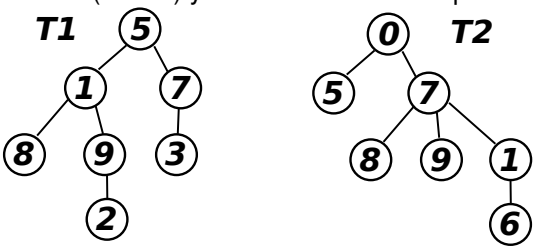
Algoritmos y Estructuras de Datos.
TPL2. Trabajo Práctico de Laboratorio 2. [2018-10-11]

PASSWD PARA EL ZIP: `5F4V GW7C 3P7H`

Ejercicios

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

[Ej. 1] **[classif]** Implemente una función
`void classify_relative(tree<int> &T,int n1,int n2,int &m1, int&m2);` que dados dos valores nodales `n1` y `n2` en un árbol `T` retorna las distancias `m1` y `m2` de ambos nodos al antecesor común más cercano. Por ejemplo si `T1=(5 (1 8 (9 2)) (7 3))`, `n1=8` y `n2=7` entonces el antecesor común es el nodo `5` (la raíz) y las distancias correspondientes son `m1=2` y `m2=1`.



Más ejemplos:

```
T1=(5 (1 8 (9 2)) (7 3)),    n1=1,n2=1   =>    m1=0,m2=0,
T1=(5 (1 8 (9 2)) (7 3)),    n1=9,n2=5   =>    m1=2,m2=0,
T2=(0 5 (7  8 9 (1 6))),     n1=8,n2=6   =>    m1=1,m2=2,
T2=(0 5 (7  8 9 (1 6))),     n1=6,n2=7   =>    m1=2,m2=0,
```

Nota: Con esta función es muy fácil de determinar el grado de parentezco de los nodos,

- Si `m1=m2=0` los nodos son iguales (o sea `n1=n2`)
- Si `m1=0` y `m2>0`: `n2` es antecesor de `n1` y viceversa
- Si `m1=m2=1` son hermanos
- Si `m1=m2=2` son *primos*
- Si `m1=1 m2=2` entonces `n1` es *tío* de `n2`

Nota: Los árboles son generados tal que los valores almacenados en los nodos son únicos. Por lo tanto dado un valor nodal que se pasa a la función (`n1` y `n2`), está garantizado que están en el árbol y son únicos.

Ayuda:

- Escribir una función
  `void get_path(tree<int> &T,int n,list<int> &L);` que retorna en `L` la lista de valores nodales enel camino que va desde la raíz hasta el nodo que contiene al valor `n`. Por ejemplo si `T1=(5 (1 8 (9 2)) (7 3))`, y `n=8` entonces debe retornar `L=(5,1,8)`.
- Para ello escribir la función recursiva
  `void get_path(tree<int> &T,tree<int>::iterator q,int n,list<int> &L);` que busca el camino pero a partir del nodo `q`. Para ello
  - Si el nodo `q`=Λ retorna la lista vacía.
  - Si el nodo contiene el valor buscado retorna una lista con sólo ese elemento.

- Si ninguna de esas opciones es verdadera recorre los hijos de **q** aplicando recursivamente la función. Si alguno de ellos retorna una lista no vacía **L** entonces prependiza el valor del nodo a esa lista.

- Una vez elaborada la función **get_path()** utilizarla para obtener los caminos **L1** y **L2** a los nodos que contienen los valores **n1** y **n2**. Por ejemplo si **T=(5 (1 8 (9 2)) (7 3))**, **n1=8** y **n2=7** entonces debemos tener **L1=(5,1,8)** y **L2=(5,7)**.
- Estas listas deben empezar en el valor de la raíz y terminar en los valores buscados **n1** y **n2**.
- Recorrer estas listas (que son los caminos) a partir de la raíz hasta el antecesor común, es decir el último valor nodal que es igual, en este caso el 5.
- A partir de allí sólo basta contar los nodos desde ese antecesor común hasta el fin de cada una de las listas. En el ejemplo **m1=2** para ir desde el 5 hasta el 8 y **m2=1** para ir desde el 5 hasta el 7.
- Sugerimos concentrarse primero en escribir correctamente la función **get_path()** y después pasar a **classify_relative()**.

[Ej. 2] **[prom-path]** Dado un árbol **T**, escribir una función **float prom_path(tree<int> &T);** que retorne la longitud promedio de los caminos desde la raíz a las hojas. Por ejemplo, para el árbol **T=(1 (2 3 4 (5 6)) 7 8)**, los 5 posibles caminos desde la raiz a una hoja son: **{1,2,3}, {1,2,4}, {1,2,5,6}, {1,7}, {1,8}**; cuyas longitudes son 2, 2, 3, 1 y 1; lo cual da un promedio de (2+2+3+1+1)/5 = 9/5 = 1.8.

[Ej. 3] **[filtra-deps]** Se tiene un **map<string,list<string>>** que representa un grafo dirigido donde los nodos son nombres paquetes de software en un repositorio, y los arcos dependencias entre paquetes.
Por ejemplo, en:

```
zinjai->{gdb,gcc,gtk,xterm}
pseint->{gtk,glut}
gtk->{Xorg}
glut->{mesa}
mesa->{Xorg}
xterm->{Xorg}
Xorg->{}
gcc->{}
gdb->{}
```

pseint->{gtk,glut} indica que para usar el paquete **pseint** es necesario instalar antes los paquetes **gtk** y **glut**; pero a su vez, para instalar estos paquetes, antes se deben instalar **mesa** y **Xorg** por sus respectivas dependencias.
Debe implementar una función
void filtra_deps(map<string,list<string>> &G, list<string> &L); que reciba el mapa del repositorio completo y una **list<string>** con nombres de paquetes; y recorte el mapa de forma que solo incluya a los paquetes a instalar y sus dependencias (directas o indirectas).
Por ejemplo, si la lista **L** es **{xterm,pseint}**, debe retornar el grafo:

```
pseint->{xterm,gtk,glut}
gtk->{Xorg}
glut->{mesa,Xorg}
mesa->{Xorg}
xterm->{Xorg}
Xorg->{}
```

Ayuda: Se sugiere utilizar el siguiente algoritmo:

- Por **cada paquete** `P` de la lista:
  a) Marcar el paquete `P`, y recursivamente todas sus dependencias, como "necesario". ("Necesarios"será un conjunto de paquetes a implementar de forma similar al conjunto de "visitados.[en] otros ejercicios.)
- Por **cada nodo** `N` del grafo:
  a) Si no es "necesario", borrar el nodo `N` del grafo.
  b) Si es “necesario”, por cada dependencia `D`:
    1) Si no es "necesaria", borrar `D` de la lista de dependencias del nodo `N`.

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.

TPL2. Trabajo Práctico de Laboratorio 2. [2018-10-11]. TABLA SEED/HASH

```
S=123 -> H1=323 H2=517 H3=090
S=577 -> H1=266 H2=672 H3=435
S=393 -> H1=888 H2=905 H3=840
S=686 -> H1=387 H2=046 H3=588
S=349 -> H1=680 H2=327 H3=451
S=762 -> H1=612 H2=218 H3=777
S=690 -> H1=785 H2=722 H3=807
S=663 -> H1=443 H2=705 H3=693
S=340 -> H1=835 H2=464 H3=817
S=872 -> H1=760 H2=279 H3=613
S=711 -> H1=793 H2=867 H3=294
S=367 -> H1=246 H2=870 H3=345
S=882 -> H1=311 H2=021 H3=077
S=162 -> H1=187 H2=196 H3=279
S=767 -> H1=678 H2=213 H3=023
S=429 -> H1=548 H2=545 H3=489
S=622 -> H1=395 H2=076 H3=382
S=969 -> H1=672 H2=651 H3=940
S=893 -> H1=541 H2=213 H3=118
S=311 -> H1=578 H2=792 H3=066
S=529 -> H1=275 H2=597 H3=157
S=721 -> H1=005 H2=607 H3=259
S=384 -> H1=942 H2=732 H3=681
S=798 -> H1=523 H2=089 H3=821
S=615 -> H1=605 H2=718 H3=314
S=386 -> H1=138 H2=650 H3=928
S=215 -> H1=938 H2=113 H3=343
S=935 -> H1=103 H2=809 H3=668
S=292 -> H1=341 H2=459 H3=088
S=821 -> H1=374 H2=976 H3=644
S=527 -> H1=959 H2=425 H3=505
S=359 -> H1=465 H2=978 H3=297
S=626 -> H1=242 H2=178 H3=861
S=226 -> H1=977 H2=055 H3=829
S=236 -> H1=610 H2=671 H3=451
S=468 -> H1=902 H2=883 H3=264
S=529 -> H1=275 H2=597 H3=157
S=630 -> H1=507 H2=718 H3=456
S=923 -> H1=806 H2=729 H3=788
S=335 -> H1=628 H2=258 H3=288
S=802 -> H1=397 H2=950 H3=557
S=958 -> H1=497 H2=958 H3=992
S=967 -> H1=013 H2=377 H3=578
S=656 -> H1=806 H2=172 H3=844
S=242 -> H1=184 H2=894 H3=857
S=973 -> H1=037 H2=840 H3=958
S=219 -> H1=293 H2=663 H3=592
S=437 -> H1=174 H2=582 H3=806
S=624 -> H1=623 H2=194 H3=759
S=670 -> H1=266 H2=092 H3=878
```