Algoritmos y Estructuras de Datos.
TPL2. Trabajo Práctico de Laboratorio. [2016-10-13]

PASSWD PARA EL ZIP: `LYW2 P51S EWPB`

Ejercicios

ATENCION: Deben necesariamente usar la opción `-std=gnu++11` al compilador, **si no no va a compilar**.

[Ej. 1] **[unordered-equal (33pt)]** Escribir una función bool que reciba dos Arboles Ordenados Orientados (AOO) y retorne **`true`** si son "desordenadmente" iguales. Se dice que dos árboles son "desordenadamente" iguales si:

(a) las raíces de ambos son iguales,
(b) el conjunto de hijos de la raíz de uno es igual al conjunto de hijos de la raíz del otro, y
(c) la condición se cumple recursivamente para cada par de subarboles de cada par de nodos equivalentes (uno de cada conjunto).

Notar que en un "conjunto" no importa el orden, entonces la condición (b) implica que las raíces tienen los mismo hijos, aunque podrían estar en diferente orden. Por ejemplo, los árboles **`(1 (2 5 6) 3 (4 7 8))`** y **`(1 (4 7 8) 3 (2 6 5))`** son desordenadamente iguales, ya que en ambos árboles los hijos de 1 son 2, 3 y 4 (aunque en diferente orden), los hijos de 2 son 5 y 6 (en dif orden), los hijos de 4 son 7 y 8, y el resto no tiene hijos.
NOTA: Como prerequisito se asume que todos los hermanos son diferentes entre sí. Es decir el árbol **`(1 2 3 2)`** no satisface el prerequisito ya que hay dos elementos **`2`** entre los hijos de **`1`**.
Algoritmo sugerido: sean **`A`** y **`B`** dos árboles de enteros (**`tree<int>`**), dados dos iteradores **`itA`** e **`itB`** apuntando a las raíces de **`A`** y **`B`** respectivamente:

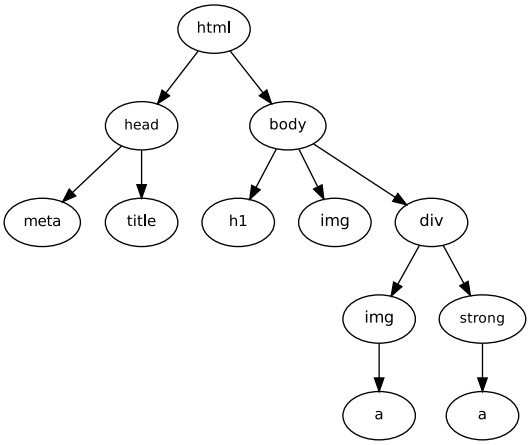
- Colocar todos los hijos del nodo **`itA`** en un mapa **`MA`**, donde las claves sean la etiquetas de dichos hijos, y los valores sean iteradores apuntando a los mismos.
- Colocar todos los hijos del nodo **`itB`** en un mapa **`MB`**, donde las claves sean la etiquetas de dichos hijos, y los valores sean iteradores apuntando a los mismos.
- Si los mapas **`MA`** y **`MB`** tienen diferente tamaño retornar **`false`**.
- Recorrer en simultáneo **`MA`** y **`MB`**, y por cada par de pares clave-valor (uno de **`MA`** y otro de **`MB`**):
  - Si las etiquetas (claves) son diferentes retornar **`false`**
  - Aplicar el algoritmo recursivamente utilizando los dos iteradores (valores), y si retorna falso, retornar falso.
- Retornar verdadero

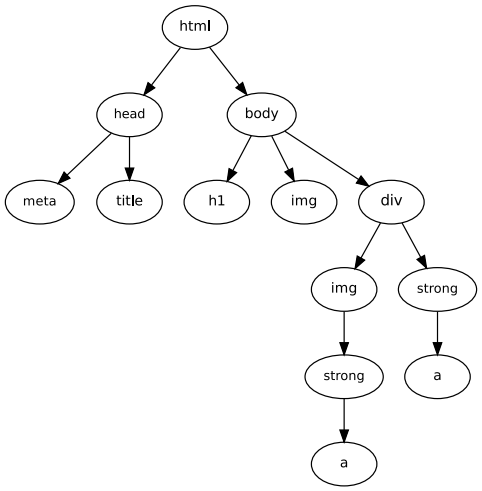
[Ej. 2] **[hay-camino (33pt)]** Un viajante quiere viajar desde una ciudad a otra siguiendo algún camino del grafo conexo de rutas **`M`**. Lamentablemente se tiene la información de que en algunas ciudades hay piquetes y es imposible pasar por ellas. Para determinar si es posible realizar el viaje se debe implementar una función,
`bool hay_camino(map<string,list<string>>&M, list<string>&P, string cini, string cend);`
que recibe en **`M`** el mapa de rutas disponibles (cada arista del grafo representa una ruta directa disponible entre las ciudades de los vértices que conecta), y en **`P`** la lista de ciudades con piquetes. La función debe retornar verdadero si existe alguna ruta que comience en la ciudad **`cini`** y finalice en **`cend`** sin pasar por ninguna de las ciudades con piquetes.

[Ej. 3] **[enhance-html (33pt)]** Los desarrolladores de un sitio web desean resaltar los links que aparecen dentro de cada página del sitio. Para ello es necesario que cada link (tag **<a>** en HTML) se encuentre dentro de un tag ****. Para resolver este problema ya contamos con un parser del código HTML que lo representa en un **tree<string>**. Por ejemplo, para el codigo HTML:

```
<html>
<head>
  <meta charset="UTF-8">
  <title>Ejemplo</title>
</head>
<body>
  <h1> Página de Ejemplo </h1>
  <img src="ejemplo.jpeg">
  <div>
    <a href="www.google.com.ar">
    <strong><a href="www.facebook.com.ar"></strong>
  <div>
</body>
</html>
```

el AOO (**tree<string>**) asociado es
T=(html (head meta title) (body h1 img (div (a img a) (strong a)))).





Se pide entonces implementar la función
void enhance_HTML(tree<string>&T); que recibe un AOO **T** con el parsing de alguna página web y lo manipula para obtener la estructura con el diseño modificado, es decir, todo tag **a** debe ser hijo de un tag **strong**. Notar que un link ya puede estar resaltado y que los links no pueden contener otros tags dentro suyo.

Instrucciones generales

- El examen consiste en que escriban las funciones descriptas más arriba; 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.
- Salvo indicación contraria 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.
- Para cada ejercicio hay dos funciones de evaluación, por ejemplo si **f** es la función a evaluar tenemos

  ```
  ev.eval<j>(f,vrbs);
  hj = ev.evalr<j>(f,seed); // para SEED=123 debe dar Hj=170
  ```

  j es el número de ejercicio, por ejemplo para el ejercicio 1 tenemos las funciones (**eval<1>** y **evalr<1>**). La primera **ev.eval<j>(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
  ```

- **ucase**: Además las funciones **eval()** tienen dos parámetros adicionales:
  Eval::eval(func_t func,int vrbs,int ucase);
  El tercer argumento 'ucase' (caso pedido por el usuario), permite que el usuario seleccione uno solo de todos los ejercicios para chequear. Por defecto está en **ucase=-1** que quiere "hacer todos". Por ejemplo **ev.eval4(prune_to_level,1,51);** corre sólo el caso 51.
- **Archivo con casos tests JSON:** Los casos test que corre la función **eval<j>** están almacenados en un archivo **test1.json** o similar. Es un archivo con un formato bastante legible. Abajo hay un ejemplo. **datain** son los datos pasados a la función y **output** la salida producida por la función de usuario. **ucase** es el número de caso.

  ```
  { "datain": {
    "T1": "( 0 (1 2) (3 4 5 6) )",
    "T2": "( 0 (2 4) (6 8 10 12) )",
    "func": "doble" },
   "output": { "retval": true },
   "ucase": 0 },
  ```

- La segunda función **evalr<j>** 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 función **evalr<j>()** de 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 funciones utilitarias como por ejemplo:
  void Eval::dump(list <int> &L,string s=""): Imprime una lista de enteros por **stdout**. Nota: Es un método de la clase **Eval** es decir que hay que hacer **Eval::dump(VX);**. El string **s** es un label opcional.
  - **void Eval::dump(list <int> &L,string s="")**
- Después del parcial deben entregar el programa fuente (sólo el **program.cpp**) renombrado con su apellido y nombre (por ejemplo **messilionel.cpp**). Primero el apellido.

mstorti@galileo/aed-3.0-45-g19b81ae/Thu Oct 13 08:38:45 2016 -0300

TPL2. Trabajo Práctico de Laboratorio. [2016-10-13]. TABLA SEED/HASH

```
S=123 -> H1=457 H2=026 H3=261
S=577 -> H1=835 H2=342 H3=197
S=393 -> H1=622 H2=562 H3=619
S=686 -> H1=920 H2=965 H3=857
S=349 -> H1=633 H2=842 H3=204
S=762 -> H1=099 H2=959 H3=686
S=690 -> H1=663 H2=718 H3=729
S=663 -> H1=313 H2=577 H3=238
S=340 -> H1=857 H2=027 H3=103
S=872 -> H1=106 H2=597 H3=206
S=711 -> H1=801 H2=540 H3=307
S=367 -> H1=563 H2=226 H3=454
S=882 -> H1=895 H2=553 H3=151
S=162 -> H1=680 H2=410 H3=297
S=767 -> H1=227 H2=693 H3=931
S=429 -> H1=937 H2=308 H3=953
S=622 -> H1=123 H2=822 H3=341
S=969 -> H1=405 H2=183 H3=330
S=893 -> H1=622 H2=256 H3=917
S=311 -> H1=440 H2=932 H3=079
S=529 -> H1=607 H2=428 H3=712
S=721 -> H1=765 H2=336 H3=669
S=384 -> H1=012 H2=284 H3=811
S=798 -> H1=225 H2=492 H3=553
S=615 -> H1=661 H2=708 H3=520
```

```
S=386 -> H1=890 H2=687 H3=826
S=215 -> H1=288 H2=391 H3=933
S=935 -> H1=348 H2=194 H3=993
S=292 -> H1=463 H2=730 H3=436
S=821 -> H1=070 H2=662 H3=761
S=527 -> H1=374 H2=545 H3=866
S=359 -> H1=730 H2=175 H3=355
S=626 -> H1=259 H2=652 H3=150
S=226 -> H1=378 H2=052 H3=482
S=236 -> H1=939 H2=371 H3=944
S=468 -> H1=162 H2=188 H3=518
S=529 -> H1=607 H2=428 H3=712
S=630 -> H1=228 H2=989 H3=545
S=923 -> H1=973 H2=697 H3=430
S=335 -> H1=526 H2=427 H3=397
S=802 -> H1=768 H2=227 H3=594
S=958 -> H1=923 H2=868 H3=865
S=967 -> H1=285 H2=409 H3=215
S=656 -> H1=424 H2=547 H3=128
S=242 -> H1=383 H2=008 H3=390
S=973 -> H1=373 H2=027 H3=567
S=219 -> H1=123 H2=190 H3=517
S=437 -> H1=098 H2=955 H3=885
S=624 -> H1=324 H2=985 H3=789
S=670 -> H1=865 H2=177 H3=038
```