Universidad Autónoma de San Luis Potosí

Facultad de Ingeniería

Área en Ciencias de la Computación

Estructuras de Datos II

Semestre: Agosto – Diciembre 2022

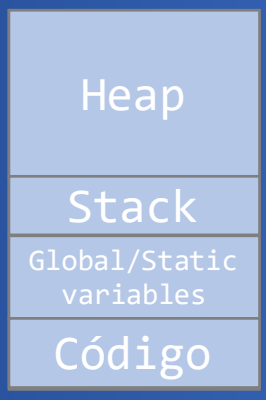
Alumno: Esparza Castañeda Hugo

Profesor: M. en C. Froylán Eloy Hernández Castro

Índice General de Temas

Martes 16 Agosto 2022

1. APUNTADORES

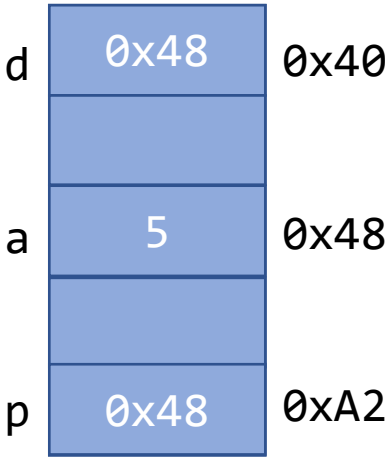


1.1 Introducción a apuntadores

Objetivo: Conocer el concepto de apuntadores en el paso de parámetros de funciones y gestión de memoria (Dinámica), así como ser capaz de utilizarlos.

Apuntador: un apuntador es una variable que contiene una dirección de memoria pero su contenido es otra dirección de memoria.

```c
int main(){
    int a;
    a = 5;
    int *p;
    p = &a;
    int *d;
    d = p;
    return 0;
}
```




```c
printf("%d", a;); //output: 5
printf("%d", *p;); //output: 5
printf("%p", p;); //output: 0x48
printf("%d", *d;); //output: 5
printf("%p", d;); //output: 0x48
```

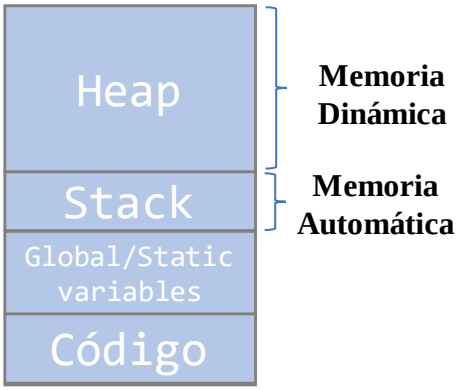
`a = 10; ⇔ *p = 10`

`scanf("%d", p) ⇔ scanf("%d", &a)`

1.2 Parámetros por valor/referencia

Jueves 18 Agosto 2022

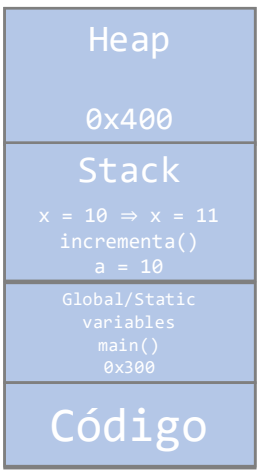
Memoria



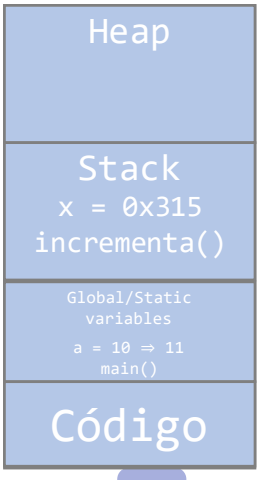

```c
#include <stdio.h>
void incrementa(int *);
int main(){
    int a;
    a = 10;
    incrementa(&a);
    printf("%d", a);
    return 0;
}

void incrementa(int *x){
    *x = *x + 1;
    return;
}
```

Por valor



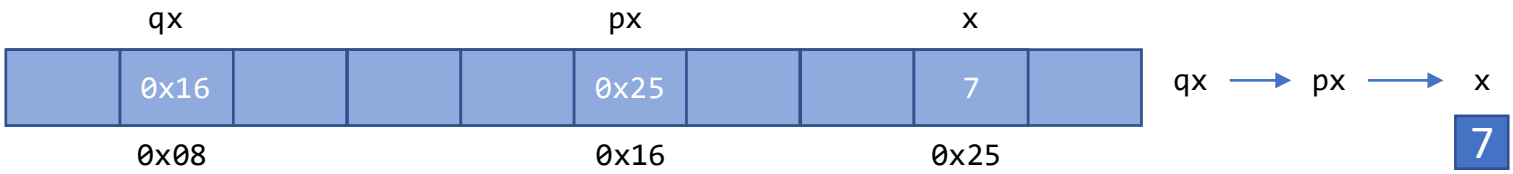
Por referencia



```
int *r;   ⇒   int *r = NULL;
```

1.3 Apuntadores a apuntadores

Viernes 19 Agosto 2022



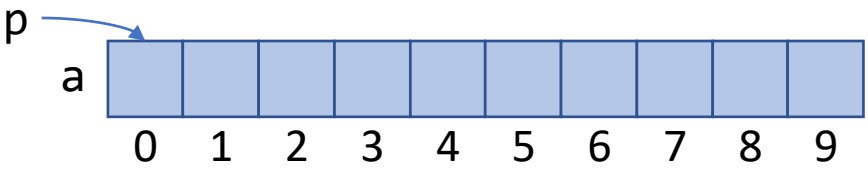

```c
#include <stdio.h>
int main(){
    int x = 7; //entero
    int *px = &x; //apuntador
    int **qx = &px; //apuntador a apunt
    printf("%p\n", px);  //output: 0x25
    printf("%d\n", *px); //output: 7
    printf("%p\n", &x);  //output: 0x25
    printf("%p\n", &px); //output: 0x16
    printf("%p\n", &qx); //output: 0x08
    printf("%p\n", qx);  //output: 0x16
    printf("%p\n", *qx); //output: 0x25
    return 0;
}
```

* = indirección = Obtener el valor almacenado en una dirección.
& = dirección = Obtener la dirección de una variable.

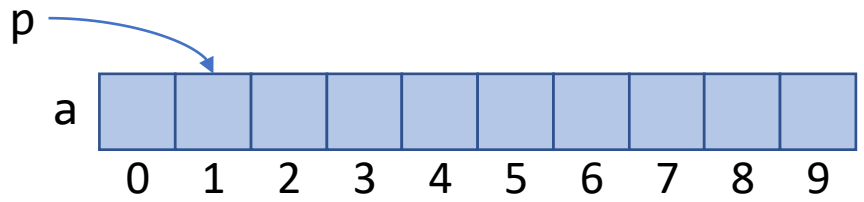
Los apuntadores a apuntadores se usan para matrices dinámicas.

1.4 Apuntadores y arreglos

```
int a[10];
int *p = a; ⇒ int *p = &a[0];
```

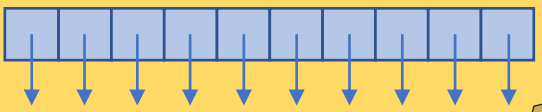



```
p = p + 1;
```



Arreglo de apuntadores

```
int *b[10];
```



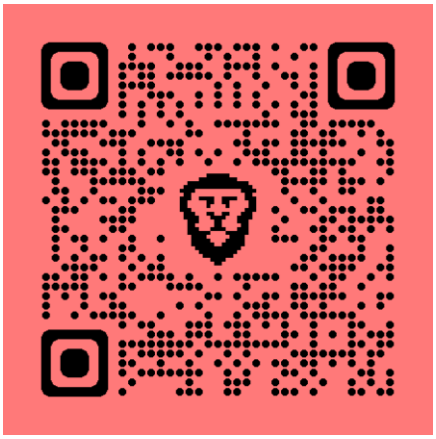
Arreglo de apuntadores apuntando a NULL

```
for(int i = 0; i < 10; i++){
    b[i] = NULL;
}
```

⇔

```
int *b[10] = {NULL};
```

Tarea #0: Resumen del video “Pointers”



Los punteros es uno de los temas más difíciles, sim embargo son de mucha utilidad, como por ejemplo al pasar datos a una función, modificarlos y luego regresarlos, cosa que no podría hacerse de otra manera.
Cuando pasamos un dato por valor a una función, lo que estamos pasando es en realidad una copia, sin embargo, cuando pasamos el dato por referencia haciendo el uso de punteros, lo que hacemos es darle acceso a la memoria en donde esta almacenado dicho valor, haciendo que todos los cambios que realicemos en la función, le sucedan al valor mismo.
Cada archivo de nuestra computadora esta almacenado en el disco duro, o en un disco de estado solido, según sea el caso. Esos discos son solo de almacenamiento, por lo tanto, no podemos trabajar directamente ahí, la manipulación de los datos únicamente puede hacerse en la memoria RAM, por lo que tenemos que movernos ahí. La memoria RAM es como un arreglo enorme. La memoria RAM es Memoria de Acceso Aleatorio, y una vez que apagamos la computadora, todos los datos de la memoria RAM se destruyen.

Retomando lo que dijimos hace un momento que la memoria RAM es un arreglo enorme dividido en celdas de bytes (dependiendo del tamaño del tipo de dato), así como los arreglos tienen índices para cada valor guardado, la memoria tiene una dirección para cada dato almacenado.
Las variables de tipo str o cadena de texto, necesitan llevar el `'\0'` para poder saber donde terminan. Una vez aclarado el tema de la memoria, lo más importante de recordar sobre los punteros es, que son solo direcciones de memoria.

Tipo de dato	Tamaño (en bytes)
int	4
char	1
float	4
double	8
long long	8
char*	4 u 8

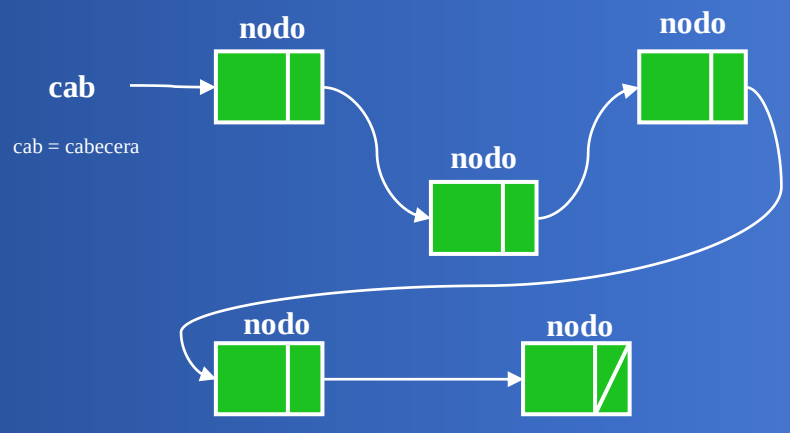
Entonces, un puntero es un tipo que dato que guarda una dirección, y el tipo únicamente nos describe el dato guardado en esa dirección de memoria.
Al saber la dirección de la memoria en donde esta guardada una variable, podemos ir directamente a esa ubicación en la memoria y manipular el dato, es por eso que no es necesario hacer una copia cuando enviamos un valor por referencia en una función.
El puntero más simple en C es el puntero `NULL`, que como su nombre o indica, no apunta a nada (lo cual en realidad puede ser muy útil).Siempre que declaramos un puntero, y no le asignamos una dirección, debemos hacer `NULL` el valor de ese puntero (son buenas practicas de programación).Podemos extraer la dirección de una variable con el operador `&` (como en los ejemplos anteriores).
El propósito principal de un puntero es permitirnos modificar o inspeccionar la ubicación a la que apunta.
Si tenemos un puntero pc, entonces `*pc` es el dato que vive en la dirección de memoria guardada en pc.
Usado en contexto, * es el operador que nos permite acceder al dato almacenado en la dirección, dicho de otro modo, no nos sirve únicamente saber la dirección, también debemos ir allí, y el operador * nos permite hacerlo.
Por lo tanto, cuando tenemos un puntero que apunta a `NULL`, es decir a nada, e intentamos ir a esa dirección con el operador * nos aparecerá el error segmentation fault.
Es por esto la importancia de declarar todos los apuntadores a `NULL` cuando no se les asigna inmediatamente una dirección significativa, ya que si no lo hacemos, nuestro apuntador podría apuntar a cualquier espacio de memoria y manipularlo accidentalmente.

1.5 Aritmética de apuntadores

Lunes 22 Agosto 2022

Operador	Nombre
*	Indirección
&	Dirección
->	Apunta a
+	Suma
-	Resta
==	Igualdad
!=	Desigualdad
>	Comparación de direcciones
>=	
<	
<=	
(tipo de dato)	Cast ó conversión explicita

2. LISTAS ENLAZADAS

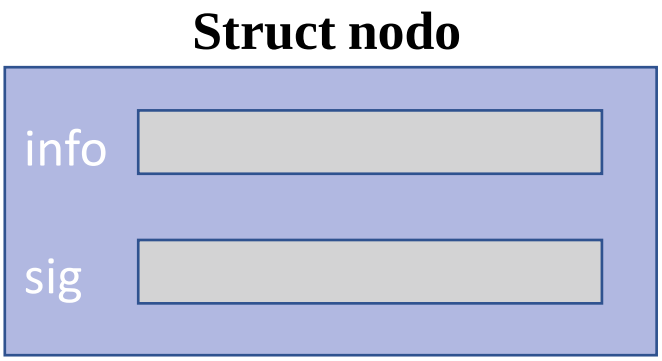


2.1 Listas simples

Objetivo: Ser capaz de diseñar diversos tipos de listas enlazadas y programar las principales operaciones para su manipulación.

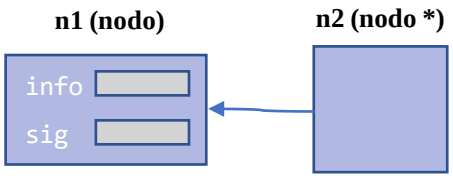
En una lista podemos guardar cualquier cosa, para este ejemplo de lista simple, haremos una lista de números enteros, cada casilla es un nodo, y para cada nodo hacemos una estructura con los datos que almacenara cada nodo.

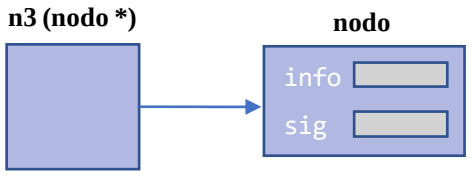
2.1.1 Definir la estructura del nodo

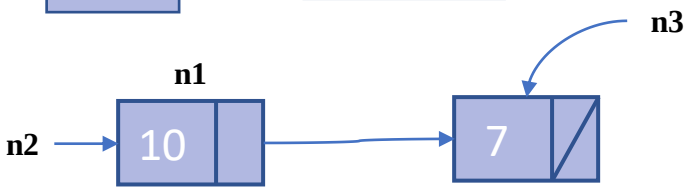



```c
struct nodo{
    int info;
    struct nodo *sig;
};
typedef struct nodo nodo_t;
```

```c
int main(){
    nodo_t n1;
    nodo_t *n2, *n3;
    n1.info = 5;
    n1.sig = NULL;
    n2 = &n1;
    n2->info = 10;
    printf("%d", n1.info);
    n3 = (nodo_t *)malloc(sizeof(nodo_t));
    assert(n3 != NULL);
    n3->info = 7;
    n3->sig = NULL;
    n2->sig = n3;
    return 0;
}
```







Output

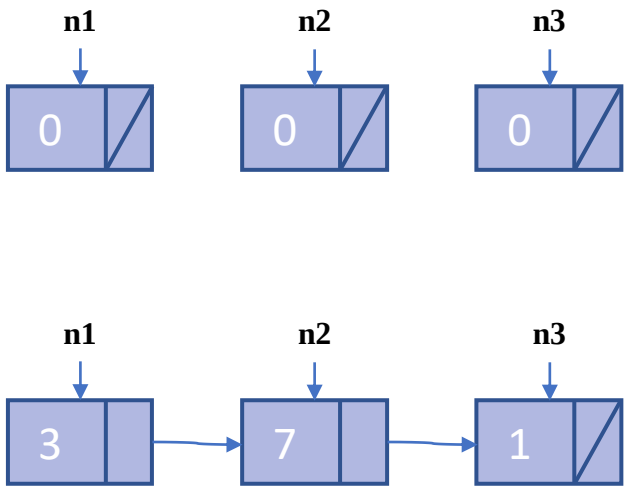
10

2.1.2 Función para crear un nodo

```c
nodo_t *crea_nodo(){
    nodo_t *nodo = NULL;
    nodo = (nodo_t*)malloc(sizeof(nodo_t));
    if(nodo == NULL){
        printf("Error: no hay memoria suficiente");
        exit(EXIT_FAILURE);
    }
    nodo->info = 0;
    nodo->sig = NULL;
    return nodo;
}
```

Ejemplo Burdo

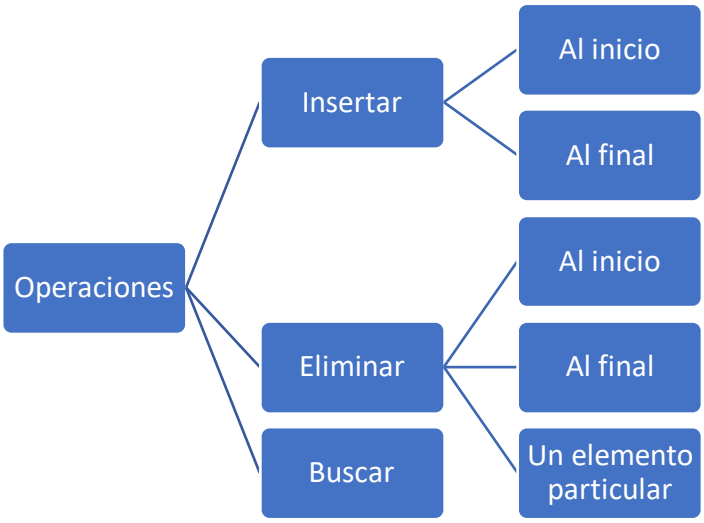
```c
int main(){
    nodo_t *n1 = crea_nodo();
    nodo_t *n2 = crea_nodo();
    nodo_t *n3 = crea_nodo();
    n1->info = 3;
    n2->info = 7;
    n3->info = 1;
    n1->sig = n2;
    n2->sig = n3;
    printf("%d ", n1->info);
    printf("%d ", n1->sig->info);
    printf("%d", n1->sig->sig->info);
    free(n1);
    free(n2);
    free(n3);
    return 0;
}
```



Output

3 7 1

2.1.3 Operaciones en listas enlazadas



2.1.3 Creando un menú de opciones

```c
int main(){
    nodo_t *lista = NULL;
    int opcion = 0, num = 0;
    do{
        opcion = selecciona_opcion();
        switch(opcion){
            case 1: printf("Valor del numero: ");
                scanf("%d", &num);
                insertar_inicio(&lista, num);
                break;
            case 2: printf("Valor del numero: ");
                scanf("%d", &num);
                insertar_final(&lista, num);
                break;
            case 3: eliminar_inicio(&lista);
                break;
            case 4: eliminar_final(&lista);
                break;
            case 5: imprimir_lista(lista);
                break;
            case 6: printf("Valor del numero: ");
                scanf("%d", &num);
                elimina_nodo(&lista, num);
                break;
            case 7: liberar_lista(&lista);
                break;
            case 8: printf("Valor del numero: ");
                scanf("%d", &num);
                printf("%p\n", buscar_dato(lista, num));
                break;
            default: puts("Opcion no valida");
                break;
        }
    }while(opcion != 0);
    return 0;
}
```

2.1.5 Función “int selecciona_opción();”

puts funciona como printf, pero es para imprimir en pantalla puro texto, y ya imprime el salto de línea sin necesidad de ponerlo.

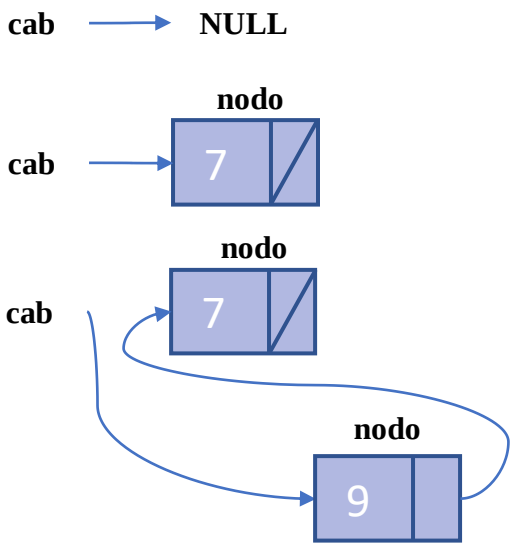
```c
int selecciona_opcion(){
    puts("Selecciona una opcion");
    puts("0. Salir");
    puts("1. Insertar inicio");
    puts("2. Insertar al final");
    puts("3. Eliminar al inicio");
    puts("4. Eliminar al final");
    puts("5. Imprimir toda la lista");
    puts("6. Eliminar nodo arbitrario");
    puts("7. Eliminar/Liberar toda la lista");
    puts("8. Buscar dato");
    int opcion;
    scanf("%d", &opcion);
    return opcion;
}
```

2.1.5 Función “void insertar_inicio(nodo_t **, int);”

```c
void insertar_inicio(nodo_t **cab, int dato){
    nodo_t *nodo = crea_nodo();
    nodo->info = dato;
    nodo->sig = *cab;
    *cab = nodo;
}
```

Ejemplo:

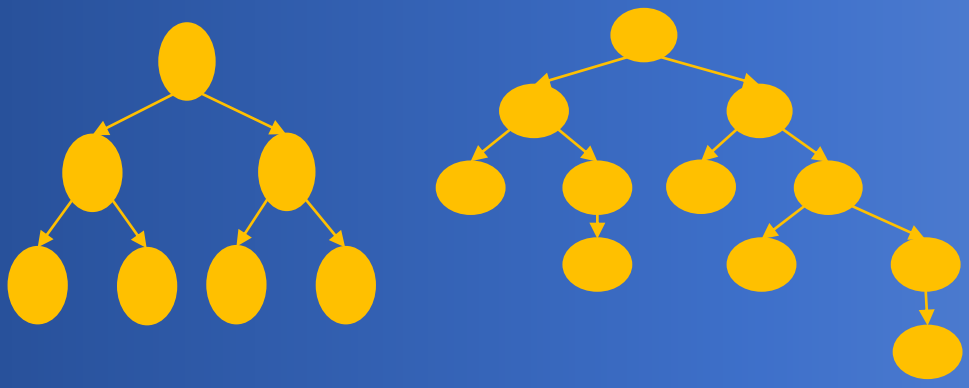
```c
int main(){
    nodo_t *lista = NULL;
    insertar_inicio(&lista, 7);
    insertar_inicio(&lista, 9);
    return 0;
}
```



3. GRAFOS

Viernes 04 Noviembre 2022

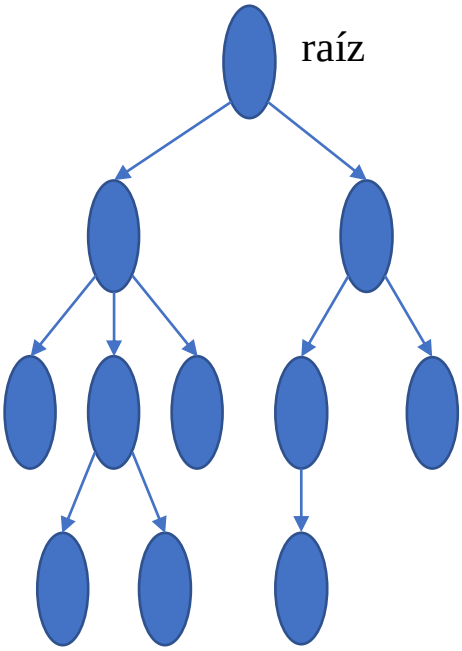
4. ÁRBOLES BINARIOS



4.1 Conceptos básicos de árboles

Objetivo: Diseñar y programar las estructuras y las operaciones básicas para la manipulación de grafos.

Lunes 07 Noviembre 2022

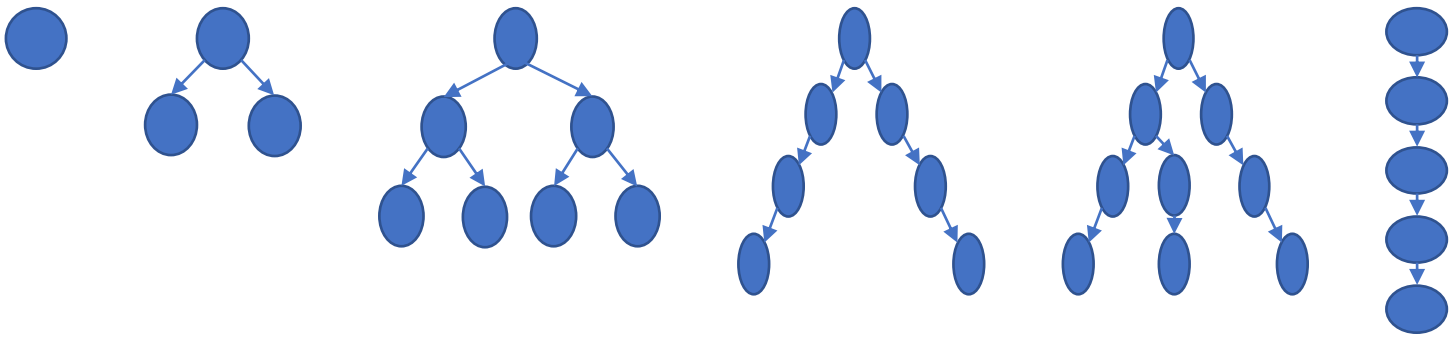


Terminología	
Raíz	Nodo principal, cabecera.
Hijos	Nodos izquierdo y derecho de un nodo.
Padre	Nodo con hijos.
Hermanos	Nodos que tienen el mismo padre.
Antecesor y descendiente	Si podemos ir desde el nodo A hacia el nodo B, entonces A es un antecesor de B, y B es un descendiente de A.
Hoja	Nodos sin hijos.
Nodo interno	Nodos que no son hojas.
Camino	Secuencia de aristas desde un nodo antecesor hasta un nodo descendiente.

Definiciones y aplicaciones

Un **árbol binario** es un árbol en el cual cada nodo puede tener a lo máximo 2 hijos.

Ejemplos:



Aplicaciones de los árboles:

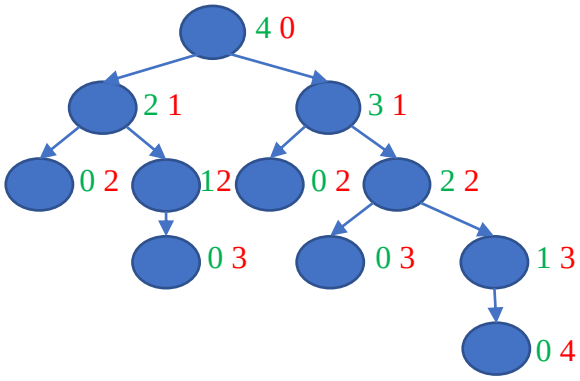
- Información con estructura jerárquica.
- Navegación web y redes.
- Bases de datos (árboles B y B+).
- Son la base para las estructuras eficientes como los conjuntos (sets).

Árbol binario completo

Todos los niveles excepto posiblemente el último están llenos y todos los nodos están lo más posible a la izquierda.

Altura de un nodo

Martes 08 Noviembre 2022



Altura de un nodo x

Es el número de aristas en el camino más largo desde x hasta una hoja. Las hojas tienen altura cero.
La altura de un árbol es la altura de la raíz (en este caso es 4).

La profundidad de un nodo x

Es la longitud del camino desde la raíz hasta el nodo x (cantidad de aristas). La profundidad de la raíz es cero.
Los nodos que tienen la misma profundidad están en un mismo nivel.
En un árbol binario perfecto, todos los niveles están completamente llenos.

El número máximo de nodos en el nivel i es 2^i. Un árbol perfecto de altura h tiene exactamente $2^{n+1} - 1$ nodos.

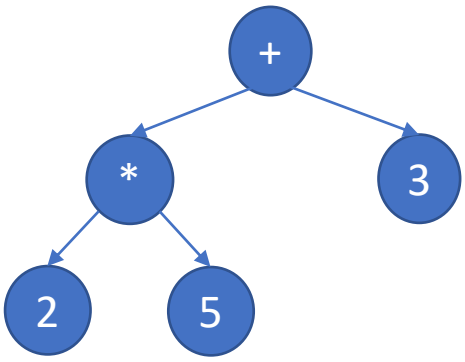
¿Cuál es la altura de un árbol binario perfecto que tiene N nodos?

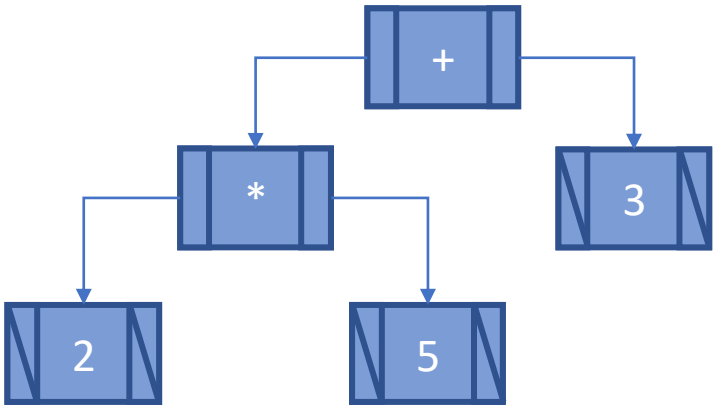
$$2^{n+1} - 1 = N \Rightarrow 2^{n+1} = N + 1 \qquad \log a^b = b \log a \qquad \log_x x = 1$$

$$\log_2\left(2^{h+1}\right) = \log_2(N+1) \Rightarrow (h+1)\log_2 2 = \log_2(N+1) \Rightarrow h = \log_2(N+1) - 1$$

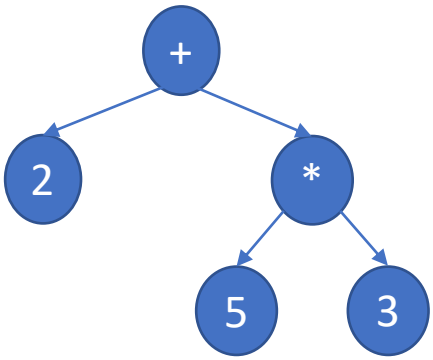
4.2 Árboles binarios de expresiones

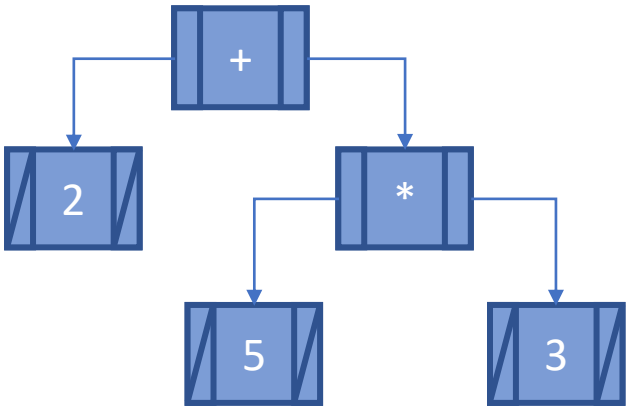
Jueves 10 Noviembre 2022



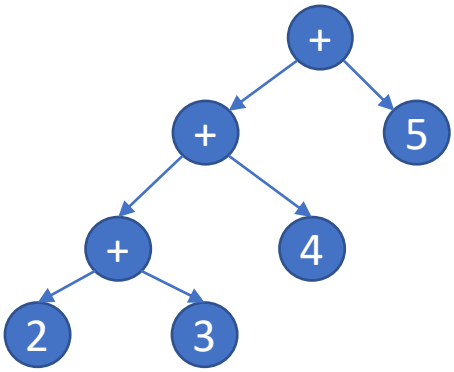


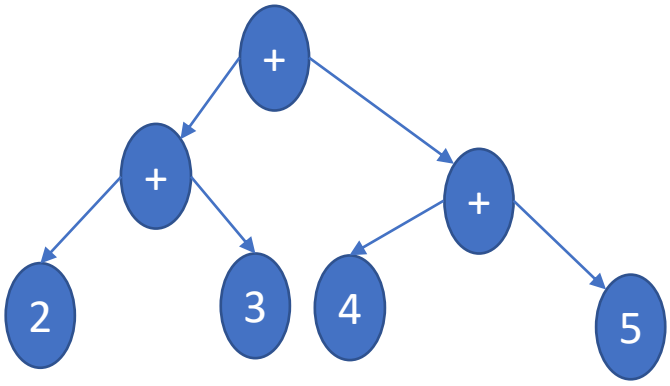
$$2 * 5 + 3 \Rightarrow 13$$





$$2 * (5 + 3) \Rightarrow 16$$





$$2 + 3 + 4 + 5 \Rightarrow 14$$

$$(2 + 3) + (4 + 5) \Rightarrow 14$$

Aunque para nosotros sea la misma operación matemática, son dos árboles completamente diferentes por los paréntesis.

4.2.1 Definir la estructura del nodo

```c
struct nodo{
    int info;
    struct nodo *izq;
    struct nodo *der;
};
typedef struct nodo nodo_t;
```

5. ÁRBOLES MULTICAMINOS