Algoritmos y Estructuras de Datos.
TPL2. 1er Trábajo Práctico de Laboratorio. [2013-10-05]

PASSWD PARA EL ZIP: `6DG1ZG7IH6`

Instrucciones

- 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 tres funciones pedidas. El paquete ya incluye el header **`tree.h`**.
- También se incluyen con el paquete los utilitarios **`tree.h`** y **`util_tree.h`**. Estos contienen diferentes funciones utilitarias que pueden servir para debuggear el programa.

Ejercicios

[Ej. 1] **[count-level]** Escribir una función **`void count_level(tree<int> &T, int l)`**, que cuenta cuantos nodos hay en el nivel **`l`** del árbol **`T`**.
Ayuda: La definición recursiva es:

$$\texttt{count\_level}(n,l) = \begin{cases} 0; & \text{si } n = \mathbf{\Lambda} \text{ o } l < 0, \\ 1; & \text{si } l = 0, \\ \sum_{c=\text{hijo den}} \texttt{count\_level}(c, l-1). & \end{cases} \quad (1)$$

Por ejemplo, para el árbol **`T=(a (b c d e) (f (g h i)))`** debe dar

```
count_level(T,1) -> 2
count_level(T,2) -> 4
```

[Ej. 2] **[is-shift]** Dados dos grafos **`G1,G2`** (como **`typedef map<int,list<int>> graph_t;`**), escribir una función **`bool is_shift(graph_t &G1,graph_t &G2,int m);`** que determina si **`G2`** es un 'shift' del **`G1`**, es decir la arista **`(x,y)`** está en **`G1`** si y solo si **`(x+m,y+m)`** está en **`G2`**.
Por ejemplo, si

```
G1 = [1->{2,4}, 2->{1,3,4}, 3->{2,4}, 4->{1,2,3}]
G2 = [4->{5,7}, 5->{4,6,7}, 6->{5,7}, 7->{4,5,6}]
```

entonces **`is_shift(G1,G2,3) -> true`**.

[Ej. 3] **[odd2even]** Dada una lista **`L`**, escribir una función
`void odd2even(list<int> &L,map<int,list<int>> &M);` que mapea cada elemento impar de **`L`** a la siguiente subsecuencia de elementos pares. Por ejemplo si **`L=(1,2,4,3,2,5,2,7,1,6,9,2,14)`** entonces debe dejar **`M={1->(2,4,6),3->{2},5->{2},7->{},9->(2,14)}`**. Si un numero impar aparece seguido de otro impar entonces el valor correspondiente debe ser la lista vacia (por ejemplo el 7 arriba). Si un numero impar aparece mas de una vez, entonces el valor correspondiente debe ser la concatenacion de todas las subsecuencias correspondiente (por ejemplo el 1 arriba).