EVALUATOR: plataforma de gestión de problemas y cursos de programación

25 de marzo de 2021

Contenidos:

1. Enunciado de la práctica

La plataforma EVALUATOR integra una colección de problemas de programación, un repositorio de sesiones compuestas por problemas de dicha colección, un conjunto de cursos formados por una o más sesiones, y un conjunto de usuarios registrados.

Cada sesión tiene un nombre que la identifica (string) y está formada por un subconjunto de problemas distintos de la colección estructurados según unas relaciones de prerequisito. En cada sesión existe un problema básico inicial que se ha de resolver antes que el resto de problemas de la misma sesión. Los problemas de una sesión pueden ser prerequisito directo de 0, 1 o 2 problemas de la misma sesión, y cada problema, excepto el básico inicial, tiene un único problema que es prerequsito directo de él.

Si en la plataforma hay N cursos, cada curso tiene un valor entero entre 1 y N que lo identifica. Un curso está formado por un subconjunto de sesiones distintas del repositorio, con la restricción de que la intersección entre cualquier par de sesiones del curso es vacía. Es decir, un problema de la colección sólo puede aparecer una vez (como máximo) en un mismo curso, integrado en una sesión determinada. Un problema puede formar parte de uno o más cursos o de ninguno. Lo mismo ocurre con las sesiones, pueden formar parte de uno o más cursos o de ninguno. E igualmente podemos decir que un problema puede formar parte de una o más sesiones (mientras no sean del mismo curso) o de ninguna.

Cada usuario tiene un nombre que lo identifica (string). Los usuarios pueden registrarse en la plataforma y darse de baja cuando quieran. Al darse de baja, toda la información asociada al usuario es borrada del sistema. Los usuarios pueden inscribirse en un curso (si se han registrado previamente en la plataforma) para tratar de resolver los problemas de las sesiones que lo forman, pero no pueden inscribirse en otro hasta que hayan resuelto todos los problemas del curso en el que se han inscrito previamente.

Cuando un usuario está inscrito en un curso, puede trabajar simultáneamente en más de una de sus sesiones, pero en cada una de ellas sólo podrá trabajar en los problemas cuyos prerequisitos ha completado con éxito. Los usuarios enviarán sus soluciones de problemas al sistema y éste les informará de si es correcta (semáforo verde) o no. Cuando un usuario ha enviado una solución juzgada como correcta, ya no podrá enviar más soluciones a ese mismo problema.

Cada problema tiene un nombre que lo identifica (string). Aunque un problema debería contener en realidad numerosa información asociada, tal como su enunciado, juegos de pruebas públicos y privados y solución de referencia (oculta), para esta práctica consideraremos que sólo es necesario gestionar el número total de envíos realizados a ese problema y cuántos de ellos han sido juzgados como correctos.

De cada usuario también mantendremos estadísticas del número de problemas que ha intentado resolver (al menos ha hecho un envío), cuántos de ellos ha resuelto satisfactoriamente y cuántos envíos en total ha realizado. También debemos saber si está inscrito en algun curso o no, si lo está en cuál, y mantener un registro de los problemas que ya ha resuelto (en ese curso o en anteriores) y de los problemas que puede intentar resolver en el curso actual (porque ya ha resuelto sus prerequisitos).

2. Funcionalidades

2.1. Decisiones sobre los datos

Los datos del programa se dividen en dos partes. Un primer grupo de problemas, sesiones, cursos y usuarios se usará para la inicialización de la plataforma. En este grupo, solo se manejarán datos sintácticamente correctos y no será necesario comprobar dicha corrección. Por ejemplo, si decimos que inicialmente el sistema contiene P problemas, el programa recibirá un entero P mayor que 0 y a continuación exactamente P strings, sin repeticiones, correspondientes a nombres de problemas. Los datos también cumplirán todas las restricciones descritas en la sección anterior, por ejemplo, un mismo problema solo podrá aparecer, como máximo, en una sesión de un mismo curso.

Una vez inicializado el sistema, el programa ofrecerá un repertorio de funcionalidades. Cada funcionalidad podrá requerir a su vez unos datos y producirá una salida que describimos con más precisión en el siguiente apartado.

2.2. Programa principal: estructura y comandos

El programa principal empezará leyendo un número inicial de problemas $P > 0$ y una secuencia de P strings que identificarán a dichos problemas. A continuación se leerá un número inicial de sesiones $Q > 0$ y una secuencia de Q sesiones. Para cada sesión, se entrará su nombre y una secuencia de identificadores de problemas, siguiendo un recorrido en preorden de su estructura de prerequisitos. Seguidamente, se leerá un número inicial de cursos $N > 0$ y una secuencia de N cursos, identificados por el orden en que se leen. Para cada curso c, entre 1 y N, se leerá primero el número de sesiones $S_c > 0$ del curso y luego una secuencia de S_c identificadores de sesiones, válidos y sin repeticiones. Para acabar las inicializaciones, se leerá un número M de usuarios registrados inicialmente seguido de sus M identificadores.

Hay que notar que las magnitudes P, Q, N y M no serán constantes. P, Q y N podrán aumentar al dar de alta posteriormente nuevos problemas, nuevas sesiones o nuevos cursos. Nunca se borrarán ni problemas ni sesiones ni cursos, y una vez creados, no se podrán modificar ni las sesiones ni los cursos. Por lo tanto, el número de sesiones S_c de un curso sí será constante, así como el número de problemas de una sesión. El número M de usuarios irá variando (aumentando o disminuyendo) a medida que se registren nuevos usuarios o se den de baja en la plataforma.

Terminadas las inicializaciones, se procesará una serie de comandos acabados en un comando `fin`. La estructura general del programa principal será la siguiente:

```
lectura de la coleccion inicial de problemas;
lectura del repositorio inicial de sesiones;
lectura del conjunto inicial de cursos;
lectura de los usuarios iniciales;
lee comando;
while (comando != "fin") {
    procesa comando;
    lee comando;
```

```
}
```

Los comandos aceptados se describen a continuación. Todo ellos se presentan en dos versiones, una con el nombre completo y otra con el nombre abreviado. La sintaxis exacta de la entrada y la salida de cada comando se podrá derivar del juego de pruebas público del ejercicio creado en el Jutge para cada entrega.

1. `nuevo_problema` p: añade un problema nuevo con identificador p. El comando admite la forma abreviada `np` p. Si ya existía un problema en la plataforma con el mismo identificador se imprime un mensaje de error. En caso contrario se imprime el número de problemas P en la colección después de añadirlo.

2. `nueva_sesion` s: añade una sesión nueva con identificador s. El comando admite la forma abreviada `ns` s. Primero se debe leer la estructura de problemas de la sesión, de la misma manera que se hizo en las sesiones iniciales (se garantiza que los problemas existen y no están repetidos). Si ya existía una sesión en la plataforma con el mismo identificador se imprime un mensaje de error. En caso contrario se imprime el número de sesiones Q en el repositorio después de añadirla.

3. `nuevo_curso`: añade un nuevo curso con identificador $N+1$ en caso de que cumpla la restricción de no repetición de problemas en el mismo curso. El comando admite la forma abreviada `nc`. Primero se lee un número de sesiones $S_{N+1} > 0$ y luego una secuencia de S_{N+1} identificadores válidos de sesiones. Si no hay intersección de problemas entre las sesiones, se añade el curso al conjunto de cursos y se imprime su identificador. En caso contrario, no se añade al conjunto (por lo que N no varía) y se imprime un mensaje de error.

4. `alta_usuario` u: registra un usuario nuevo con identificador u. El comando admite la forma abreviada `a` u. Si ya existía un usuario en la plataforma con el mismo identificador se imprime un mensaje de error. En caso contrario se imprime el número de usuarios registrados M después de añadirlo.

5. `baja_usuario` u: da de baja un usuario con identificador u. El comando admite la forma abreviada `b` u. Si no existe un usuario en la plataforma con identificador u se imprime un mensaje de error. En caso contrario se imprime el número de usuarios registrados M después de borrarlo. Si más tarde se da de alta otro usuario con el mismo nombre es como si el anterior no hubiera existido.

6. `inscribir_curso` u c: inscribe un usuario u en el curso con identificador c. El comando admite la forma abreviada `i` u c. Si el usuario u no existe en la plataforma o ya está inscrito en otro curso que no ha sido completado, o si el curso c no existe, se imprime un mensaje de error. En caso contrario se imprime el número de usuarios inscritos en el curso c después de añadirlo.

7. `curso_usuario` u: consulta el curso en el que está inscrito un usuario u. El comando admite la forma abreviada `cu` u. Si el usuario u no existe en la plataforma se imprime un mensaje de error. En caso contrario, se imprime el identificador del curso o un cero si no está inscrito en ningún curso.

8. `sesion_problema` c p: consulta la sesión del problema p en el curso con identificador c. El comando admite la forma abreviada `sp` c p. Si el curso c no existe o si el problema p no existe o si el problema p no pertenece al curso c, se imprime un mensaje de error. En caso contrario, se imprime el identificador de la sesión donde se realiza el problema p en el curso c.

9. `problemas_resueltos` u: se listan en orden creciente por identificador los problemas solucionados con éxito por el usuario u, ya sea en el curso en que está inscrito actualmente (si lo está en alguno) como los resueltos en cursos anteriores. Además de los identificadores se imprime también el número de envíos realizados por el usuario a cada problema del listado (valor mayor o igual que uno). Si el usuario u no existe en la plataforma, se imprime un mensaje de error. El comando admite la forma abreviada `pr` u.

10. `problemas_enviables` u: se listan en orden creciente por identificador los problemas que el usuario no ha solucionado todavía en el curso en el que está inscrito actualmente, pero a los cuales ya puede realizar un envío (porque cumple todos sus prerequisitos, directos e indirectos). Además de los identificadores se imprime también el número de envíos realizados por el usuario a cada problema del listado (valor mayor o igual que cero). Si el usuario u no existe en la plataforma o no está inscrito en ningún curso, se imprime un mensaje de error. El comando admite la forma abreviada `pe` u.

11. `envio` u p r: se toma nota del resultado r ($r = 1$ si éxito o $r = 0$ si fallo) de un nuevo envío del usuario u al problema p. Se garantiza que el usuario u está registrado e inscrito en un curso donde aparece el problema p y que p pertenece al conjunto de problemas a los que u puede enviar una solución por cumplir los prerequisitos. Todo envío da lugar a la actualización de las estadísticas del usuario y del problema. Además, en el caso $r = 1$ también hay que actualizar los problemas resueltos y enviables del usuario y comprobar si con ello ha completado el curso en el que estaba inscrito. En caso de que lo haya completado, el usuario deja de estar inscrito en el curso. El comando admite la forma abreviada `e` u p r.

12. `listar_problemas`: se listan los problemas de la colección, indicando para cada problema el número t de envíos totales y el número e de envíos con éxito a dicho problema, en ambos casos de usuarios presentes o pasados. También se ha de escribir el ratio $(t + 1)/(e + 1)$, y los problemas han de listarse en orden creciente por dicho ratio. En caso de empate, se han de listar por orden creciente de identificador. El comando admite la forma abreviada `lp`. Si se desea lo mismo para un solo problema se usa `escribir_problema` p. Si p no existe en la plataforma se imprime un mensaje de error. El comando admite la forma abreviada `ep` p.

13. `listar_sesiones`: se listan las sesiones actuales de la plataforma, ordenadas crecientemente por su identificador y mostrando, para cada sesión, el número de problemas que la forman y los identificadores de dichos problemas, siguiendo su estructura de prerequisitos en postorden. El comando admite la forma abreviada `ls`. Si se desea lo mismo para una sola sesión se usa `escribir_sesion` s. Si s

no existe en la plataforma se imprime un mensaje de error. El comando admite la forma abreviada `es` s.

14. `listar_cursos`: se listan los cursos actuales de la plataforma, ordenados crecientemente por su identificador y mostrando, para cada curso, el número de usuarios actuales o pasados que lo han completado, el número de usuarios inscritos actualmente, el número de sesiones que lo forman y los identificadores de dichas sesiones, en el mismo orden en el que se leyeron cuando se creó el curso. El comando admite la forma abreviada `lc`. Si se desea lo mismo para un solo curso se usa `escribir_curso` c. Si c no existe en la plataforma se imprime un mensaje de error. El comando admite la forma abreviada `ec` c.

15. `listar_usuarios`: se listan los usuarios registrados actualmente en la plataforma, ordenados crecientemente por su nombre y mostrando, para cada usuario, cuántos envíos en total ha realizado, cuántos problemas ha resuelto satisfactoriamente, cuántos problemas ha intentado resolver (al menos ha hecho un envío), y el identificador del curso en el que está inscrito o un cero si no está inscrito en ninguno. El comando admite la forma abreviada `lu`. Si se desea lo mismo para un solo usuario se usa `escribir_usuario` u. Si u no existe en la plataforma se imprime un mensaje de error. El comando admite la forma abreviada `eu` u.

16. `fin`: termina la ejecución del programa