Algoritmos y Estructuras de Datos.
TPLSR. Recuperatorio de Trabajo Práctico de Laboratorio. [2014-11-07]

PASSWD PARA EL ZIP: **FK5 FV4 AFF K53**

Ejercicios

[Ej. 1] **[isomorph]** Dos árboles binarios `B1`, `B2` son **isomorfos** si se pueden aplicar una serie de inversiones entre los hijos derechos e izquierdos de los nodos de `B2` de manera que quede un árbol semejante a `B1`, es decir que tiene la misma estructura. Por ejemplo `(1 . (2 3 4))` es isomorfo a `(1 (3 4 2) .)`. En particular si dos árboles son isomorfos la cantidad de nodos que tienen una profundidad y altura dadas es la misma. Por ejemplo en el caso anterior ambos árboles tienen 4 nodos, de los cuales dos a 2 a profundidad 2, 1 a profundidad 1 y por supuesto 1 a profundidad 0 (raíz).

Concretamente `B1` es isomorfo a `B2` si

- Ambos son Λ
- Los subárboles de los hijos izquierdos de `B1` y `B2` son isomorfos entre sí y los derechos también
  ó
  el subárbol del hijo izquierdo de `B1` es isomorfo al derecho de `B2` y el derecho de `B1` es isomorfo al izquierdo de `B2`.

Nota: Notar que sólo se mira la **estructura** del árbol, no los valores de los nodos.

Consigna: Escribir una función
`bool btisomorph(btree<int> &B1,btree<int> &B2);`

[Ej. 2] Dado un `set<int>` y un entero `M` determinar si existe un subconjunto de `S` que sume exactamenet `M`.

Consigna: Escribir la función
`bool has_sum(set<int> &S, int M);`

Ayuda:

- Si hay un elemento en `S` que es `M` ya está.
- Si todos los elementos de `S` son mayores que `M` ya está.
- Si no probar (recursivamente) **para cada uno** de los elementos `x` de `S` que son `x<M` para ver si `S-{x}` tiene un subconjunto `S'` con suma `M-x`. Si esto ocurre entonces `S'+{x}` tiene suma `M`.

[Ej. 3] Sean `L1`, `L2` de longitud `N` dos listas de enteros tales que `L2` es una permutación cíclica de `L1`, es decir `L2[k] = L1[(k+m)%N]` para un cierto entero `m<N`. Escribir una función
`int find_shift(list<int> &L1,list<int> &L2);` que retorna el shift `m`.

Instrucciones generales

- El examen consiste en que escriban las funciones descriptas más abajo; impleméntandolas en C++ de tal forma que el código que escriban **compile y corra correctamente**, es decir, no se aceptará un código que de algún error de compilación o que tire alguna excepción/señal de interrupción en runtime. Básicamente se hace una evaluación de caja negra, aunque le daremos un rápido vistazo al código.
- Pueden utilizar todas las funciones y utilidades del estándar de C++ que por supuesto contiene a la librería STL.
- Se incluye un template llamado `program.cpp`. En principio sólo tienen que escribir el cuerpo de las funciones pedidas. El paquete ya incluye el header `tree.h`.
- Para cada ejercicio hay dos funciones de evaluación, por ejemplo si `f` es la función a evaluar tenemos

```
ev.evalj(f,vrbs);
hj = ev.evaljr(f,seed); // para SEED=123 debe dar Hj=170
```

`j` es el número de ejercicio, por ejemplo para el ejercicio 1 tenemos las funciones (`eval1` y `eval1r`). La primera `ev.evalj(f,vrbs);` toma una serie de casos de prueba de entrada, le aplica la función del usuario `f` y compara la salida del usuario (`user`) con respecto a la esperada (`ref`). Si la verbosidad (el argumento `vrbs`) se pone en uno, entonces la función evaluadora reporta por consola los datos de entrada, la salida de la función de usuario y la salida esperada

```
m: 10, k: 3
T(ref): (10 (7 (4 1) 1) (4 1) 1)
T(user): (10 (7 (4 1) 1) (4 1) 1)
EJ1|Caso0. Estado: OK
```

- La segunda función `evaljr` es el chequeo que llamamos **SEED/HASH**. La clase evaluadora genera una serie de contenedores a partir de la semilla `seed`, se los pasa a la función del usuario `f()`. Las respuestas de la `f()` van siendo procesadas por la función interna de hash que genera un **checksum** `H` de las respuestas. Por ejemplo para el primer ejercicio si `seed=123` entonces el checksum es `H=523`. Una vez que el alumno termina su tarea se le pedirá que corra la clase evaluadora con un valor determinado de la semilla `seed` y se comprobará que genere el valor correcto del checksum `H`.
  Desde el punto de vista del alumno esto no trae ninguna complicación adicional, simplemente debe llenar el parámetro `seed` con el valor indicado por la cátedra, recompilar el programa y correrlo. La cátedra verificará el valor de salida de `H`.
- En la clase evaluadora cuentan con las siguientes funciones utilitarias:
  - `void dump(vector<set<int> > &VX,string s="")`: Imprime un mapa entero/entero. Nota: Es un método de la clase `Eval` es decir que hay que hacer `Eval ev; ev.dump(VX);`. El string `s` es un label opcional.
  - Análogamente está `void dump(set<int> S,string s="")`.
  - `btree<int>::lisp_print()`: Lisp print de un árbol. Nota: esta pertenece a la clase `tree`. Uso: `btree<int> T; T.lisp_print();`