Algoritmos y Estructuras de Datos.
TPL2. Trabajo Práctico de Laboratorio 2. [2014-10-04]

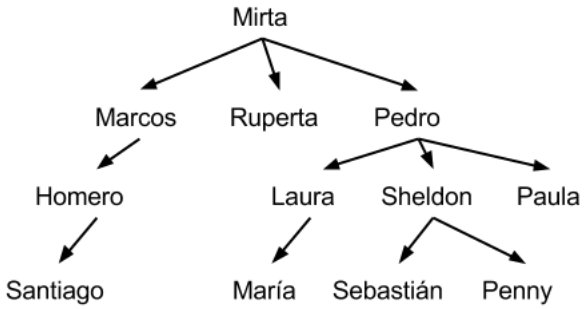
Ejercicios

[Ej. 1] **[common-relative]:** Durante la juntada que hicieron Santi y Seba para comer el asado, comenzaron a preguntarse sobre sus familias y de quienes eran descendientes. Fue por esto que se les ocurrió hacer a cada uno su árbol genealógico. Para ello buscaron en Internet a sus parientes e hicieron numerosas llamadas a familiares y a cada persona que lleva su apellido en la guía telefónica (más de uno le cortó la llamada porque pensaron que era una encuesta de una importante empresa nacional de telefonía). Una vez que reunieron suficiente información y vieron el árbol genealógico que formaron, quedaron estupefactos al ver cuán grande les quedó.

Cansados por toda la tarea de inteligencia, nuevamente recurren a usted para que implemente una función

```
string common_relative( tree <string> &T, string name1, string name2);
```

que dados un árbol **T** y dos personas que pertenecen al mismo (**name1** y **name2**), retorne el ancestro más próximo que tengan en común.



En este ejemplo, si preguntamos cual representa el pariente en común más cercano entre María y Sheldon, éste será Pedrito. Por otro lado, si nos hacemos esta misma pregunta pero con Santiago y Laura, su respuesta deberá ser Mirta.

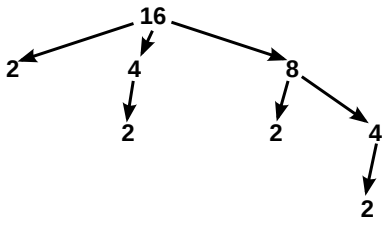
Si la pregunta es entre Santiago y Homero, la respuesta debe ser Homero.

Ayuda: Para cada uno de los nombres **name1** y **name2** encontrar el camino que va desde la raíz al nombre, y luego recorrer los caminos hasta el primer nodo cuyo valor es diferente. Por ejemplo en el caso anterior el camino que va a María es **L1=(Mirta,Pedro,Laura,María)** mientras que el que va a Sheldon es **L1=(Mirta,Pedro,Sheldon)** por lo tanto las recorriendo las listas vemos que el único camino en comun es Mirta. Para el par Santiagp, Homero en cambio los caminos son **L1=(Mirta,Marcos,Homero,Santiago)** y **L1=(Mirta,Marcos,Homero)**, por lo tanto el último antecesor común es Homero.

[Ej. 2] **[div-tree]** Dado un número natural **n**, se busca generar un árbol con todos sus divisores como hijos (sin incluir los divisores triviales 1 y **n**). A su vez, cada uno de sus hijos deberán tener como descendientes a sus correspondientes divisores. Y así sucesivamente hasta que éstos resulten ser números primos. Los números que sean primos, representarán hojas en el árbol. (Recordar que los números primos sólo tienen como divisores al número 1 y a sí mismos.)

La función a implementar será void **div_tree(tree<int> &T, int N);**

Por ejemplo, si a la función se le pasa como argumento al número 16, deberá retornar en **T** el siguiente árbol:



Restricciones: Los hijos deberán encontrarse en orden creciente.

Ayuda: Para obtener los divisores de un número **N**, basta con utilizar un contador **I** que recorra desde el número 2 hasta **N/2** inclusive. Si se verifica que (**N% I == 0**), entonces **I** es divisor de **N**.

[Ej. 3] [bordering-countries]:

Un licenciado en geografía tiene dos estructuras de tipo correspondencias. Una de ellas posee como claves a diferentes países, y como valor, las ciudades que tienen cada uno de ellos.

La otra estructura tiene como ciudades como claves, y como valor a las ciudades vecinas.

A este entusiasta licenciado se le ocurrió preguntarse, para un determinado país, qué países limítrofes tiene, en base a las ciudades que tiene conectadas.

El licenciado nos dió un ejemplo de posibles correspondencias:

```
countries["Francia"] = {"Calais", "París", "Perpignan"};
countries["Gran Bretaña"] = {"Dover", "Londres"};
countries["España"] = { "Barcelona", "Madrid"};

neighbours["Paris"]={"Lyon", "Calais"}
neighbours["Calais"]={"Lyon", "Ginebra","París"}
neighbours["Perpignan"]={"París", "Barcelona"}
neighbours["Dover"]={"Londres"}
neighbours["Londres"]={"Dover"}
neighbours["Barcelona"]={"Madrid", "Perpignan"}
neighbours["Madrid"]={"Barcelona"}
```

Entonces para el ejemplo, si queremos saber ¿qué países limitan con Francia?, la respuesta será **{"España"}**.

Su trabajo consiste en definir una función

```
list<string> bordering_countries(map<string,list<string> >
   &countries, map<string,list<string> > &neighbours, string   country);
```

que retorne una lista con los países limítrofes al país **country**. La lista deberá estar ordenada alfabéticamente y no deberá tener elementos repetidos.

[Ej. 4] [is-palindrome]: Dada una pila **S** determinar si es un **palíndromo**, es decir si al invertir los elementos queda igual.

```
bool is_palindrome(stack<int> &S);
```

Por ejemplo, para **P1=(0 1 2 3 2 1 0)** debe retornar **true** y para **P2=(0 1 2 3 4 3)** debe retornar **false**.

Restricciones: solo se pueden usar pilas auxiliares y no se debe usar espacio adicional, es decir la suma de todas las longitudes de las pilas debe ser $n + O(1)$. Puede ser destructiva.

Ayuda: Pasar los elementos de **S** a una pila auxiliar **S2** y contarlos. Sabiendo el numero de elementos **n**, pasar **[n/2]** elementos de **S2** a **S**. Si **n** es impar sacar un elemento de **S2**. Ahora **S** y **S2** deben ser iguales. Ahora se debe verificar si **S** es igual a **S2**.

En el ejemplo anterior **P1** debe quedar **S=S2=(2 1 0)**, mientras que para **P2** debe quedar **S=(3 4 3)**, y **S2=(2 1 0)**.

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 funciones pedidas. El paquete ya incluye el header **`tree.h`**.

mstorti@galileo/aed-2.0.5-79-gd0712f5/Thu Oct 2 22:39:25 2014 -0300