

Algoritmos y Estructuras de Datos.

RECUP TPL3. Recuperatorio del Trabajo Práctico de Laboratorio 3. [2013-11-21]

PASSWD PARA EL ZIP: **XPXC LWIK 7MUL**

(Son 12 letras mayúsculas y números, sin espacios)

Instrucciones

- El examen consiste en que escriban las funciones descriptas más abajo; implementándolas 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 tres funciones pedidas. El paquete ya incluye el header **tree.h**.
- También se incluyen con el paquete la clase de árbol binario **btree.h**.
- Algunas funciones utilitarias que pueden servir

```
void print(set<int> &S);  
void print(list<set<int> > &LS);  
T.lisp_print();
```

- Las funciones **evaluar** ahora tienen un segundo argumento **int vrbs** (*verbosity*). Si **vrbs==1** entonces la clase evaluadora imprime los argumentos de entrada, valores retornados por la función del usuario (*user*) y esperados (*reference*). Por ejemplo en el caso de **sum_sim** si hacemos

```
ev.evaluar3(sum_sim, 1);
```

entonces la clase evaluadora imprime algo así como

```
A: (3 (9 5 (9 . 9)) 7)  
B: (3 7 (9 (9 9 .) 5))  
retval: user 0, ref 1
```

A, B son los árboles datos y el valor espera de retorno es 1, mientras que la función escrita por el usuario retorno 0 (incorrecto).

Ejercicios

[Ej. 1] **[subsup]** Dado una lista de conjuntos de enteros **LS**, y un conjunto **W** encontrar la lista **LSUB** de aquellos conjuntos S_j de **LS** tales que son subconjuntos de **W** ($S_j \subseteq W_j$). También debe devolver los conjuntos **LSUP** que son supraconjuntos de **W**, es decir los ($S_j \supseteq W_j$).

Ejemplo: Si **LS**=({1, 2}, {2}, {2, 3, 4}, {2, 4, 6}, {1, 3, 5}) y **W**={2, 4}, entonces debe retornar **LSUB**=({2}) y **LSUP**=({2, 3, 4}, {2, 4, 6}).

Consigna: Escribir una función

```
typedef list<set<int>> ls_t;  
void subsup(ls_t &LS, set<int> &W, ls_t &LSUB, ls_t &LSUP);
```

que realiza la tarea indicada. Los conjuntos en **LSUB** y **LSUP** deben quedar en el mismo orden que en el conjunto original.

Ayuda1: Recordar que para determinar si $A \subseteq B$ basta con hacer la diferencia $A - B = \emptyset$. Recomendamos escribir una función auxiliar `bool issubset(set<int> &A, set<int> &B)`.

Ayuda2: Recorrer los conjuntos en **LS** y verificar si incluyen a **W** o son incluidos por **W**. Dependiendo de esto incluirlo en **LSUB** o **LSUP**. **OJO** que puede ocurrir que un S_j esté en ambos (cuando ocurre?).

[Ej. 2] [inall] Dado una lista de conjuntos **LS** determinar si alguno de ellos está incluido en todos los otros, es decir si existe un S_j tal que $S_j \subset S_k$ para todo k .

Consigna: Escribir una función

```
typedef list<set<int>> ls_t;
bool inall(ls_t &LS, set<int> &S);
```

tal que

- Si alguno de los S_j satisface las condiciones entonces devuelve **true** y lo retorna por **S**.
- Caso contrario devuelve **false** (**S** queda indeterminado).

Ejemplos:

- **LS** = ({1, 2, 3}, {2, 3, 4}) retorna **false**
- **LS** = ({1, 2, 3}, {1, 2, 3, 4}, {1, 2, 3}) retorna **true** y **S** = {1, 2, 3}.
- **LS** = ({1, 2, 3}, {1}, {1, 2}) retorna **true** y **S** = {1}

Ayuda: Si existe el tal conjunto debe ser el de menor tamaño. Por lo tanto

- Buscar el **S** de menor tamaño.
- Verificar si **S** es subconjunto de todos los otros.

[Ej. 3] [sum-sim] Dados dos AB, **T1** y **T2**, escribir una función que verifique que la cantidad de nodos en el nivel 1 en **T1** y **T2** sea la misma, y que además la suma de los nodos de ese nivel coincida.

Consigna: Sea **AB_t** el tipo de árbol binario

```
typedef btree<int> AB_t;
```

Escribir una función

```
bool sum_sim(AB_t & T1, AB_t & T2, int l);
```

que realiza la tarea indicada.

Ayuda: Notar que la cantidad de nodos en el nivel **L** se puede escribir recursivamente como

$$\text{nodecount}(n, L) = \begin{cases} 0; & \text{si } n = \Lambda, \\ 1 + \text{nodecount}(l, L - 1) + \text{nodecount}(r, L - 1); & \text{caso contrario,} \end{cases} \quad (1)$$

Para la suma de los valores de los nodos tenemos algo similar

$$\text{nodesum}(n, L) = \begin{cases} 0; & \text{si } n = \Lambda, \\ *n + \text{nodesum}(l, L - 1) + \text{nodesum}(r, L - 1); & \text{caso contrario,} \end{cases} \quad (2)$$

Se sugiere encarar algunas de las siguientes estrategias,

- Escribir las funciones auxiliares **nodesum** y **nodecount** y aplicarlas a la raíz de **T1** y **T2** y chequear los valores retornados.
- Escribir una función que calcule las dos cosas al mismo tiempo, retornándolas por un **pair<int, int>**.
- Escribir una función que calcule las dos cosas al mismo tiempo retornándolas por referencia.