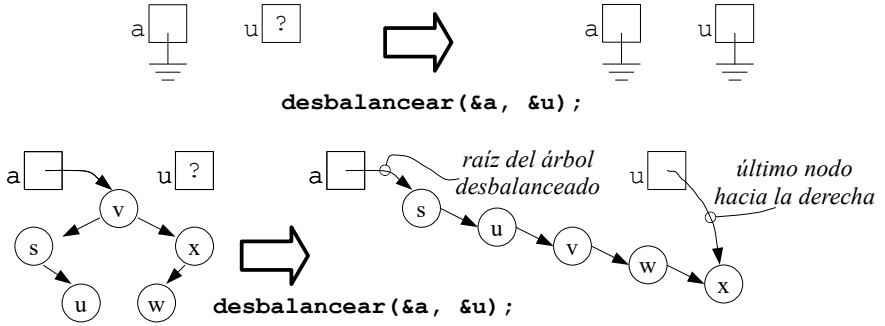
Parte a.- Programe la función:

```
typedef struct nodo {
  int id, hash;  // el campo hash lo usa test-desbalancear.c
  struct nodo *izq, *der;
} Nodo;
void desbalancear(Nodo **pa, Nodo **pult);
```

Esta función recibe en **pa* un árbol de búsqueda binaria (ABB) y entrega en el mismo **pa* un ABB *equivalente* pero desbalanceado al extremo. Un ABB está desbalanceado al extremo si y solo si (i) es el árbol vacío, o (ii) su subárbol izquierdo está vacío y su subárbol derecho está desbalanceado al extremo. Es decir todos sus nodos tienen su subárbol izquierdo vacío. Además la función entrega en **pult* la dirección del último nodo yendo hacia la derecha.

Las 2 figuras de más abajo muestran 2 ejemplos de uso. Las variables *a* y *u* son de tipo *Nodo**.



Restricciones: Debe ser recursivo. Su solución debe tomar tiempo O(*n*), en donde *n* es el número de nodos del árbol. No puede pedir memoria adicional con *malloc*. Reutilice los mismos nodos, reasignando los campos *izq* y *der*. Recuerde: el resultado debe seguir siendo un ABB.

Ayuda: Considere el caso en que el árbol y sus subárboles izquierdo y derecho no son vacíos. Sea *I* el subárbol izquierdo desbalanceado y *UI* su último nodo. Haga que el subárbol **derecho** de *UI* sea el nodo **pa*. Sea *D* el subárbol derecho desbalanceado y *UD* su último nodo. Haga que el subárbol izquierdo del nodo **pa* sea el árbol vacío y haga que su subárbol derecho sea *D*. Ahora decida Ud. qué valores debe entregar en **pa* y **pult* y qué hacer cuando el árbol o alguno de sus subárboles son árboles vacíos.

Parte b.- Programe la función:

```
Nodo *desbalanceado(Nodo *a, Nodo **pult);
```

Esta función recibe en *a* un ABB y retorna un nuevo ABB equivalente a *a*, pero desbalanceado al extremo. No puede modificar el árbol *a*. Debe pedir memoria con *malloc* para los nodos del nuevo árbol. Además la función entrega en **pult* la dirección del último nodo yendo hacia la derecha. Al crear un nuevo nodo a partir de un nodo de *a*, debe copiar los campos *id* y *hash*.

Restricciones: Debe ser recursivo. Su solución debe tomar tiempo O(*n*), en donde *n* es el número de nodos del árbol. Recuerde: el resultado debe seguir siendo un ABB.

Instrucciones

Baje *t3.zip* de U-cursos y descomprímalo. El directorio *T3* contiene los archivos (a) *test-desbalancear.c*, que prueba si su tarea funciona, (b) desbalancear.*h* que incluye los encabezados de las funciones pedidas, (c) *Makefile* que le servirá para compilar su tarea y (d) prof.ref-x86_64 y prof.ref-aarch64 binarios de benchmark. Ud. debe crear un archivo *desbalancear.c* y programar ahí las funciones pedidas.

Pruebe su tarea bajo Debian 12 nativo o virtualizado con VirtualBox, Vmware, Qemu o WSL 2. **Ejecute el comando make sin parámetros**. Le mostrará las opciones que tiene para compilar su tarea. Estos son los requerimientos para aprobar su tarea:

- *make run* debe felicitarlo. Su solución no debe ser 80% más lenta que la solución del profesor.
- *make run-g* debe felicitarlo.
- *make run-san* debe felicitarlo y no reportar ningún problema como por ejemplo goteras de memoria.

Cuando pruebe su tarea con *make run* asegúrese que su computador esté en modo alto rendimiento y que no estén corriendo otros procesos intensivo en CPU al mismo tiempo.

Entrega

Ud. solo debe entregar por medio de U-cursos el archivo *desbalancear.zip* generado por el comando *make zip*. **A continuación es muy importante que descargue de U-cursos el mismo archivo que subió, luego descargue nuevamente los archivos adjuntos y vuelva a probar la tarea tal cual como la entregó**. Esto es para evitar que Ud. reciba un 1.0 en su tarea porque entregó los archivos equivocados. Créame, sucede a menudo por ahorrarse esta verificación. Se descontará medio punto por día de atraso. No se consideran los días de receso, sábados, domingos o festivos.