria. Los enanos abandonan sus aldeas para acudir a la llamada. El punto de encuentro es la casa de Bilbo Bolsón en La Comarca.

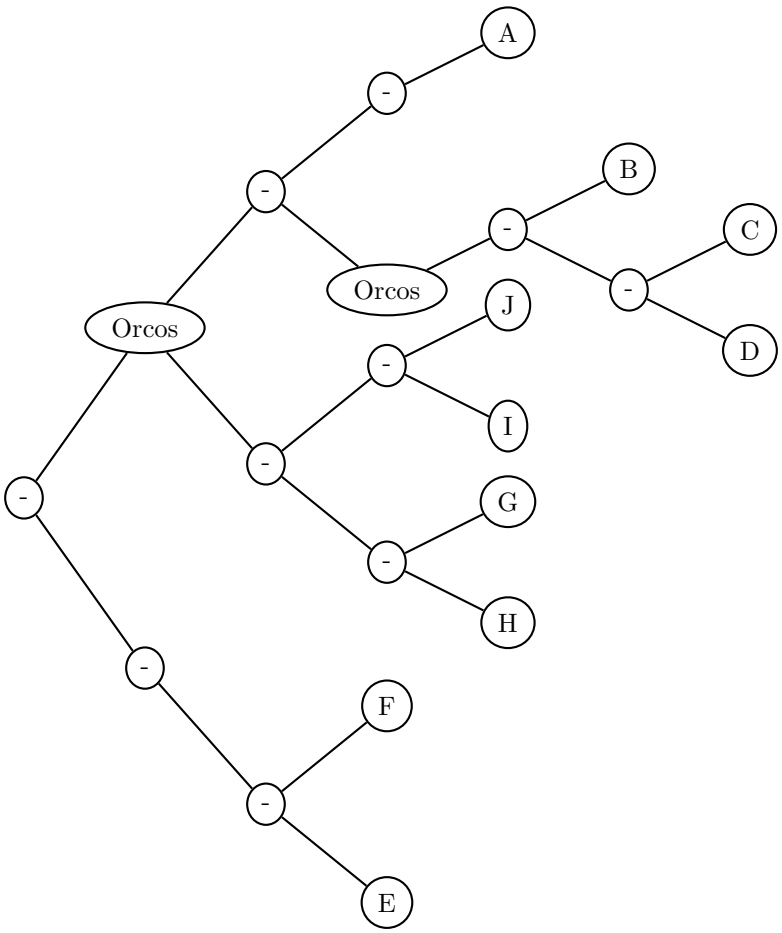
Por el camino se encuentran y van formando grupos, cada uno liderado por uno de los enanos. Cuando dos grupos se encuentran, el nuevo líder es el del grupo más numeroso. Si los dos grupos tienen el mismo número de enanos, eligen al que tiene nombre menor en orden alfabético.

Durante su marcha pueden sufrir ataques de los orcos que invariablemente reducen a la mitad el grupo (en los grupos de tamaño impar se redondeará por abajo). Todos los enanos están dispuestos a morir antes que su líder. Éste solo muere si el grupo desaparece completamente.

Sabiendo cómo se van encontrando los enanos y los ataques de orcos que sufrirán en su camino, debes determinar cuántos enanos consiguen llegar a La Comarca y quién llega liderando el grupo.

Ejemplo. En el siguiente ejemplo tenemos varios enanos que parten a reunirse con su jefe representados con letras. Los enanos D y C se encuentran quedando como líder C por ser dos grupos con el mismo número de enanos y C ser menor alfabéticamente. A continuación se encuentran con B y queda como líder C por ser el líder del grupo más numeroso. Estos tres enanos sufren un ataque orco en el que mueren dos de ellos. El superviviente es el líder C.

A continuación se junta con el enano A que viene por el camino de la izquierda quedando liderados por A. Estos se encuentran con los 4 que vienen por su derecha liderados por G, que se proclama líder por ser su grupo más numeroso. Sufren un ataque orco quedando reducidos a 3. Al llegar a La Comarca se juntan con un grupo de 2 liderado por E. Cuando se encuentran con Thorin son 5 liderados por G.



Requisitos de implementación.

Se implementará una función `enanos` que resuelva el problema. Se indicará en un comentario el coste de la función implementada, para árboles equilibrados y para el caso peor.

Entrada

La entrada comienza con el número de casos de prueba. Para cada caso se muestra el recorrido en preorden del árbol que representa cómo se van uniendo los enanos y los ataques que sufren (el recorrido puede mostrarse en varias líneas). Los enanos comienzan su andadura siempre desde nodos hoja y su valor es una cadena de caracteres sin blancos. Los orcos se encuentran siempre en nodos internos y su valor es la cadena de caracteres `Orcos`. Los nodos internos que representan lugares de paso o la unión de dos grupos de enanos tienen como valor un guión. Los árboles vacíos se representan con el carácter punto.

El árbol siempre tiene al menos un elemento.

Salida

Para cada caso de prueba se escribirá en una línea el nombre del líder del grupo que llega, seguido del número de enanos del grupo. Si no consigue llegar ningún grupo, se escribirá la cadena `Ninguno`.

Entrada de ejemplo

```
4
- Orcos - - A . . . Orcos - - D . . C . . B . . . - - H . . G . . - I . . J . . -
   . - K . . L . .
- Orcos Dori . . Balin . . Orcos Nori . . Fili . .
- Orcos . Ori . . Orcos Orcos Bombur . . Fili . . Orcos Gloin . . .
Orcos - Kili . . Nori . . Balin . .
```

Salida de ejemplo

```
G 5
Balin 2
Ninguno
Kili 1
```

Autor: Isabel Pita

Ejercicio 3. Smart farming.

(3 puntos) El sector agrícola está empezando a aplicar técnicas de bigdata para mejorar la productividad. Vamos a diseñar un sistema que nos permita detectar diversas plagas para proceder a fumigar adecuadamente las zonas en las que se producen. La cooperativa agrícola ha dividido sus tierras en sectores, en los que ha situado unos robots capaces de diferenciar distintos tipos de plagas. Cuando detectan una plaga los robots envían al sistema la plaga detectada, el sector en el que está y el número de plantas que están afectadas. Un sector puede tener sus plantas afectadas por varios tipos de plagas. La cooperativa de vez en cuando alquila una avioneta y fumiga los sectores más afectados.

La implementación hará uso de los tipos `sector` y `plaga` que se representan con un `string`.

Las operaciones son las siguientes:

- `alta(id, n)`: da de alta un nuevo sector `id` con `n` plantas. Si el sector ya estaba dado de alta se incrementa su número de plantas con el valor recibido como parámetro.
- `datos(id,p,n)`: el sistema recibe notificación de una plaga `p` desde el sector `id` con `n` plantas afectadas. Las plantas afectadas por una plaga continuan afectadas hasta que se fumiga ese sector, por lo tanto si el robot manda varios avisos sobre la misma plaga en el mismo sector antes de que haya una fumigación cada aviso acumulará las plantas del aviso anterior (el número de plantas del aviso anterior para esa plaga en ese sector se desprecia porque está incluido en los nuevos datos).

  Si el `id` del sector no está dado de alta en el sistema se lanzará una excepción con el mensaje `Sector no existente`. Si la plaga no se encuentra registrada en ese sector se registrará y si el número de plantas afectadas es mayor que el número de plantas del sector se lanzará una excepción con el mensaje `Numero de plantas incorrecto`.
- `fumigar()`: la operación realiza dos acciones. Por un lado, devuelve un vector con los sectores que tienen un cuarto, o más, de sus plantas afectadas por una plaga en el momento en que se produce la notificación (infectadas >= plantas / 4.0). Por otro lado, realiza la fumigación de esos sectores.

  En el vector se indicará el sector afectado y la plaga de que se trata. Los sectores aparecen en el vector *en el orden en que fueron alcanzando el cuarto de sus plantas afectadas.* El vector no tendrá elementos repetidos, si un par (sector, plaga) ya se encuentra en el vector no se añadirá otra vez. Recuerda que las plantas solo sanan cuando se realiza esta operación, mientras tanto continuan infectadas.

  Si no hay ningún sector en estas condiciones se devuelve el vector vacío.

  Los sectores fumigados contra una plaga no necesitan volver a fumigarse contra esa plaga hasta que vuelvan a tener un cuarto de sus plantas afectadas. Se supone que al fumigar todas las plantas quedan sanas.
- `plagas(id)`: Obtiene en un vector todas las plagas que ha sufrido el sector, que se da cómo parámetro, desde que se dió de alta en la aplicación en orden alfabético, junto a cada plaga se indicará el número actual de plantas afectadas por la plaga que todavía no han sido tratadas. Si el `id` del sector no está dado de alta en el sistema se lanzará una excepción con el mensaje `Sector no existente`.

Requisitos de implementación.

Seleccionar los tipos de datos adecuados para representar la información y obtener una solución eficiente. En la cabecera de cada función debe indicarse el coste de la misma.

Los métodos del TAD no deben mostrar nada por pantalla. El manejo de la entrada y salida de datos se realizará en funciones externas al TAD.

Debe justificarse el tipo representante elegido y dar el coste de todas las operaciones

Entrada

La entrada consta de una serie de casos de prueba. Cada caso está formado por una serie de líneas, en las que se muestran las operaciones a llevar a cabo, una por cada línea: el nombre de la operación seguido de sus argumentos. La palabra FIN en una línea indica el final de cada caso.

Salida

Para cada caso de prueba se escribirán los datos que se piden. Las operaciones que generan datos de salida son:

- `fumigar`, que debe escribir una línea del tipo `Fumigar la plaga` *nombre de la plaga* `del sector` *nombre del sector* por cada plaga que vaya a fumigar. Debe dejarse un caracter blanco a ambos lados de los dos puntos.
- `plagas`, que debe escribir `Plagas del sector` *nombre del sector* `:` seguido de las plagas y su número de plantas afectadas separadas por un caracter blanco y en orden alfabético. Debe dejarse un caracter blanco a ambos lados de los dos puntos.

Cada caso termina con una línea con tres guiones (`---`).

Si alguna operación produce una excepción se mostrará el mensaje de la excepción como resultado de la operación.

Entrada de ejemplo

```
alta sector3 100
alta sector2 200
datos sector2 pulgon 100
datos sector2 cochinilla 20
datos sector3 cochinilla 50
fumigar
datos sector2 cochinilla 500
datos sector2 cochinilla 100
datos sector3 pulgon 50
fumigar
alta sector3 50
datos sector3 cochinilla 120
plagas sector3
plagas sector2
fumigar
FIN
```

Salida de ejemplo

```
Fumigar la plaga pulgon del sector sector2
Fumigar la plaga cochinilla del sector sector3
ERROR: Numero de plantas incorrecto
Fumigar la plaga cochinilla del sector sector2
Fumigar la plaga pulgon del sector sector3
Plagas del sector sector3 : cochinilla 120 pulgon 0
Plagas del sector sector2 : cochinilla 0 pulgon 0
Fumigar la plaga cochinilla del sector sector3
---
```

Autor: Isabel Pita