Algoritmos y Estructuras de Datos.
TPL3. Trabajo Práctico de Laboratorio 3. [2014-11-01]

PASSWD PARA EL ZIP: `X35 926 RWF NG9`

Ejercicios

[Ej. 1] **[bijsubset]** Dado un conjunto `S` y una función de **mapeo** `T(*f)(T)` determinar el conjunto `S1` de elementos `x` de `S1` tales que no existe otro elemento `z` con `f(x)=f(z)`. Es decir, para los elementos de `S1` y su imagen, la relación es biyectiva.

Por ejemplo si `S={-1,1,-2,2,3,4,5}` y `f(x)=x*x` entonces debe ser `S1={3,4,5}` ya que los elementos `(-1,1)`, y `(-2,2)` los elementos de la imagen son los mismos.

Consigna: Escribir una función
`void bijsubset(set<int> &S, map_fun_t f, set<int> &S1);` que realiza la tarea indicada.
Donde
`typedef int(*map_fun_t)(int);`

Ayuda: Recorrer los elementos `x` de `S`, aplicarles la función `y=f(x)` y almacenarlos `y` en un conjunto `Y`. Al mismo tiempo si se detecta que el elemento `y` ya estaba en `Y` almacenarlo en otro conjunto `Ydup`. Por ejemplo en el caso anterior deberíamos terminar con `Y={1,4,16,25}` e `Ydup={1,4}`. Ahora volver a recorrer los elementos de `S` e insertarlos en `S1` sólo si su imagen no está en `Ydup`.

[Ej. 2] **[preimage]** Dados un `vector<set<int> > VX` y un `set<int> Y`, y una función de mapeo `T(*f)(T)` encontrar el conjunto de `VX[j]` **preimagen** de `Y` tal que si se le aplica `f` a sus elementos, se obtiene `Y`.

Por ejemplo si `VX=[{2,3,4},{-1,1,2},{5,6,7},{-1,-2}]` y `Y={1,4}`, y `f(x)=x*x` entonces debe retornar `true`, y `preY=VX[1]`.

Consigna: escribir la función
`bool preimage(vector< set<int> >&VX,map_fun_t f, set<int> &Y,set<int> &preY);`
Donde:
`typedef int(*map_fun_t)(int);`

Nota: Si hay varios `VX[j]` que satisfacen la condición entonces debe retornar el primero. En el ejemplo `{-1,-2}` tambien es preimagen de `{1,4}`, pero `{-1,1,2}` está primero.

Nota 2: Si no hay ninguno que sea la preimagen entonces debe retornar `false` y `preY={}`.

Ayuda: Escribir una función
`bool ismapped(set<int> &X,map_fun_t f,set<int> &Y)` que determina si `X` es la preimagen de `Y` por `f`. Luego recorrer los elementos de `VX` y aplicarles `ismapped()`.

[Ej. 3] **[is-recurrent-tree]** Dado un árbol binario lleno (todos sus nodos interiores tienen los dos hijos), verificar si es un "árbol recurrente". Un árbol binario se considera recurrente si cada uno de sus nodos interiores es la suma de sus 2 nodos hijos.

Por ejemplo

- `T = (5 (2 1 1) (3 1 (2 1 1))) -> true`
- `T = (5 (3 (2 1 1) 1) (3 1 (2 1 1))) -> false` (la raíz no es la suma de sus hijos)
- `T = (5 (2 1 1) (3 1 (1 1 1))) -> false` (el nodo con valor 3 no es la suma de sus hijos, al igual que uno de sus hijos tampoco es la suma de sus hojas).

Además lo generalizaremos en el estilo de **programación funcional**, reemplazando la suma por cualquier función binaria asociativa, de la siguiente forma:

Consigna: Escribir la función
`bool is_recurrent_tree(btree< int > &T,assoc_fun_t g);` donde:
`typedef int(*assoc_fun_t)(int, int);` que debe retornar `true` en caso que el árbol binario `T` sea recurrente (con respecto a `g(.,.)`), y false en caso contrario.

[Ej. 4] **[ordered-tree]** Dado un vector de números enteros positivos, se deberá armar un árbol binario de manera que la posición i-ésima del vector verifica si es mayor que la raíz. En caso que sea mayor, intentará insertarse en el subárbol derecho de la misma, y sino en el subárbol izquierdo. Si el hijo correspondiente está vacío entonces lo inserta allí, si no lo compara con el elemento, y vuelve a aplicar la regla recursivamente.

Consigna: Escribir la función
`void ordered_tree(vector<int> &V, btree< int > &T);` que debe retornar por referencia el árbol binario `T` que cumpla con el enunciado.

Ejemplos:

`V = [1 5 10 8 7 2] -> T = (1 . (5 2 (10 (8 7 .) .)))`

Se puede ver que el primer elemento del vector será la raíz del árbol. Luego el siguiente elemento, al ser mayor que la raíz se inserta a la derecha debido a que no existía este hijo derecho previamente. Después el siguiente elemento correspondiente al 10, es mayor que la raíz, pero como existe este hijo derecho, ahora comprueba si es mayor o menor que 5 y al ser mayor, inserta a la derecha nuevamente.

`V = [5 4 2 6 8 7 3] T = (5 (4 (2 . 3) .) (6 . (8 7 .)))`

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);
h1 = ev.evaljr(f,seed); // para SEED=123 debe dar H1=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();`**