Organización del computador 2

Trabajo Práctico 1: Árboles

Fecha de entrega: 3 de octubre de 2022

Universidad Nacional de General Sarmiento

Universidad Nacional de General Sarmiento

Instituto de Industria

Desarrollado en C y lenguaje ensamblador para arquitectura x86

Felipe Corso Rodrigues y **Matias Agustin Morales**

Este documento describe lo que realizamos para resolver los problemas propuestos en el enunciado. Además, les comentamos dificultades encontradas en el camino y una breve explicación de cómo se resuelve cada problema.

1. Implementación de las funciones en asm y llamadas desde c. Para reservar memoria y crear los árboles se utilizó la función llamada malloc, la cual nos permite reservar porciones de memoria para guardar nuestros árboles.

   ab* crearArbolB(int valor)

Primero tenemos que acceder al nodo paso por parámetro el cual lo guardamos en el registro ECX, una vez hecho esto llamamos a malloc y le pedimos que nos reserve 12 bytes de memoria, el mismo nos los devuelve en el registro EAX, por lo tanto para guardar nuestros nodos nos vamos moviendo dentro del registro EAX ([EAX],[EAX+4],[EAX+8]). Notemos que guardamos el valor pasado por parámetro en la primera posición y luego guardamos los otros dos nodos como vacíos (es decir con un 0). Finalmente regresamos a EAX.

A continuación se adjunta una imagen del código:

```
crearArbolBA:
    push ebp              ;apilo ebp
    mov ebp, esp          ;muevo el esp al ebp

    mov ecx, [ebp + 8]    ;valor parametro

    push ecx              ;pusheo el valor para no perderlo
    push 12               ;pusheo 12 bytes para malloc
    call malloc           ;recervo la memoria con malloc
    add esp, 4            ;restauro pila
    pop ecx               ;pop de valor

    mov [eax], ecx        ;guardo valor
    mov edx,0
    mov [eax + 4], edx    ;seteo nodo a puntero izq en 0
    mov [eax + 8], edx    ;seteo nodo a puntero der en 0

    jmp finalizar         ;termino programa
```

arbol ab* crearArbolB(int valor, ab* izq, ab* der)

Primero guardamos en los registros ECX, EBX Y EDX los nodos pasados por parámetros. Análogamente con la función anterior, usamos malloc para reservar 12 bytes y guardar los nodos en el registro EAX, pero a diferencia de la otra función usamos una etiqueta (agregarNodo) extra la cual guarda en [EAX+4] y [EAX+8] los nodos izquierdo y derecho pasados por parámetro. Finalmente también retornamos a EAX.

A continuación se adjunta una imagen del código:

```
crearArbolB:
    push ebp              ;apilo ebp
    mov ebp, esp          ;muevo a ebp el esp

    mov ecx, [ebp + 8]    ;valor parametro
    mov ebx, [ebp + 12]   ;valor puntero izq
    mov edx, [ebp + 16]   ;valor puntero der

    call agregoNodo       ;llamo a agregar nodo
    jmp finalizar         ;fin del programa

agregoNodo:
    push ecx              ;pusheo ecx de respaldo
    push edx              ;pusheo edx de respaldo
    push 12               ;pusheo 12 bytes para malloc
    call malloc           ;recervo memoria con malloc
    add esp, 4            ;vuelvo la pila a su lugar
    pop edx               ;desapilo edx
    pop ecx               ;desapilo ecx

    mov [eax], ecx        ;guardo valor
    mov [eax + 4], ebx    ;seteo nodo a puntero izq
    mov [eax + 8], edx    ;seteo nodo a puntero der

    jmp finalizar         ;termino programa
```

2. Se realizó el siguiente código de C con el fin de utilizar las funciones creadas en lenguaje ensamblador.

Primero definimos la estructura de árbol definida en el enunciado del trabajo práctico, después definimos las funciones que serían llamadas desde nasm y la función mostrar_arbol() realizada en c. La función crearArbolBA nos crea un árbol desde nasm, pasándole como parámetro un número, el cual va ser el valor nodo raíz del árbol. La función crearArbolB nos crea un árbol pasándole por parámetro un número y dos nodos los cuales vana ser enlazados al valor inicial, de esta forma a través de nasm nos crea un árbol. La última función mostrar_arbol() nos muestra por pantallas los nodos del árbol y con un 0 podemos visualizar los nodos vacíos.

Luego está el main el cual ejecuta todas las funciones anteriormente nombradas. Podemos hacer una división de 4 bloques de código en el main (tomados como división de bloques a los espacios entra el código). Los dos primero definimos una estructura de árbol y las igualamos a la la función crearArbolBA para inicializar estas variables, después imprimimos con por pantalla estos árboles. El tercer bloque lo usamos para crear árboles con la función crearArbolB a la cual le pasamos los árboles creados en los bloques anteriores, de esta forma usamos ambas estructuras de árbol creadas por las funciones implementadas en nasm. El último bloque nos imprime por pantalla el último árbol creado (ar2).

Para correr el programa usamos las siguientes líneas de código que linkean el archivo de nasm con el de C y lo compila:

- nasm -f elf -o arbol.o argol.asm
- gcc -m32 -o invocar arbol.c arbol.o
- ./invocar

Salida del programa:

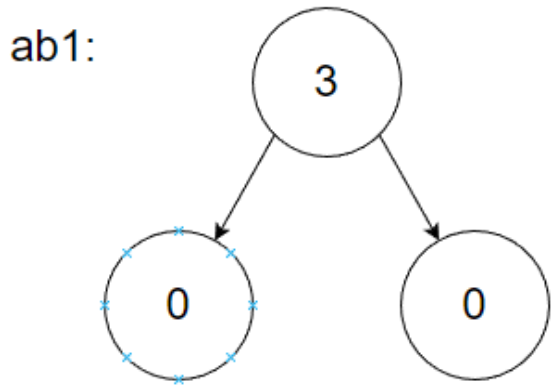
```
Primer arbol:
nodo :3
nodo :0 nodo :0
----------------------
Segundo arbol:
nodo :4
nodo :0 nodo :0
----------------------
Arbol con dos punteros:
nodo :5
nodo :12
nodo :3
nodo :0 nodo :0
nodo :4
nodo :0 nodo :0
nodo :7
nodo :0 nodo :0
```

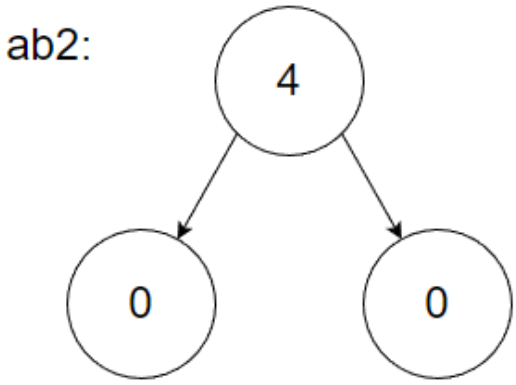
A continuación se adjunta una imagen del código en C:

```c
struct arbol{
        int valor;
        struct arbol* izq;
        struct arbol* der;
};

struct arbol* crearArbolBA(int valor);
struct arbol* crearArbolB( int valor,struct arbol* izq, struct arbol* der);
void mostrar_arbol(struct arbol* ar);

int main(int argc, char **argv)
{
    //Creo un arbol con funcion crearArbolBA y lo imprimo
    struct arbol* ab1;
    printf("Primer arbol: \n");
    ab1=crearArbolBA(3);
    mostrar_arbol(ab1);
    printf("---------------------\n");

    //Creo otro arbol con funcion crearArbolBA y lo imprimo
    struct arbol* ab2 ;
    printf("Segundo arbol: \n");
    ab2=crearArbolBA(4);
    mostrar_arbol(ab2);
    printf("---------------------\n");

    //Creo un arbol con mas nodos
    struct arbol* ar = crearArbolB(12,ab2,ab1);
    struct arbol* ar2 =crearArbolB(5,ar,crearArbolBA(7));


    printf("Arbol con dos punteros: \n");
    mostrar_arbol(ar2);
}
```

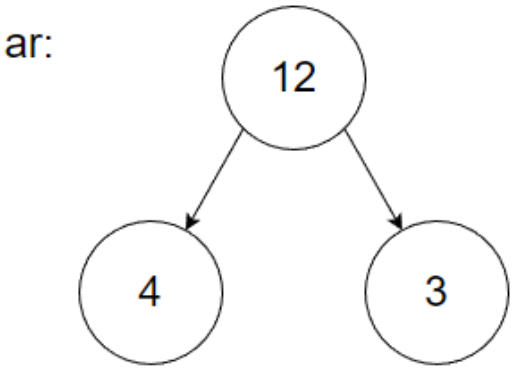
Gráficos de los árboles creados:

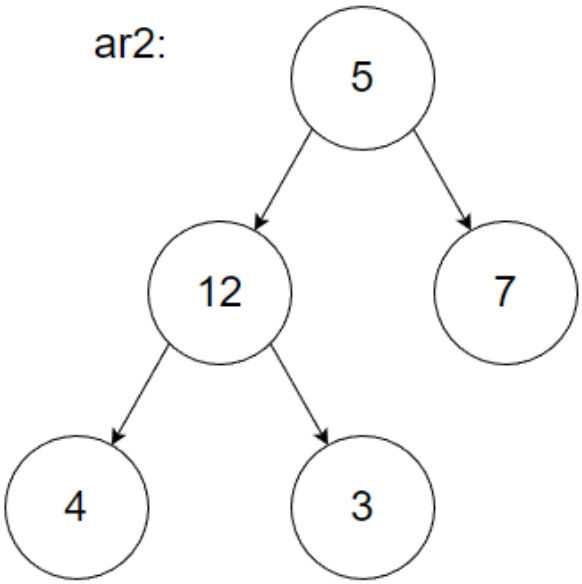
-Arboles creados con las función **crearArbolBA**





-Arboles creados con las función **crearArbolB**





3. La función **minimo(&arbol)** recibe por parámetro una dirección de memoria de un árbol y devuelve el nodo con valor mínimo.

Primero tenemos la etiqueta buscarMinimo la cual nos guarda el valor del primer nodo en la variable inicializada “mínimo”, una vez guardado el valor vamos a la etiqueta buscarMinimoAux la cual va a recorrer el árbol de forma recursiva y en cada iteración va a estar llamando a la etiqueta compararMinimo que será explicada a continuación, una vez que termina de recorrer el árbol mueve al registro EAX el nodo guardado en la variable mínimo para retornarla y finalizar el programa. La etiqueta compararMinimo compara el valor guardado en la variable “minimo” con el valor del nodo en el cual estamos parados, si el valor es mínimo va a la etiqueta guardarMinimo para guardar el valor en la variable “mínimo”.

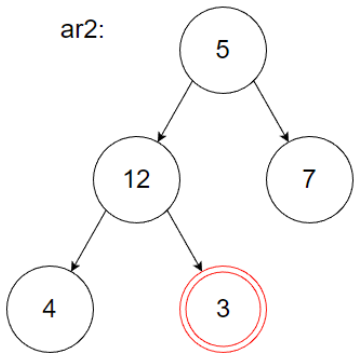
A continuación se adjunta las imágenes del código:

```
buscarMinimo:                   ;guarda el primer valor como minimo
    push ebp
    mov ebp, esp

    mov ebx, [ebp + 8]          ;accedo al valor del primer nodo
    mov ebx,[ebx]
    mov [minimo],ebx            ;guardo el valor del nodo en res

    mov esp, ebp
    pop ebp

    jmp buscarMinimoAux
```

```
buscarMinimoAux:                    ;recorre para buscar el minimo
    push ebp
    mov ebp, esp

    mov ebx, [ebp + 8]          ;puntero nodo
    mov ecx,[minimo]            ;pongo el minimo en ecx para comparar

    cmp ebx, 0                  ;si es cero finalizo
    je finalizar

    call compararMinimo           ;verifico si es minimo y lo guarda

    push ebx                    ;apilo nodo actual para pasar por parametr
    mov eax, [ebx + 4]          ;paso el nodo izq
    push eax                    ;apilo el nodo izq para pasar por parametr
    call buscarMinimoAux        ;llamo a buscarMin para la recursividad
    add esp, 4                  ;recupero esp
    pop ebx                     ;recupero nodo actual
    push ebx                    ;apilo nodo actual para pasar por parametr
    mov eax, [ebx + 8]          ;paso el nodo derecho
    push eax                    ;apilo el nodo der para pasar por parametr
    call buscarMinimoAux            ;llamo a buscarMin para la recursivi
    add esp, 4                  ;recupero esp

    mov eax,minimo              ;guardo el minimo para retornarlo
    jmp finalizar               ;finalizo programa
```

```
compararMinimo:
     cmp ecx,[ebx]              ;compara si el valor del nodo es
     jg guardarMinimo                    ;salta a min para guard
     ret

guardarMinimo:
     mov ecx,[ebx]              ;muevo el valor del nodo a ecx
     mov [minimo],ecx           ;guardo en minimo el nodo
     ret
```

Gráfico del árbol con el nodo mínimo en rojo:



Codigo en c:

```
printf("Nodo Minimo: %d \n",buscarMinimo(ar2)->valor) ;
printf("----------------------\n");
```

Salida del programa ejecutado desde C:

```
----------------------
Arbol con dos punteros:
nodo :5
nodo :12
nodo :3
nodo :0 nodo :0
nodo :4
nodo :0 nodo :0
nodo :7
nodo :0 nodo :0
----------------------
Nodo Minimo: 3
----------------------
```

4. Se implementó la función void eliminarTodos(ab* árbol, int valor) en ASM

Primero tomamos los valores pasados por parámetros y comparamos si el valor del parámetro es igual al valor del nodo a eliminar, si así es vamos a la etiqueta nodoEncontrado, si no es vamos a desplazarnos recursivamente para la izquierda y para la derecha para encontrar el nodo a borrar. Cuando encontramos el nodo vamos a la etiqueta borrarNodo la cual va a recorrer el árbol a partir del nodo especificado y de forma recursiva va ir eliminando los nodos en la etiqueta liberarNodo la cual con la función de C llamada free libera el espacio en memoria asignado anteriormente con malloc. Una vez que eliminamos todos los nodos correspondientes finalizamos el programa.

A continuación se adjuntan imagenes del código:

```
eliminarTodos:
    push ebp
    mov ebp, esp

    mov edx, [ebp + 12]      ;valor parametro
    mov ebx, [ebp + 8]       ;puntero nodo
    cmp ebx, 0               ;si es cero no hay nodo
    je finalizar             ;finalizo programa
    cmp edx, [ebx]           ;comparo el parametro con valor nodo
    je nodoEncontrado
    jmp borrarIzq            ;finalizo programa

nodoEncontrado:
    push ebx                 ;guardo el valor
    call borrarNodo          ;salto a borrar nodo
    add esp, 4               ;recupero esp
    jmp finalizar            ;finalizo programa
```

```
borrarNodo:
    push ebp
    mov ebp, esp

    mov ebx, [ebp + 8]        ;puntero nodo
    cmp ebx, 0                ;si es cero finalizo
    je finalizar

    call liberarNodo

    push ebx                  ;guardo nodo actual
    mov eax, [ebx + 4]        ;paso el nodo izq
    push eax
    call borrarNodo                ;llamo a borrarNodo\

    add esp, 4                ;recupero esp
    pop ebx                   ;recupero nodo actual
    push ebx                  ;guardo el nodo actual
    mov eax, [ebx + 8]        ;paso el nodo derecho
    push eax
    call borrarNodo                ;llamo a borrar nodo
    add esp, 4                ;recupero esp

    jmp finalizar             ;finalizo programa

liberarNodo:
    push ebx                   ;pusheo el nodo actual
    call free                  ;libero el nodo actual
    add esp, 4
    ret
```

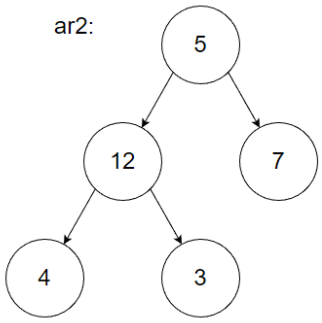
Codigo en C para eliminar:

```
printf("Prueba de elimar nodo: \n") ;
mostrar_arbol(eliminarTodos(ar2,5));
```

Salida del código:

```
 ------------------------
Prueba de elimar nodo:
nodo 0 (vacio)
```

Ejemplos de nodos borrados para el siguiente árbol:



Borrado de 12:

```
----------------------
Prueba de elimar nodo:
nodo :5
nodo :0 nodo :7
nodo :0 nodo :0
```

Borrado de 7:

```
----------------------
Prueba de elimar nodo:
nodo :5
nodo :12
nodo :3
nodo :0 nodo :0
nodo :4
nodo :0 nodo :0
nodo :0
```

Borrado de 3:

```
Prueba de elimar nodo:
nodo :5
nodo :12
nodo :0 nodo :4
nodo :0 nodo :0
nodo :7
nodo :0 nodo :0
```

Problemas encontrados:

En el primer punto la mayor dificultad fue entender cómo funcionaba malloc para poder guardar los nodos en memoria, una vez entendido se nos dificulto pensar cómo enlazar los nodos pero pudimos generar el nodo necesario en ambos casos.

En el segundo punto no nos encontramos con mayores dificultades. Existía la duda de si los punteros a árboles pasados desde c, eran a árboles que tenían la misma estructura que los árboles que creamos nosotros en asm. Pero se llegó a que la estructura si permite esta similitud.

El tercer punto se pudo resolver parcialmente. No se pudo hacer un recorrido de los subárboles del nodo inicial. Por lo tanto solo se evalúa el nodo inicial y los valores de sus subnodos.

En el cuarto punto la mayor dificultad fue la de recorrer el árbol y luego borrar sus subárboles con la función free de c. Pudimos testear su funcionamiento para los nodos hijos pero para el nodo raíz no estamos seguros de su correcto funcionamiento.