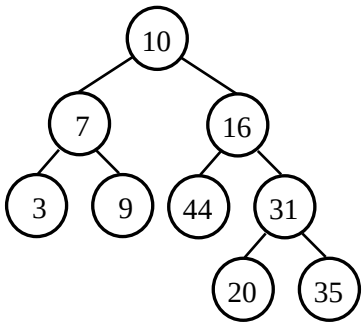
Problema 1

Considere la siguiente definición del tipo *AB* de los árboles binarios de enteros, en memoria dinámica:

typedef nodoAB * AB
struct nodoAB { int dato; ABB izq, der; }

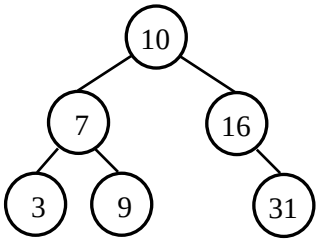
1. Un árbol binario perfecto es un árbol lleno en donde todas las hojas están en el mismo nivel. Un árbol lleno es un árbol donde todos los nodos interiores tiene dos sub arboles.

t1



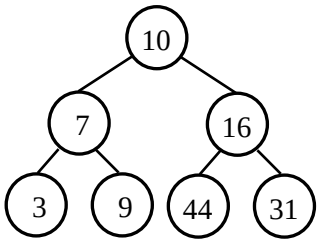
No es árbol perfecto, las hojas están en diferentes niveles.

t2



No es árbol perfecto, el nodo interior 16 no tiene dos sub arboles

t3



Es árbol perfecto

Implemente una función recursiva maxAlturaPerfecto que dado un AB a, retorna la altura del árbol máximo perfecto con la misma raíz que el AB a pero podando los nodos que considere necesarios.

Nota: La altura del árbol máximo perfecto es igual al largo del camino entre la raíz y la hoja mas cercana. La altura del árbol vacío es 0.

int maxAlturaPerfecto (AB a)

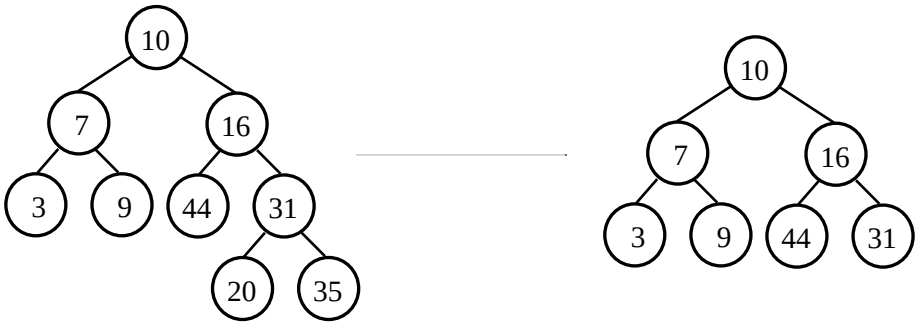
Ejemplos:

- *maxAlturaPerfecto(t1) = 3*
- *maxAlturaPerfecto(t2) = 2*
- *maxAlturaPerfecto(t3) = 3*

2. Implemente una función ***podaArbolPerfecto*** que dado un AB a, retorna el árbol máximo perfecto con la misma raíz que el AB a pero podando los nodos que considere necesarios.

   El árbol retornado no puede compartir memoria con a. El árbol máximo perfecto del árbol vacío es el propio árbol vacío.

   AB podaArbolPerfecto (AB a)



Ejemplo de árbol retornado por la función podaArbolPerfecto

Problema 2

Considere la siguiente especificación del TAD Tabla no acotada de int (dominio) en char* (rango).

struct representacionTabla;

typedef representacionTabla* Tabla;

// PRE: -
// POS:Retorna la tabla vacia no acotada, donde cantidad es una estimación de la cantidad de correspondencias a guardar (aunque pueden llegar a guardarse mas correspondencias que 'cantidad').
Tabla crearTabla (int cantidad);

// PRE: -
// POS: Agrega la correspondencia (dom, ran) a la tabla 't', siempre y cuando 'dom' no tenga imagen en la tabla 't'; en caso contrario actualiza la imagen de 'dom' con 'ran'. La correspondencia insertada no debe compartir memoria con el dominio y rango que se recibe como parametro en esta operación.
void insertar (Tabla& t, int dom, char* ran);

// PRE: Existe una correspondencia en la tabla 't' con 'dom' como dominio.
// POS: Elimina la correspondencia que tiene a 'dom' como dominio en la tabla 't'.
void eliminar (Tabla& t, int dom);

// PRE: Existe una correspondencia en la tabla 't' con 'dom' como dominio.
// POS: Retorna la imagen de 'dom' en la tabla 't'. La imagen retornada no comparte memoria.
char* recuperar (Tabla t, int dom);

// PRE: -
// POS: Retorna true si 'dom' tiene imagen en la tabla 't', en caso contrario retorna false.
bool estaDefinida (Tabla t, int dom);

Se pide:

1. **Implemente el TAD *Tabla*** de tal manera que las operaciones *insertar, eliminar, recuperar y estaDefinida* tengan O(1) de tiempo de ejecución en caso promedio. Para ello: *i)* desarrolle la representación del TAD (*representacionTabla*), justificando la elección; y *ii)* implemente las operaciones: *crearTabla, insertar, y eliminar*. Asuma que las restantes operaciones están implementadas.

   Si lo considera conveniente, puede utilizar las siguientes funciones:

   - *PRE: -*
     POS: Devuelve el menor número mayor a n que sea primo.
     int mayorNúmeroPrimo(int n)

   - *PRE: -*
     POS: Devuelve una copia sin compartir memoria del array c.
     char* copia(char* c)

2. Justifique cuales son las dos condiciones necesarias para que la operación *insertar* tenga O(1) de tiempo de ejecución en caso promedio, teniendo en cuenta la representación elegida en la parte 1.