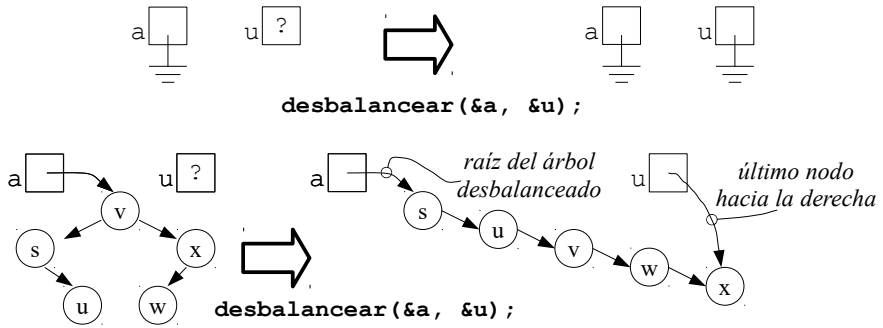
Parte a.- Programe la función:

```
typedef struct nodo {
  int id, hash;
  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 izquierdo 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. que 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*, no olvide 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-t3.c*, que prueba si su tarea funciona, (b) *t3.h* que incluye los encabezados de las funciones pedidas, y (c) *Makefile* que le servirá para compilar su tarea. Ud. debe crear un archivo *t3.c* y programar ahí las funciones pedidas. Compile su tarea con el comando *make* sin parámetros. Depure su tarea con el comando *ddd test-t3*.

A partir de esta tarea es obligatorio que Ud. pruebe su tarea con valgrind. Para ello compile su tarea con *make test-valgrind-ddd* y lance ddd de la forma que se indicó en la tarea 2. Valgrind no debe reportar ningún problema. De lo contrario su tarea será rechazada.

Una vez que su tarea funcione correctamente en la plataforma de su elección, cree un archivo *resultados.txt* y pruebe su tarea bajo Debian 10 (con soporte para programas de 32 y 64 bits) con los comandos *make test-g*, *make test-O* y *make test-O-m32*. Copie y pegue la salida de las 3 pruebas en el archivo *resultados.txt*. La compilación no puede arrojar errores o warnings y la ejecución debe terminar mostrando "Felicitaciones ..." para las 3 pruebas. De otro modo su tarea será rechazada.

Entrega

Ud. solo debe entregar los archivos *t3.c* y *resultados.txt* en el formato *.zip* por medio de U-cursos. Se descontará medio punto por día de atraso. No se consideran los días de vacaciones, sábado, domingo o

festivos.