Árboles binarios de búsqueda aleatorizados X49250_es

El objetivo de esta práctica es escribir un programa que soporta las operaciones diversas sobre *conjuntos*, que deberán ser implementados utilizando árboles binarios de búsqueda aleatorizados (RBSTs). Dichas operaciones inclurán:

- Crear un conjunto vacío
- Añadir un elemento x a un conjunto
- Eliminar un elemento x de un conjunto
- Determinar si un elemento x dado pertenece o no a un conjunto
- Unir dos conjuntos
- Hallar el elemento i-ésimo, $1 \leq i \leq n$, para un conjunto de n elementos
- ...

Entrada

La entrada consiste en una secuencia de órdenes, una orden por línea. Cada orden consiste en una palabra reservada que denota la operación a realizar, el identificador del conjunto sobre el cual se aplica, y a continuación los demás parámetros de la operación. Para simplicar los elementos de los conjuntos son strings. La sintaxis es la siguiente:

- **init** *id_conjunto*: crea un conjunto vacío con el identificador dado; si ya se había creado un conjunto con dicho identificador, se vacía su contenido previo.
- **ins** *id_conjunto elem*: añade el string *elem* al conjunto *id_conjunto*; no hace nada si el elemento ya pertenecía al conjunto; da un error si el conjunto no había sido creado previamente.
- **del** *id_conjunto elem*: elimina el string *elem* del conjunto *id_conjunto*; no hace nada si el elemento no pertenece al conjunto y da un error si el conjunto no existe.
- **cont** *id_conjunto elem*: devuelve cierto si y sólo si el string *elem* pertence al conjunto *id_conjunto*; da un error si el conjunto no existe.
- **merge** *id_conjunto_1 id_conjunto_2*: se unen los dos conjuntos dados; el conjunto *id_conjunto_1* pasa a ser la unión de los dos conjuntos originales, y el conjunto *id_conjunto_2* se vacía; da un error si alguno de los dos conjuntos no existe.
- **card** *id_conjunto*: devuelve la cardinalidad del conjunto *id_conjunto*; da un error si el conjunto no existe.
- **nth** *id_conjunto i*: devuelve el i-ésimo elemento del conjunto *id_conjunto*, cuya cardinalidad es n; da un error si $i < 1$, $i > n$ o el conjunto no existe. conjunto *id_conjunto*.
- **leq** *id_conjunto elem*: devuelve el número de elementos menores o iguales que el string *elem* en orden lexicográfico del conjunto *id_conjunto*; da un error si el conjunto no existe.

- **gt** *id_conjunto elem*: devuelve el número de elementos mayores que el string *elem* en orden lexicográfico del conjunto *id_conjunto*; da un error si el conjunto no existe.
- **between** *id_conjunto elem_1 elem_2*: devuelve una lista ordenada crecientemente de los elementos del conjunto *id_conjunto* mayores o iguales *elem_1* y menores o iguales a *elem_2*; devuelve una lista vacía si *elem_1* $>$ *elem_2* y da un error si el conjunto no existe.
- **min** *id_conjunto*: devuelve el menor elemento en orden lexicográfico del conjunto; da un error si el conjunto no existe o está vacío.
- **max** *id_conjunto*: devuelve el mayor elemento en orden lexicográfico del conjunto; da un error si el conjunto no existe o está vacío.
- **all** *id_conjunto*: devuelve una lista ordenada crecientemente de todos los elementos del conjunto *id_conjunto*; da un error si el conjunto no existe.

Salida

Para cada operación se reproducirá la línea de la entrada correspondiente precedida por un *prompt* `>` ; en la línea siguiente se imprimirá:

- `OK`: en las operaciones que modifican el o los conjuntos (por ejemplo, **ins** o **merge**), y que se han ejecutado con éxito.
- `ERROR`: si la operación da error.
- El resultado (string, número, booleano, lista de strings) que corresponda en el caso de las operaciones consultoras, cuando se puedan ejecutar con éxito; si el resultado es una lista, se imprimerán los elementos encerrados por corchetes (`[]`) y separados por comas.

Observación

Vuestro programa deberá estar escrito en C++. Para la evaluación de la práctica se usarán juegos de pruebas públicos y privados, pero sólo es imprescindible que las prácticas pasen los juegos de pruebas públicos. Los resultados de los juegos de pruebas privados sólo se usarán para perfilar la nota de la práctica. Si sólo uno de los programas entregados pasa los juegos de pruebas entonces sólo se evaluará ese programa (cada programa puntúa entre 0 y 5). La calificación obtenida será la misma para los dos integrantes del equipo.
Cualquier estructura de datos, incluso un simple vector o lista podría servir para implementar la funcionalidad pedida; no obstante, sólo serán válidas las prácticas en las que utilizen los árboles binarios de búsqueda aleatorizados para implementar todas las operaciones.
Los algoritmos deberán estar debidamente comentados. Asimismo, en la entrega deben figurar claramente (en un bloque de comentarios al inicio, por ejemplo) los nombres de los dos miembros del equipo.
La fecha y hora límite de entrega se publicará en un aviso del Racó.

Información del problema

Autor : Conrado Martinez
Generación : 2013-06-13 13:22:08