

Algoritmos y Estructuras de Datos.

RECUP-TPL2. Recuperatorio Trabajo Práctico de Laboratorio 2. [2013-10-22]

PASSWD PARA EL ZIP: **OZU5DAUTYFIB**

Instrucciones

- El examen consiste en que escriban las funciones descritas 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 los utilitarios **tree.h** y **util_tree.h**. Estos contienen diferentes funciones utilitarias que pueden servir para debuggear el programa.
- Algunas funciones utilitarias que pueden servir

```
void printmap(map<int,int> &M);  
void printmap(map<int, list<int> >& M);  
void printl(list<int> &L);  
T.lisp_print();
```

Ejercicios

- [Ej. 1] **[evpath]** Escribir una función `int evpath(tree<int> &T)`, que devuelve la longitud del máximo camino donde todos los elementos de los nodos son pares. Por ejemplo, para el árbol $T = (0 \ (1 \ 2 \ 3) \ (2 \ (4 \ 5 \ 7) \ 3))$ debe dar 2, ya que el máximo camino de pares es $(0 \ 2 \ 4)$ de longitud 2.
Ayuda: La definición recursiva es:

$$\text{evpath}(T, n) = \begin{cases} -1; & \text{si } n = \Lambda, \\ 0; & \text{si } n \text{ es impar,} \\ 1 + \max_{c=\text{hijo de } n} \text{evpath}(T, c); & \text{caso contrario.} \end{cases}$$

- [Ej. 2] **[checksz]** Dados dos grafos $G1, G2$ (como `typedef map<int, list<int>> graph_t`), escribir una función `bool checksz(graph_t &G1, graph_t &G2)`; que determina si un dado nodo tiene la misma cantidad de vecinos en $G1$ y en $G2$. Es decir las claves de $G1$ y $G2$ deben ser las mismas y dada una clave k , las imágenes $G1[k]$ y $G2[k]$ deben tener el mismo tamaño.
Ejemplo 1:

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

Ejemplo 2:

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

ya que los tamaños de **G1 [2]**, **G2 [2]** son diferentes y además la clave 3 aparece sólo en **G2**.

Ayuda: Chequear que los tamaños de ambos grafos (o sea la cantidad de claves) sea la misma. Recorrer las claves de **G1** y verificar que la clave en **G2** exista y los tamaños de las imágenes sean iguales.

Alternativa: Para cada uno de los grafos construir un mapa **map<int, int> G1S, G2S** que mapean las claves de **G1, G2** en los tamaños de las listas. Luego verificar que **G1S, G2S** sean iguales.

[Ej. 3] [maxodd] Dada una lista **L**, escribir una función

void maxodd(list<int> &L, map<int, int> &M); que mapea cada elemento par de **L** al máximo de la siguiente subsecuencia de elementos impares. Por ejemplo si **L = (1, 2, 4, 3, 2, 5, 8, 2, 7, 1, 6, 9, 2, 14)** entonces debe dejar **M = {2 → 7, 4 → 3, 6 → 9}**.

- Si un número par aparece seguido de otro par entonces la subsecuencia de impares está vacía y en el mapa final no debe asignarle ningún valor. (Como en el ejemplo de arriba donde el 8 no recibe ningún valor.)
- Si un número par aparece mas de una vez, entonces el valor correspondiente debe ser el máximo de todas las subsecuencias correspondientes.
- Ignorar elementos impares que estén al principio de la lista (en el ejemplo de arriba el 1).