Unidad 4 Estructuras no lineales

4.1 Árboles

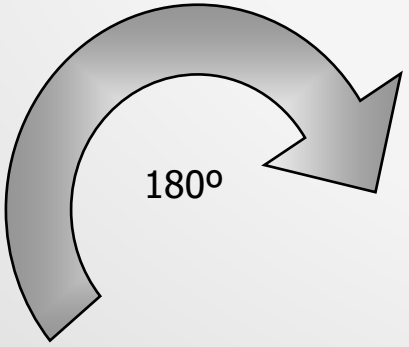
4.2 Grafos

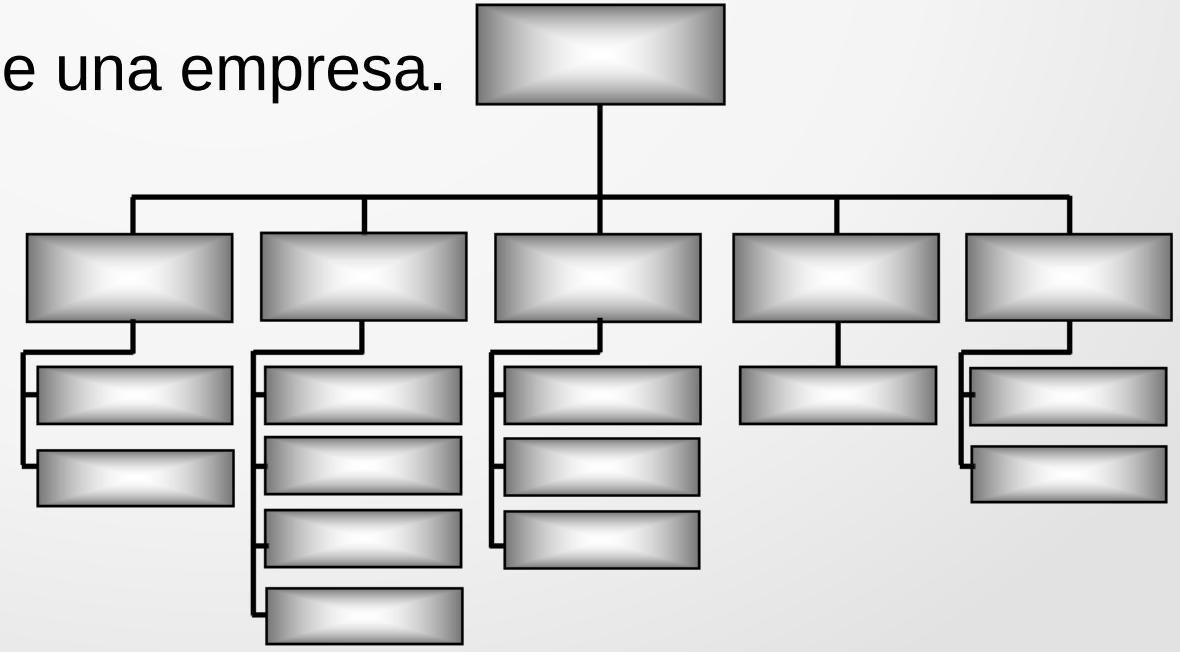
4.1.1 Clasificación de Árboles

Una estructura de árbol es una forma de representar la **JERARQUÍA** de ciertos objetos en una forma gráfica.

Se le llama **árbol** porque la gráfica se asemeja a un árbol natural, aunque, a diferencia de los arboles naturales, estas estructuras jerárquicas crecen hacia abajo, es decir, la raíz se encuentra arriba y las hojas en la parte inferior.

Ejemplo: El organigrama de una empresa.





4.1.1 Clasificación de Árboles

Clasificación de árboles.

En la teoría computacional, los árboles tienen diversos usos.

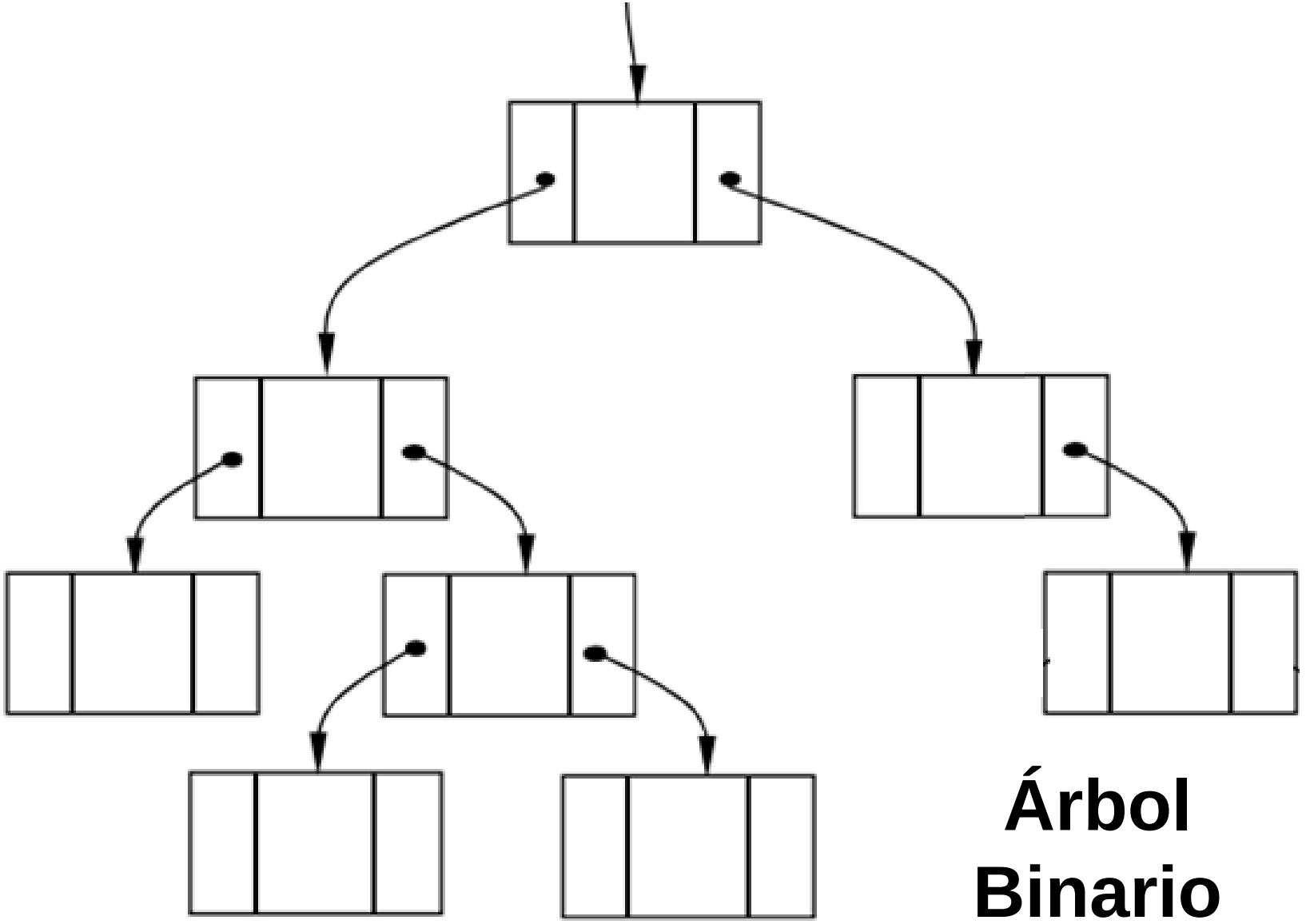
La clasificación que nos interesa, por ahora, es la relativa al número de nodos a que puede apuntar otro nodo y al grado de eficiencia que puede aportar un árbol para la búsqueda de datos almacenados en él.

4.1.1 Clasificación de Árboles

Arboles

- **Binarios**
  Cada nodo solo puede apuntar a otros 2 nodos como máximo
  - **Balanceados**
    Garantiza cierto grado de eficiencia durante su operación
  - **No balanceados**
- **No Binarios**
  Cada nodo puede apuntar a cualquier cantidad de nodos
  - **Balanceados**
    Garantiza cierto grado de eficiencia durante su operación
  - **No balanceados**

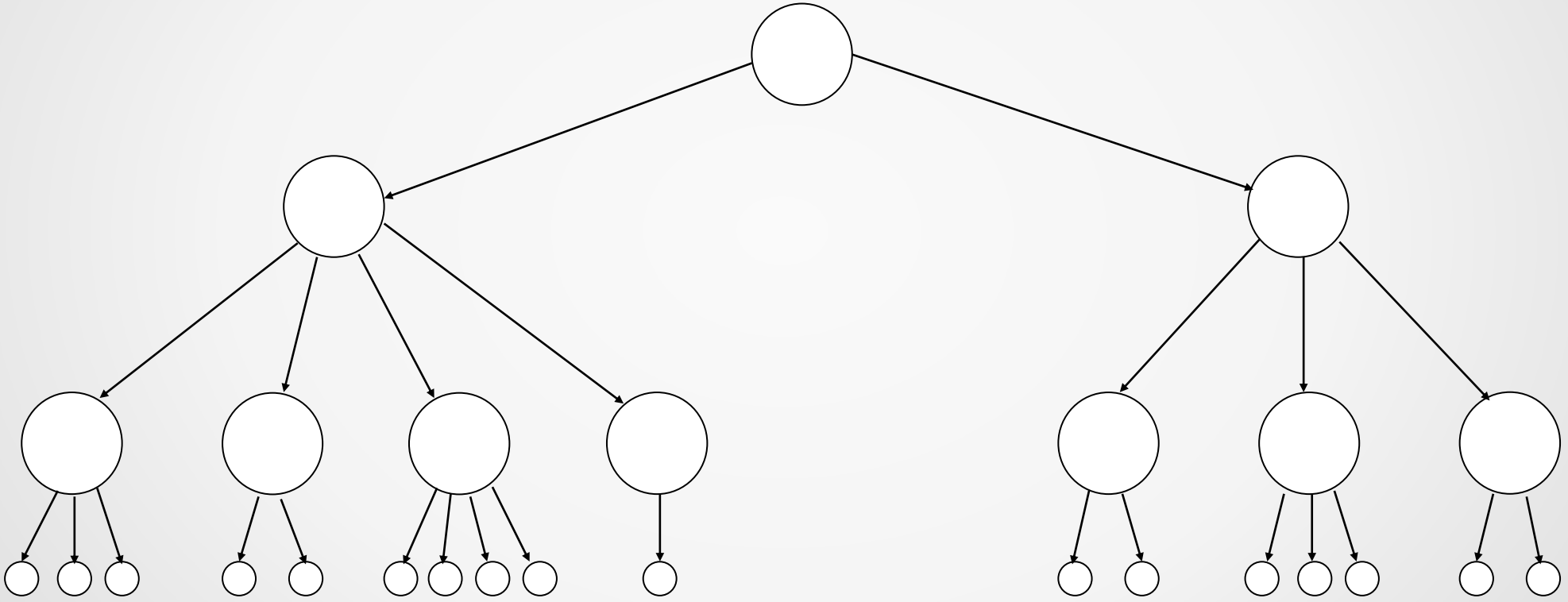
4.1.1 Clasificación de Árboles



Árbol Binario

4.1.1 Clasificación de Árboles

Árbol No Binario



4.1.2 Operaciones Básicas sobre AB

4.2 Arboles Binarios. Operaciones Básicas.

Si se desea localizar cierto dato en una lista lineal ordenada MUY GRANDE, el proceso puede tomar mucho tiempo.

Las estructuras NO LINEALES incluyen más de una ruta para organizar los datos.

Por lo tanto las búsquedas sobre estructuras NO LINEALES requieren menos tiempo.

Los Árboles Binarios son estructuras NO LINEALES, y si se implementan en memoria dinámica, son NO SECUENCIALES.

4.1.2 Operaciones Básicas sobre AB

Una aplicación de los árboles, es la Búsqueda.

Significa que se conservan en los nodos del árbol datos únicos (Llaves de Identificación), para fines, entre otros, de localización rápida.

Los nodos de un Árbol Binario de Búsqueda pueden apuntar a otros dos, uno que le precede en la lista y otro que le sucede.

LISTA ENCADENADA DOBLE ORDENADA

- **Apuntador al Nodo Predecesor Inmediato (El dato menor más cercano de toda la lista).**
- **Apuntador al Nodo Sucesor Inmediato (El dato mayor más cercano de toda la lista).**

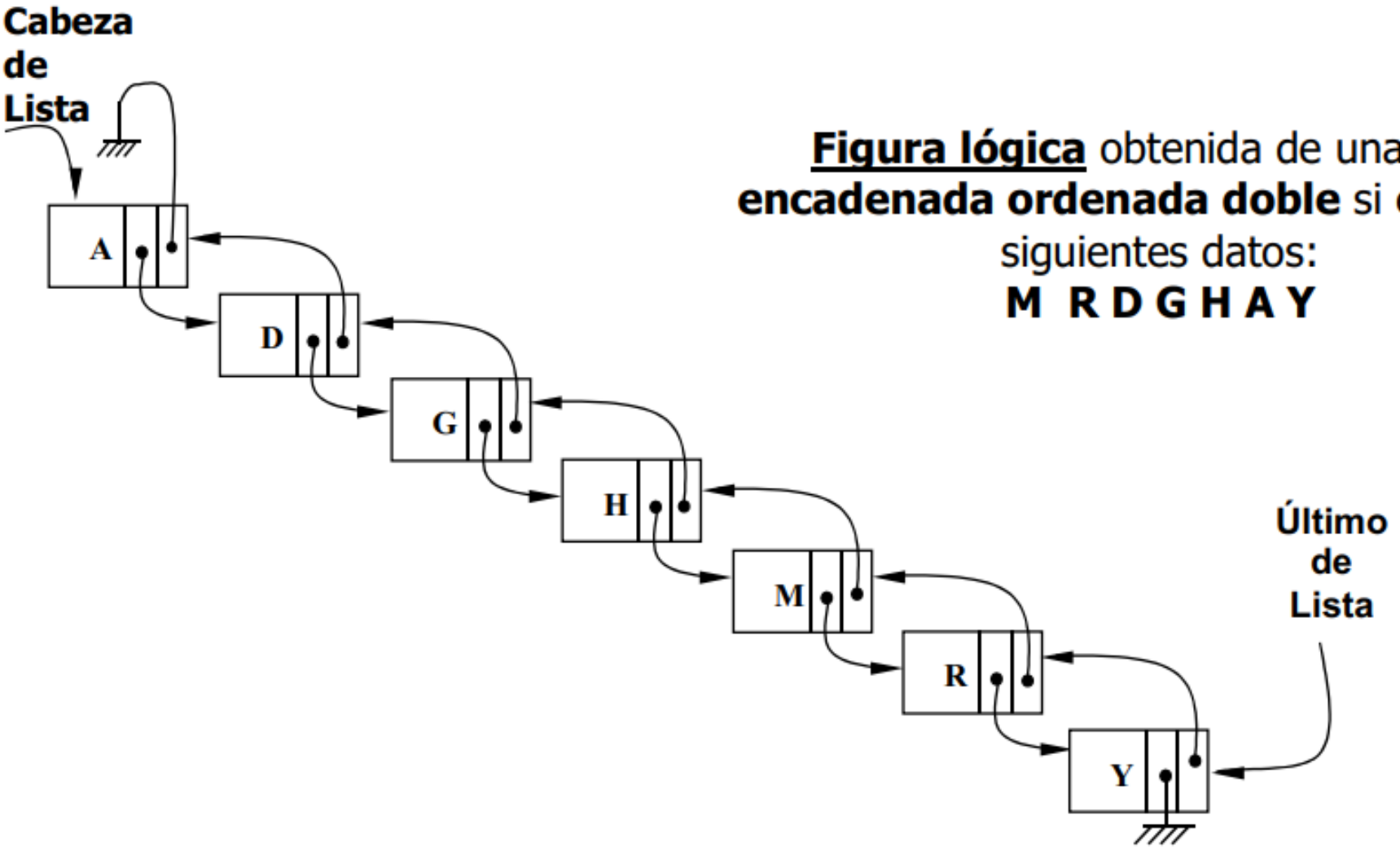
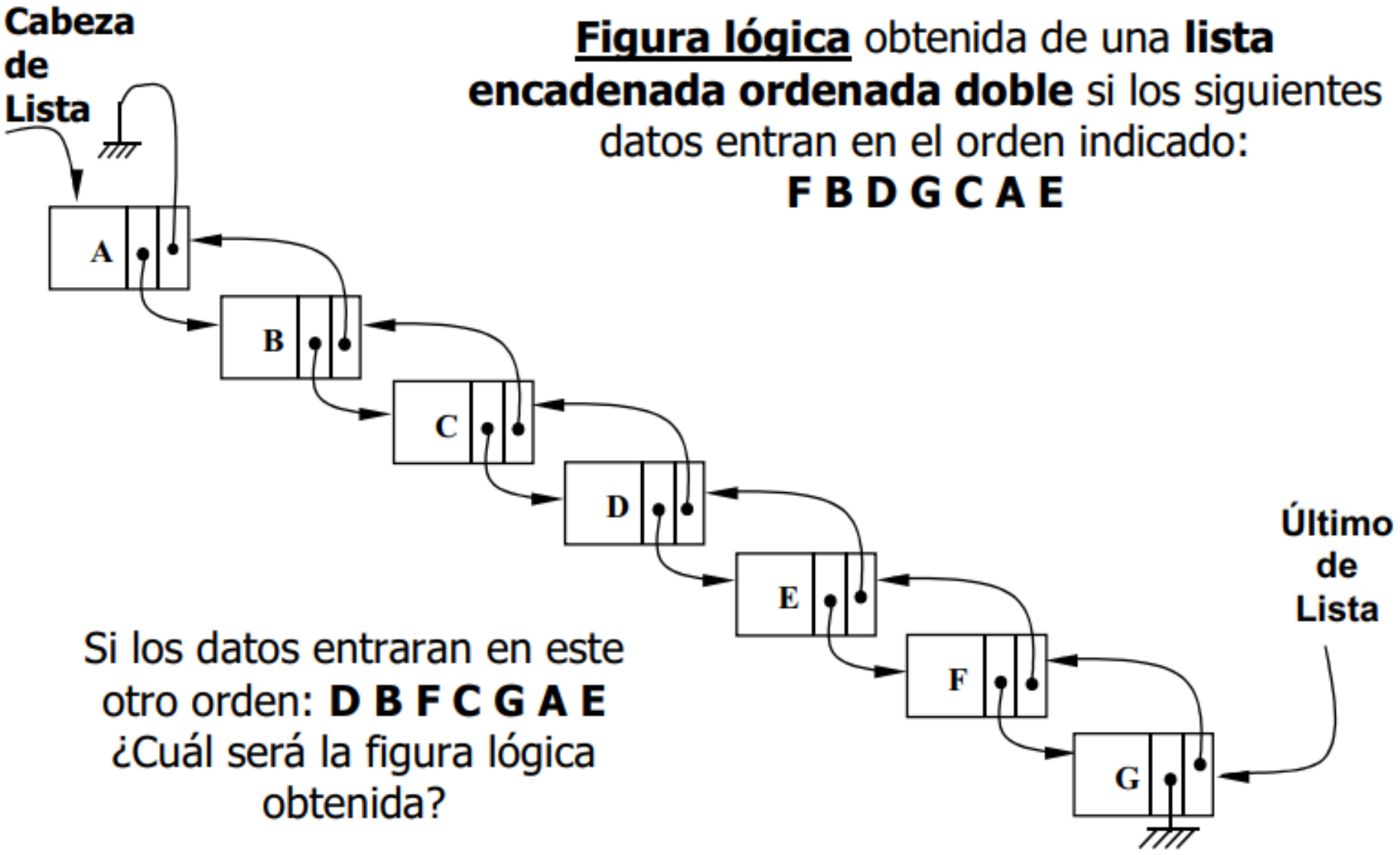


Figura lógica obtenida de una **lista encadenada ordenada doble** si entran los siguientes datos:

M R D G H A Y

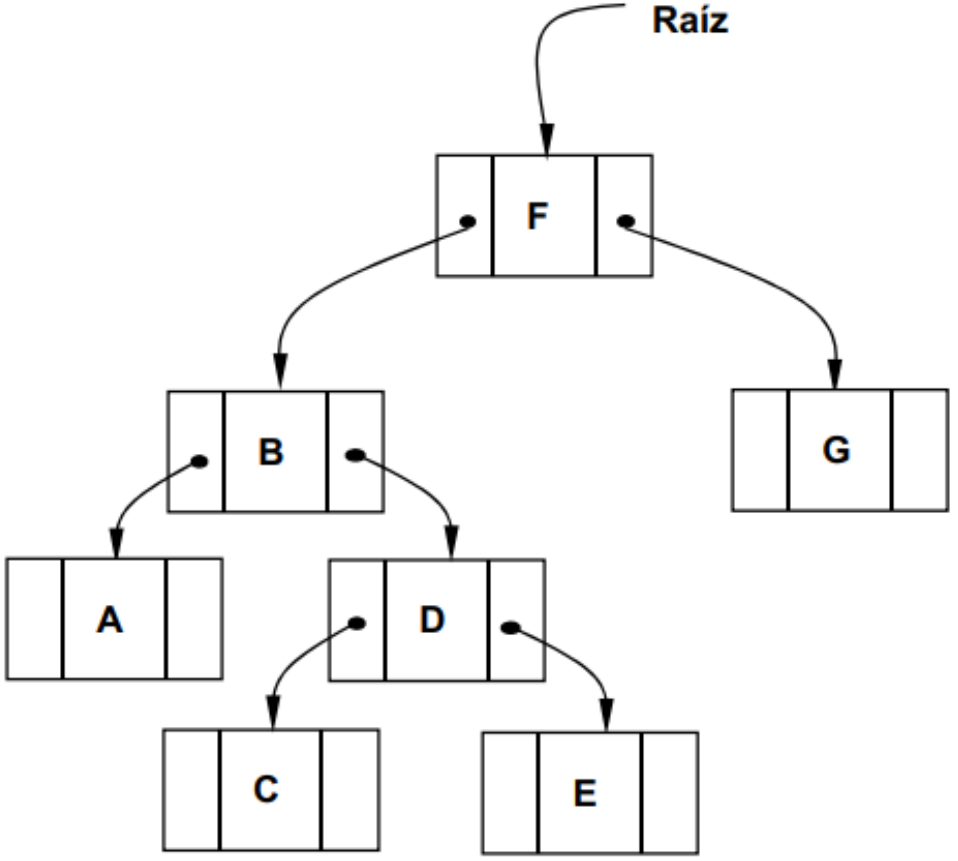
4.1.2 Operaciones Básicas sobre AB



ARBOL BINARIO DE BUSQUEDA

- **Apuntador al Nodo IZQUIERDO (Cualquiera de los datos de la lista que sea Menor).**
- **Apuntador al Nodo DERECHO (Cualquiera de los datos de la lista que sea Mayor).**

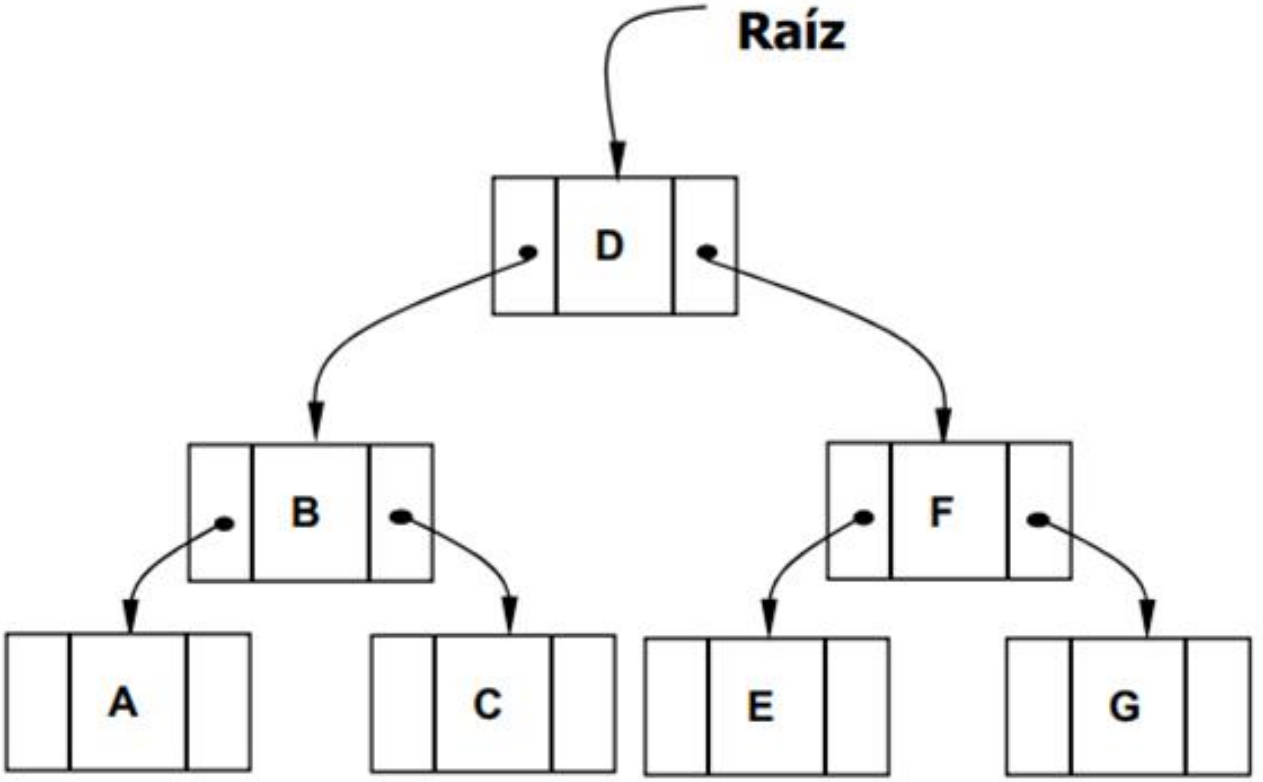
Si los datos **" F B D G C A E "** se guardan en un **árbol binario de BUSQUEDA**, la figura lógica resultante sería como la siguiente:



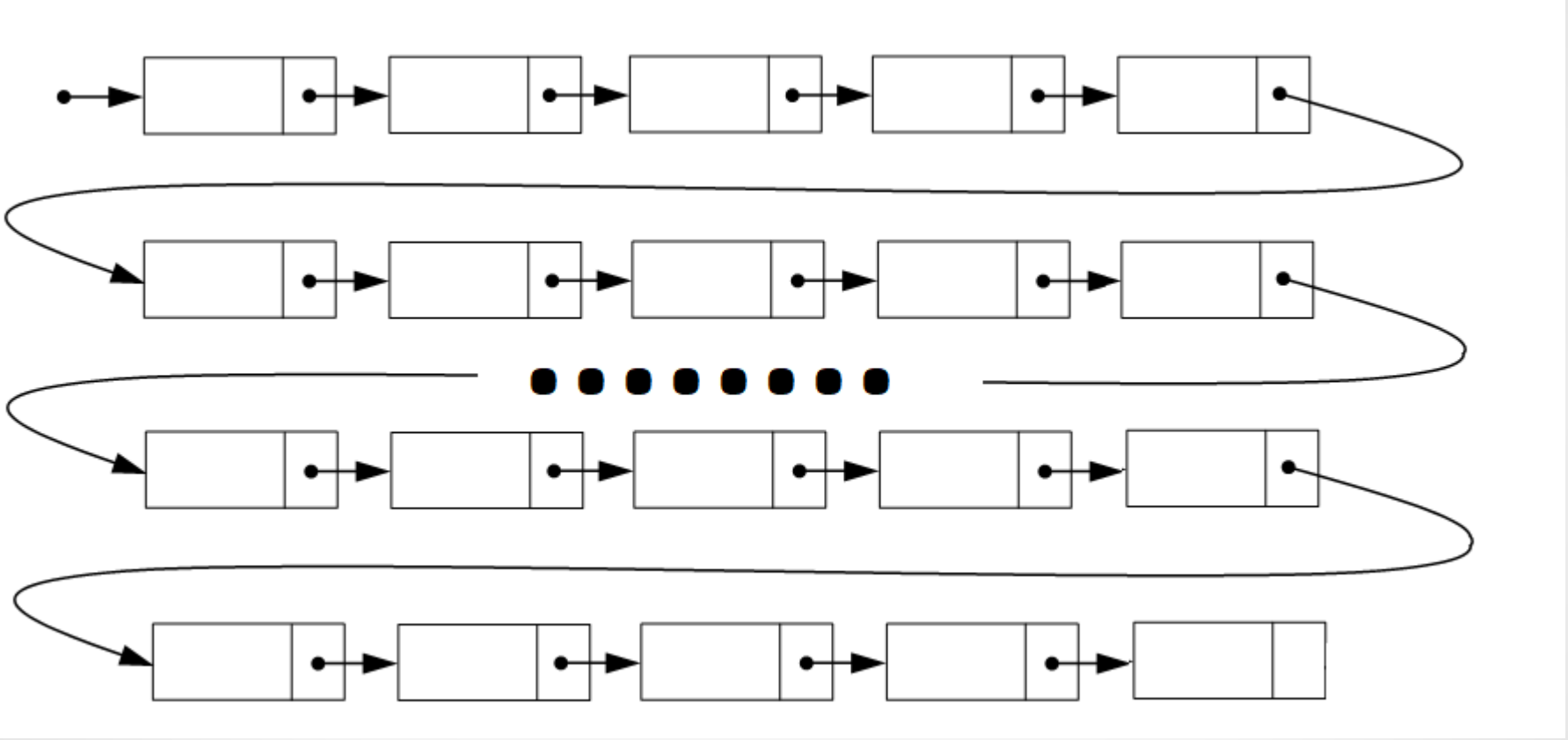
- **Cada nodo apunta a un dato menor por la izquierda.**
- **Cada nodo apunta a un dato mayor por la derecha.**
- **Se requiere un apuntador al primer nodo de la lista (RAIZ).**

4.1.2 Operaciones Básicas sobre AB

Si los datos entrasen en el siguiente orden: " **D B F C G A E** " la forma lógica resultante sería:



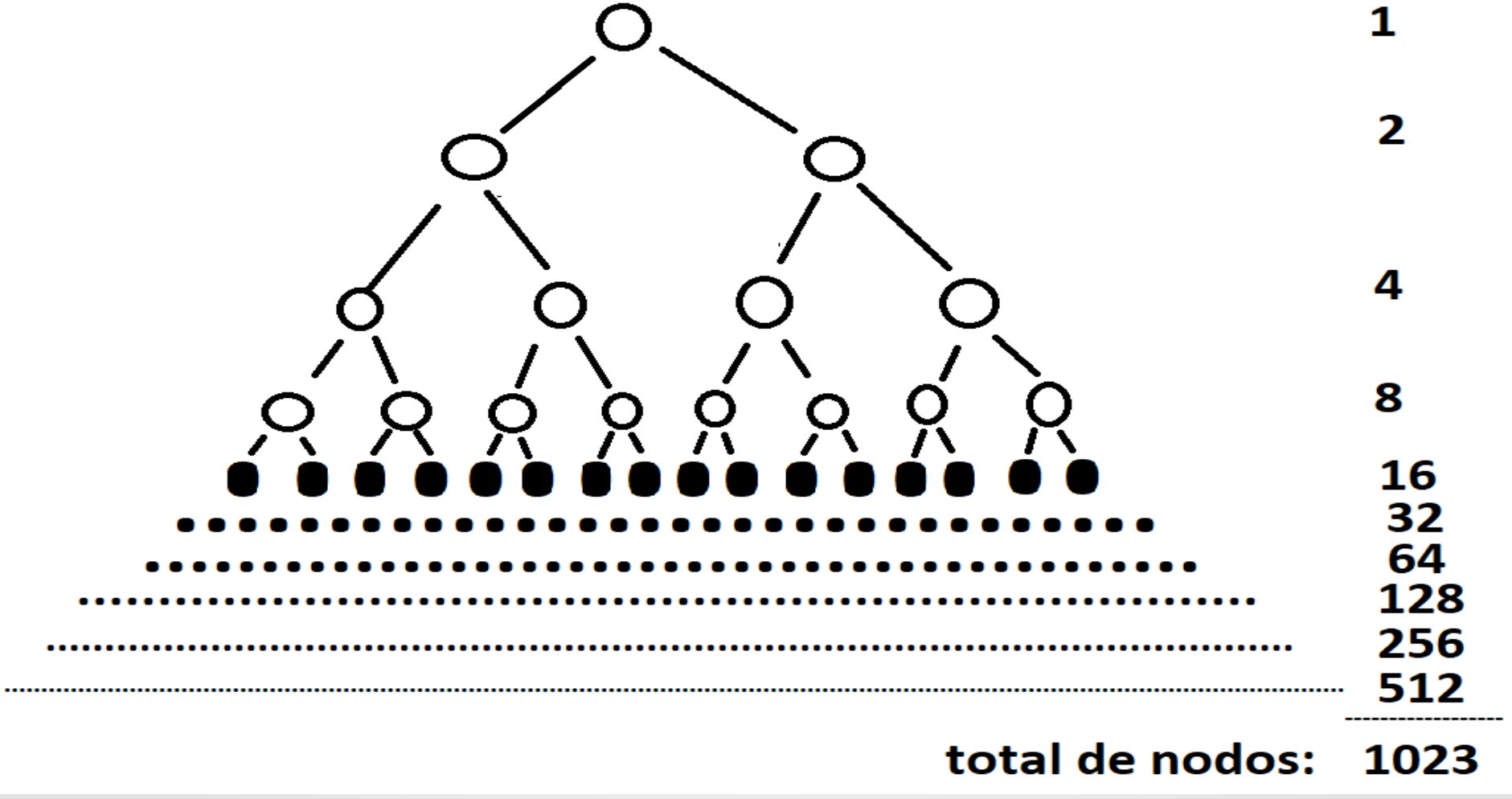
¿Por qué un árbol binario teóricamente es mejor que una lista encadenada?



En una lista encadenada con 1000 nodos **¿cuantas visitas en promedio** se tienen que hacer para localizar cualquiera de los datos?

Respuesta: 500 .

En un árbol binario completo con 1023 nodos, el número máximo de visitas es 10



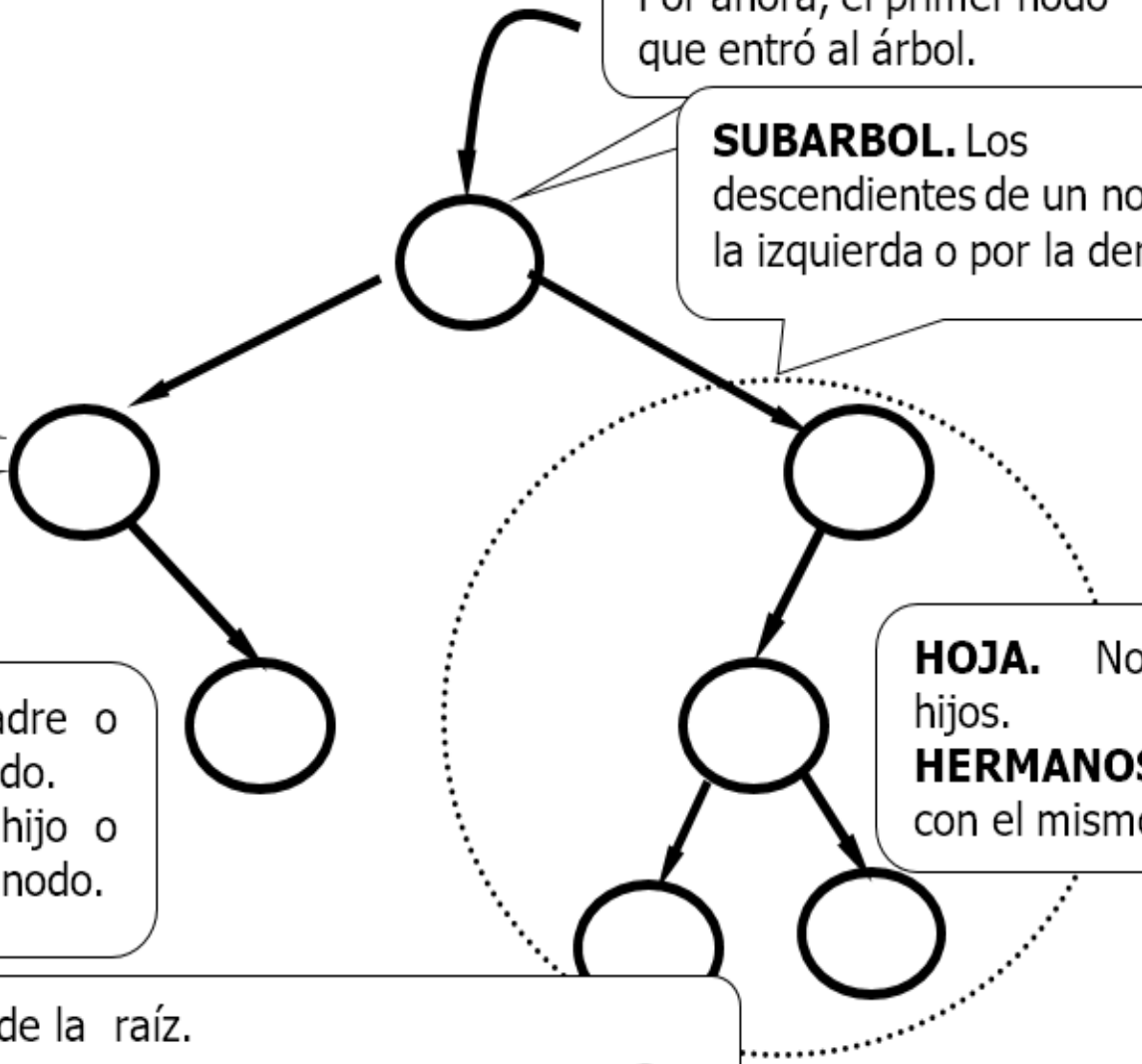
Vocabulario Básico

RAIZ
Por ahora, el primer nodo que entró al árbol.

SUBARBOL. Los descendientes de un nodo por la izquierda o por la derecha.

HIJO
Nodo apuntado por otro. Si está apuntado por la izquierda es hijo izquierdo, si es por la derecha, es hijo derecho.

PADRE
Nodo que apunta a uno o dos nodos.



ANTEPASADO. Nodo que es padre o ANTEPASADO del padre de otro nodo.
DESCENDIENTE. Nodo que es hijo o DESCENDIENTE de un hijo de otro nodo.

HOJA. Nodo sin hijos.
HERMANOS. Nodos con el mismo padre.

NIVEL. Distancia de un nodo desde la raíz.
Nivel de la raíz = 0. Número máximo de nodos en el **nivel "n"** es 2^n.

4.1.2 Operaciones Básicas sobre AB

Creación e Inserción en Arboles Binarios de Búsqueda.

- Todos los árboles inician vacíos.
  - La variable RAIZ debe señalar NULL (Fin de Lista).
- Para añadir nuevos nodos a un ABB hay que hacer una búsqueda del lugar que ocupará cada nodo nuevo.
  - Consiste en mover un apuntador a la izquierda o a la derecha hasta encontrar el lugar.

4.1.2 Operaciones Básicas sobre AB

El algoritmo siguiente es un proceso de búsqueda para localizar cierto valor dentro de un árbol.

En un árbol binario de búsqueda, a la izquierda de cada nodo se encuentra un nodo con valores menores o iguales y a la derecha los mayores.

```
Temp ← RAIZ
MIENTRAS Temp != NULO & INFO(Temp) != VALOR
        SI VALOR > INFO(Temp)
            Temp ← DER(Temp)
        DE LO CONTRARIO
             Temp ← IZQ(Temp)
        FIN SI
FIN MIENTRAS
```

4.1.2 Operaciones Básicas sobre AB

Casos en los que se requiere hacer búsquedas:

1. Para dar de alta un nuevo nodo al arbol.
2. En los nodos se guarda una buena cantidad de información, por lo que se requerirá hacer consultas de los datos.
   - El dato que determina el lugar en el árbol y que por lo tanto se requiere para hacer las consultas, se llama ***llave***.
3. Se puede requerir eliminar cierta información y la búsqueda es indispensable para localizar un nodo donde se encuentra.

**El algoritmo de búsqueda deberá adecuarse según a las características de cada proceso de búsqueda particular*

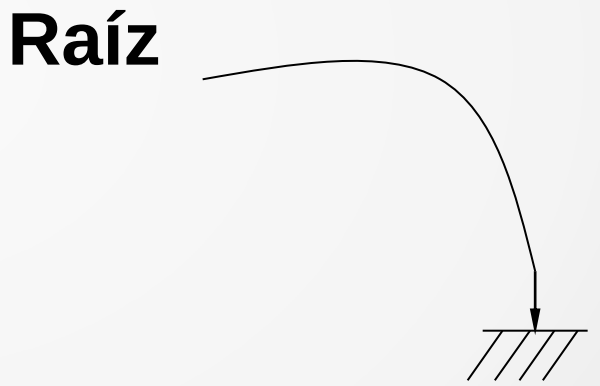
4.1.2 Operaciones Básicas sobre AB

En un ABB cada nuevo nodo se añade como una HOJA, por lo tanto, hay que adecuar el algoritmo de búsqueda que se vio anteriormente para llegar a un nodo con un NULO en uno de los campos de apuntador mientras se viaja por él.

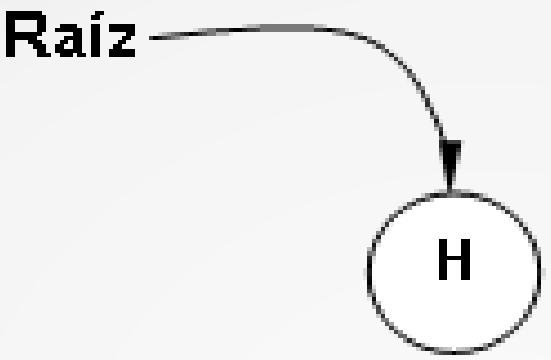
Antes de analizar los algoritmos de creación, primero se debe aprender a crear ABB. Se ejercitará con las llaves siguientes en el orden indicado: H Q M K A F O.

4.1.2 Operaciones Básicas sobre AB

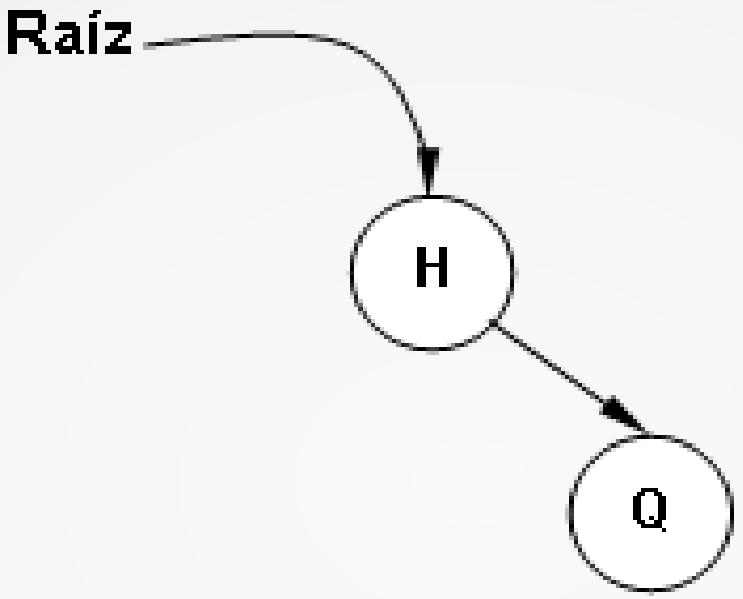
Estado Inicial



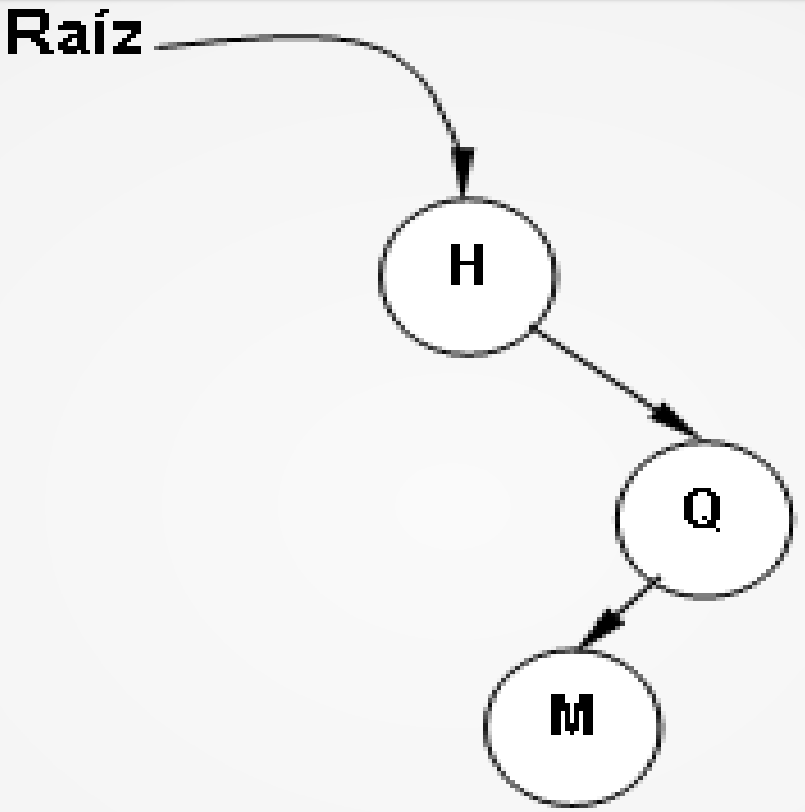
4.1.2 Operaciones Básicas sobre AB



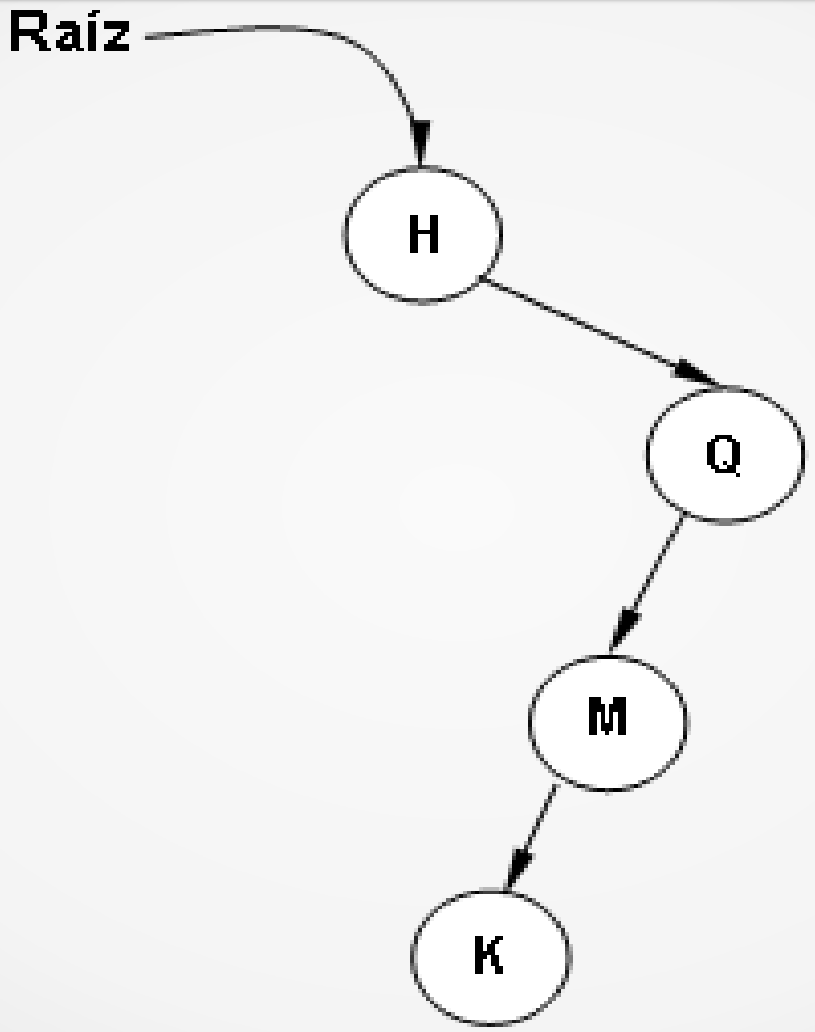
4.1.2 Operaciones Básicas sobre AB



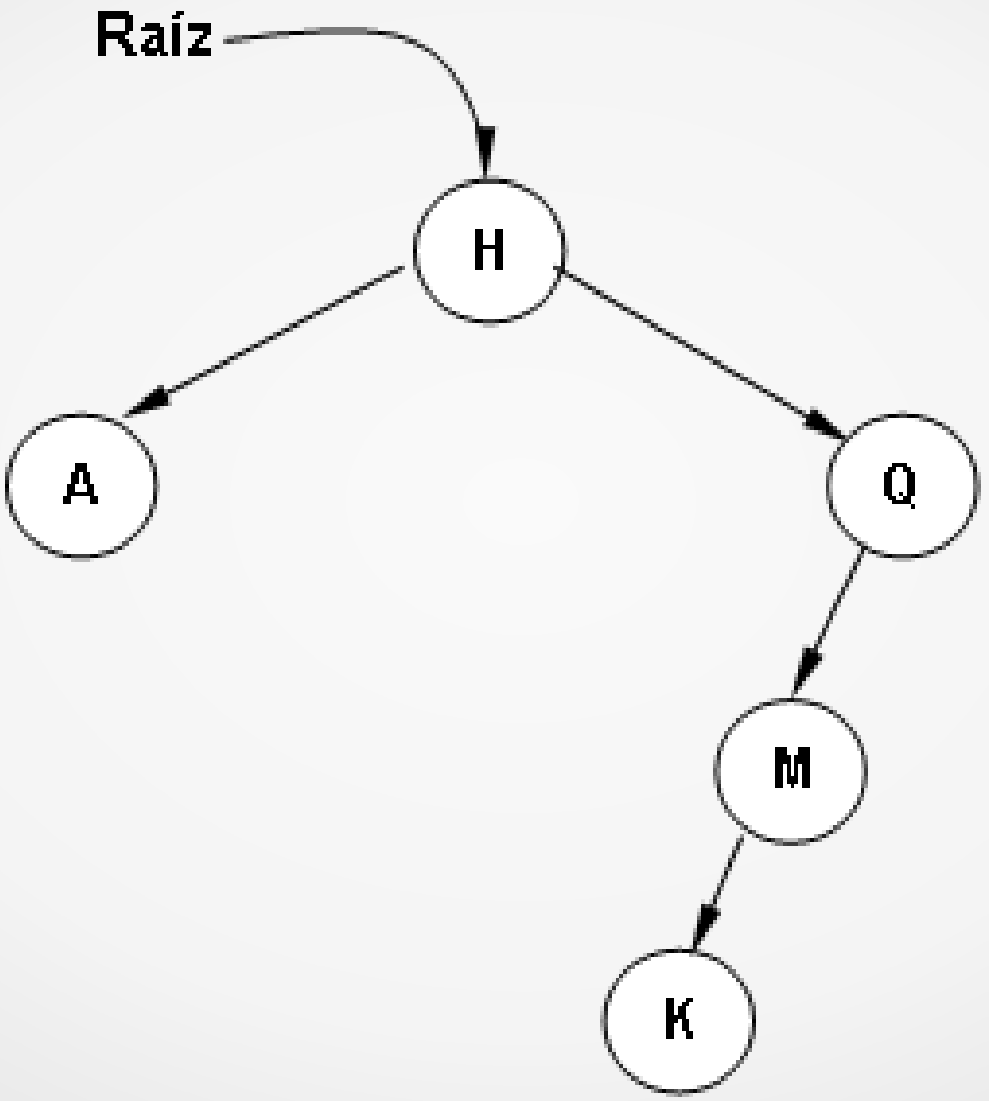
4.1.2 Operaciones Básicas sobre AB



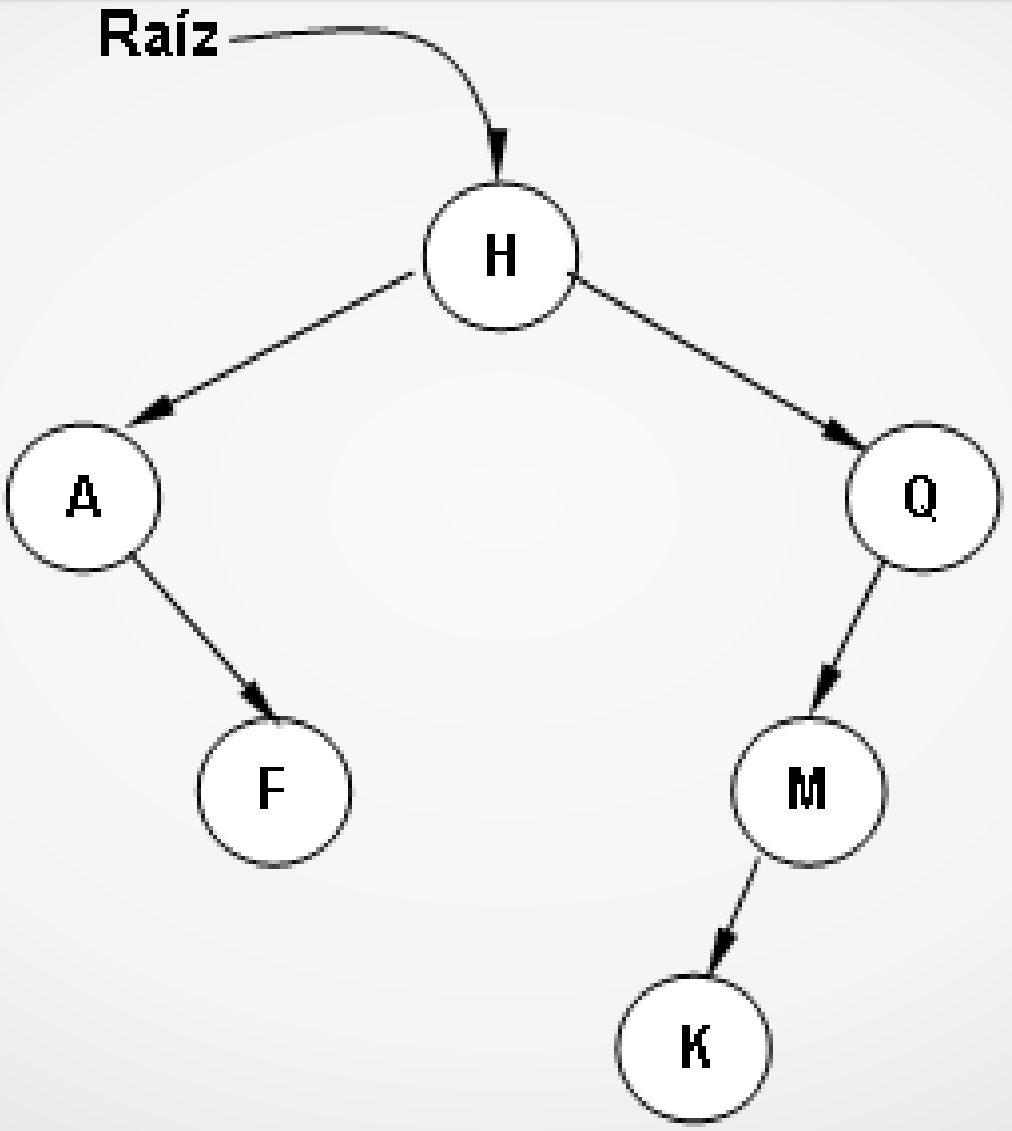
4.1.2 Operaciones Básicas sobre AB



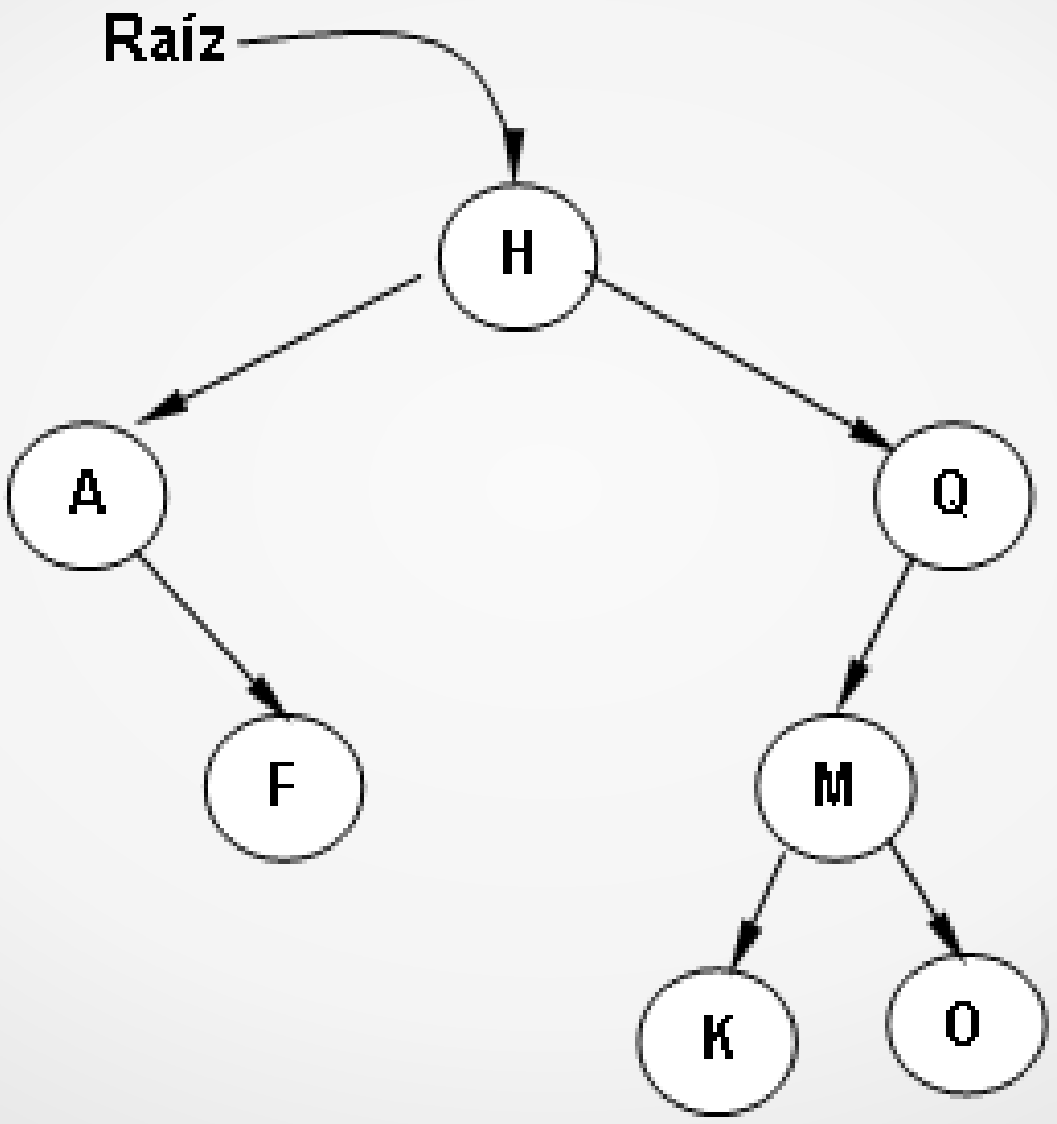
4.1.2 Operaciones Básicas sobre AB



4.1.2 Operaciones Básicas sobre AB



4.1.2 Operaciones Básicas sobre AB



4.1.2 Operaciones Básicas sobre AB

Ejercicio 1: Construir un árbol a partir de los siguientes datos en el orden indicado:

4.1.2 Operaciones Básicas sobre AB

Ejercicio 2: Construir un árbol binario de búsqueda a partir de los siguientes datos en el orden indicado:

J	R	M	K
W	Z	T	L
Q	S	P	X
G	A	C	H

4.1.2 Operaciones Básicas sobre AB

Algoritmos iterativos para añadir datos a un Árbol Binario de Búsqueda.

Procesos a efectuar durante altas de datos a un ABB:

1. Crear un nodo para el nuevo dato.
2. Buscar el lugar donde insertarlo.
3. Colocar el apuntador al nuevo nodo.

4.1.2 Operaciones Básicas sobre AB

1

NuevoNodo ← Dirección
otorgada por
el Sistema Operativo

IZQ(NuevoNodo) ← NULO

DER(NuevoNodo) ← NULO

INFO(NuevoNodo) ← VALOR

4.1.2 Operaciones Básicas sobre AB

2

```
Temp ← RAIZ
PadreNN ← NULO
MIENTRAS Temp != NULO
        PadreNN ← Temp
        SI INFO(NuevoNodo) > INFO(Temp)
                Temp ← DER(Temp)
        DE LO CONTRARIO
                Temp ← IZQ(Temp)
        FIN SI
FIN MIENTRAS
```

*PadreNN=Padre del Nuevo Nodo

4.1.2 Operaciones Básicas sobre AB

3

```
SI PadreNN = NULO
        RAIZ ← NuevoNodo
DE LO CONTRARIO
        SI INFO(NuevoNodo) > INFO(PadreNN)
                DER(PadreNN) ← NuevoNodo
        DE LO CONTRARIO
                IZQ(PadreNN)  ← NuevoNodo
        FIN SI
FIN SI
```

// Ejemplo Árbol Binario de Búsqueda EjemploABB01.cpp

```
#include <iostream>
#include <cstring>
#include <conio.h>
using namespace std;
#include "ClaseABB.h"
int main()
{
        char opcion;
        string dato;
        ABB arbol;
        do {
            cout << "1.- Altas\n" << "2.- Consultar\n";
            cout << "0.- Terminar\n\n";
            cout << "Opcion: ";opcion = getch(); cout << opcion;
            cout << endl;
            switch (opcion){
            case '1':
                 cout<<"Nuevo Dato: ";getline(cin,dato);
                 arbol.Anadir(dato);
                 break;
            case '2':
                 cout<<"Dato a buscar: ";getline(cin,dato);
                 arbol.Consultar(dato);
                 break;
            }
        }
        while (opcion!='0');
        system("pause");
        return 0;
}
```

// ClaseABB.h

```
class ABB{
    private:
            struct Nodo {
                Nodo* izq;
                string info;
                Nodo* der;
            };
            Nodo* raiz;

    public:
            ABB(); // Constructor
            void Anadir(string);
            void Consultar(string);
            // ~ABB(); Pendiente el destructor ()
};

ABB::ABB(){
    raiz=NULL;
};

void ABB::Anadir(string x){
    Nodo *nn,*temp,*pnn;
    // paso 1
    nn = new Nodo;
    nn->info = x;
    nn->izq = NULL;
    nn->der = NULL;
    // paso 2
    temp = raiz;
    pnn  = NULL;
    while (temp!=NULL){
        pnn = temp;
        if (nn->info>temp->info)
           temp = temp->der;
        else
           temp = temp->izq;
    };
    // paso 3
    if (pnn==NULL)
       raiz = nn;
    else
       if (nn->info > pnn->info)
          pnn->der = nn;
       else
          pnn->izq = nn;
    };
```

```
void ABB::Consultar(string x){
    Nodo *p,*padre;
    // Se busca el valor
    p = raiz;
    padre  = NULL;
    while ((p!=NULL) and (p->info != x)){
        padre = p;
        if (x>p->info)
           p = p->der;
        else
           p = p->izq;
    };

    if (p==NULL)
        cout << "[" << x << "] no existe" << endl;
    else
        if (padre==NULL)
            cout << "[" << x << "] es la raiz" << endl;
        else{
            cout << "El padre de " << "[" << x << "] es: "
                 << padre->info << endl;
            if ((p->izq==NULL) and (p->der==NULL))
               cout << "[" << x << "] no tiene hijos" << endl;
            else if ((p->izq!=NULL) and (p->der!=NULL))
               cout << "[" << x << "] tiene DOS hijos" << endl;
            else
               cout << "[" << x << "] tiene UN hijo" << endl;
            system("pause");
            }

}
```

4.1.2 Operaciones Básicas sobre AB

EJERCICIO 1:

Modificar la Clase ABB para que:

- En la opción “consulta”, indique si el nodo encontrado es hijo izquierdo o derecho de su padre.
- Indique el nivel del nodo.
- Además, debe mostrarse que valores contiene cada uno de los hijos que en todo caso tiene.

Pruebe la URL siguiente para añadir nodos a un ABB
https://www.cs.usfca.edu/~galles/visualization/BST.html

4.1.2 Operaciones Básicas sobre AB

EJERCICIO 2:

Escriba un programa, usando manejo dinámico de memoria, que guarde en cada nodo de un árbol, el Número de Control y el nombre de un alumno, usando el siguiente algoritmo:

Mostrar un menú con las siguientes opciones (apéguese al algoritmo descrito):

4.1.2 Operaciones Básicas sobre AB

1.- Altas

- Pedir al usuario que escriba el Número de Control de un alumno.
- En caso de que exista en el árbol ese Número de Control, el programa debe mostrar un mensaje indicando que no se puede dar de alta de nuevo (se debe mostrar el nombre de la persona a la que corresponde ese número de control).
- Si no existe, el programa pedirá al usuario que escriba el nombre del alumno que se va a dar de alta.
- Ya con ambos datos, se busca el lugar adecuado para el nuevo nodo en el árbol (de acuerdo a los algoritmos estudiados) y se enlaza.

4.1.2 Operaciones Básicas sobre AB

2.- Consultas

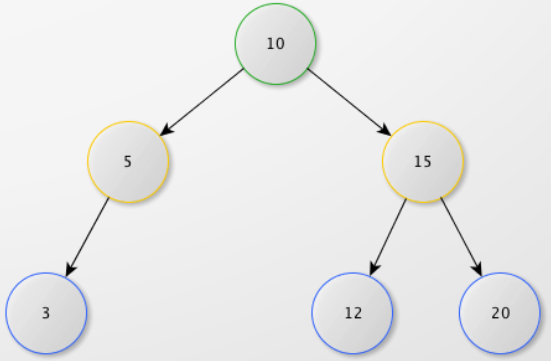
- Pedir al usuario que escriba el Número de Control de un alumno.
- Se buscará dentro del árbol ese Número de Control.
- En caso de que no exista en el árbol, se indicará con un mensaje.
- En caso de que se encuentre, el programa debe mostrar el nombre del alumno al que corresponde el Número de Control.

3.- Terminar

4.1.2 Operaciones Básicas sobre AB

EJERCICIO 3:

- Escriba un programa, para crear un árbol binario, usando manejo dinámico de memoria.
- El tipo de datos que guardará es ***int***.
- Debe mostrar un menú con las siguientes opciones:
  1. Añadir dato.
  2. Mostrar el árbol.
  3. Salir



4.1.2 Operaciones Básicas sobre AB

Eliminación en Arboles Binarios de Búsqueda

Es evidente que es más complejo eliminar un nodo con 2 hijos que una hoja

Por lo que conviene desglosar el problema en 3 casos:

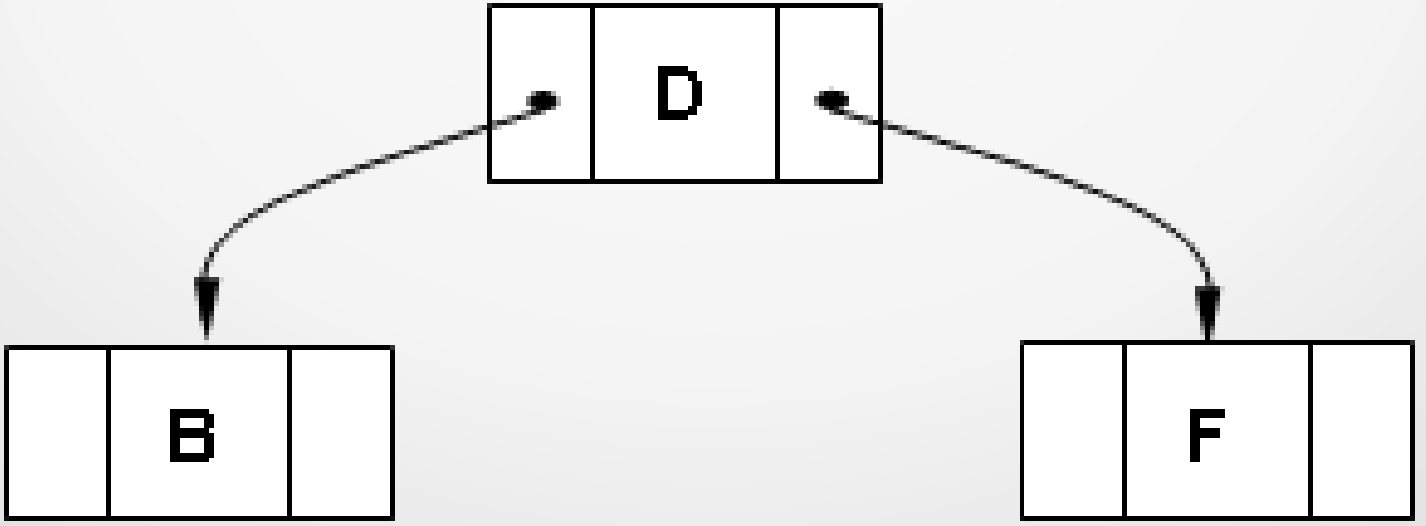
1. Eliminar una hoja
2. Eliminar un nodo que tiene solo un hijo
3. Eliminar un nodo que tiene 2 hijos

4.1.2 Operaciones Básicas sobre AB

1. Supresión de una hoja

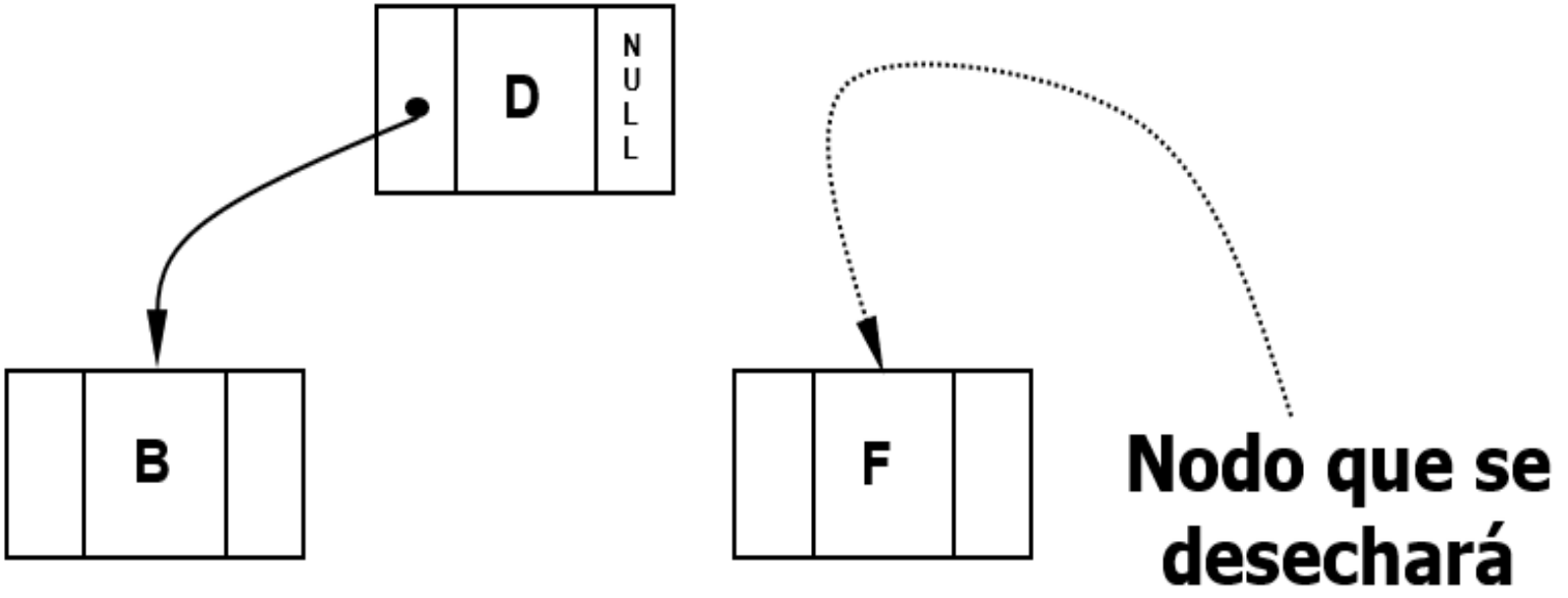
Procedimiento muy simple.
Hay que colocar el apuntador de su padre en FIN DE LISTA.
Deshacerse del nodo ya eliminado.

Estado original de cierto ARBOL o SUB-ARBOL:



4.1.2 Operaciones Básicas sobre AB

Al eliminar la F, el sub-árbol resultante será:

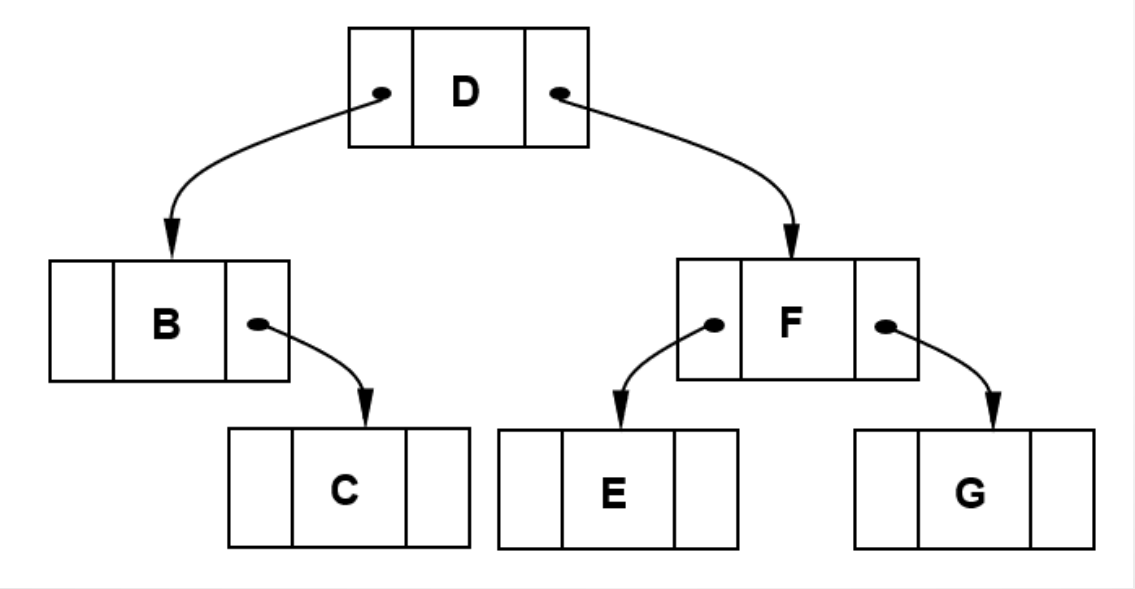


4.1.2 Operaciones Básicas sobre AB

2. Supresión de un nodo con un hijo

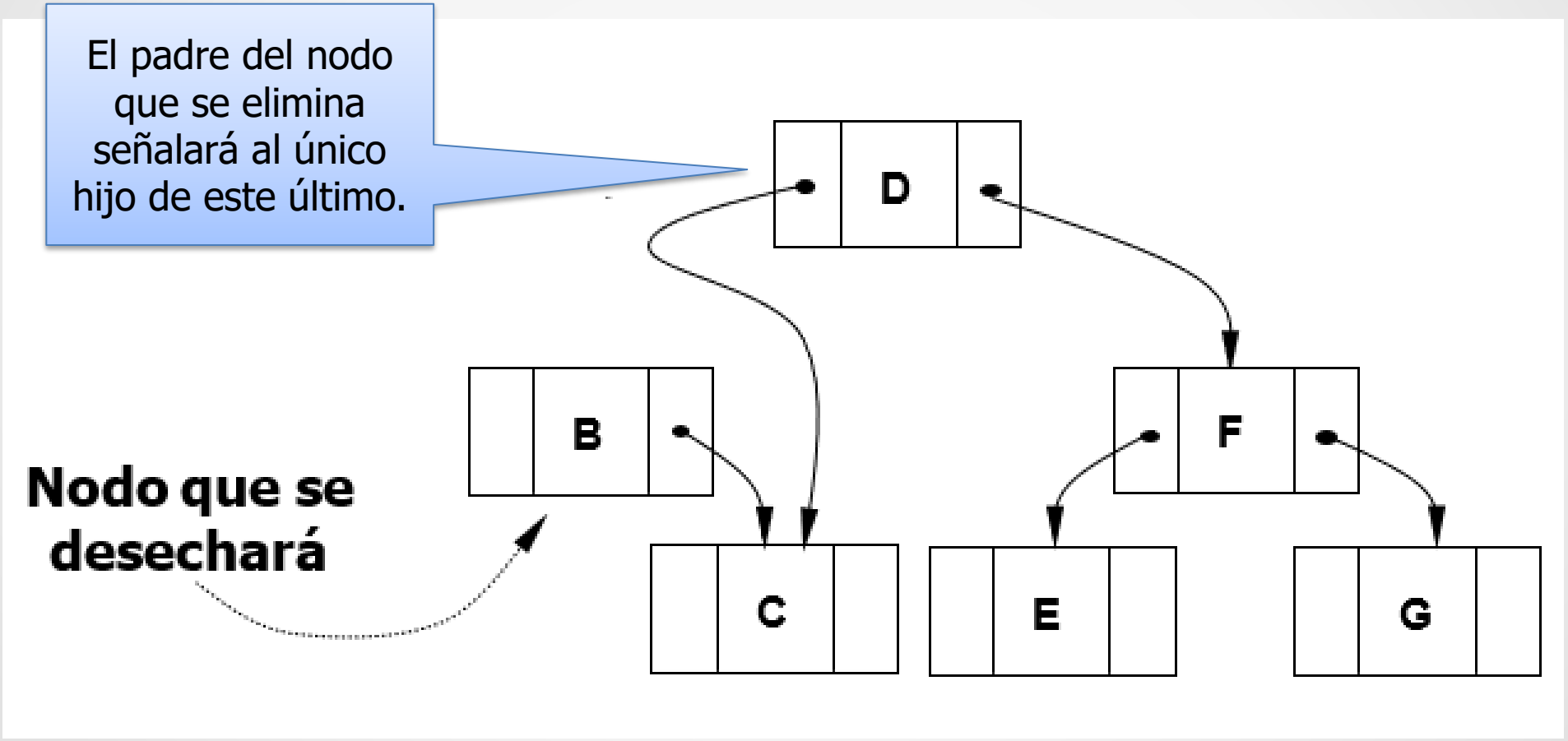
No se puede usar el algoritmo de la página anterior porque se perderían los descendientes del nodo que se elimina.

Estado original de cierto ARBOL o SUB-ARBOL:



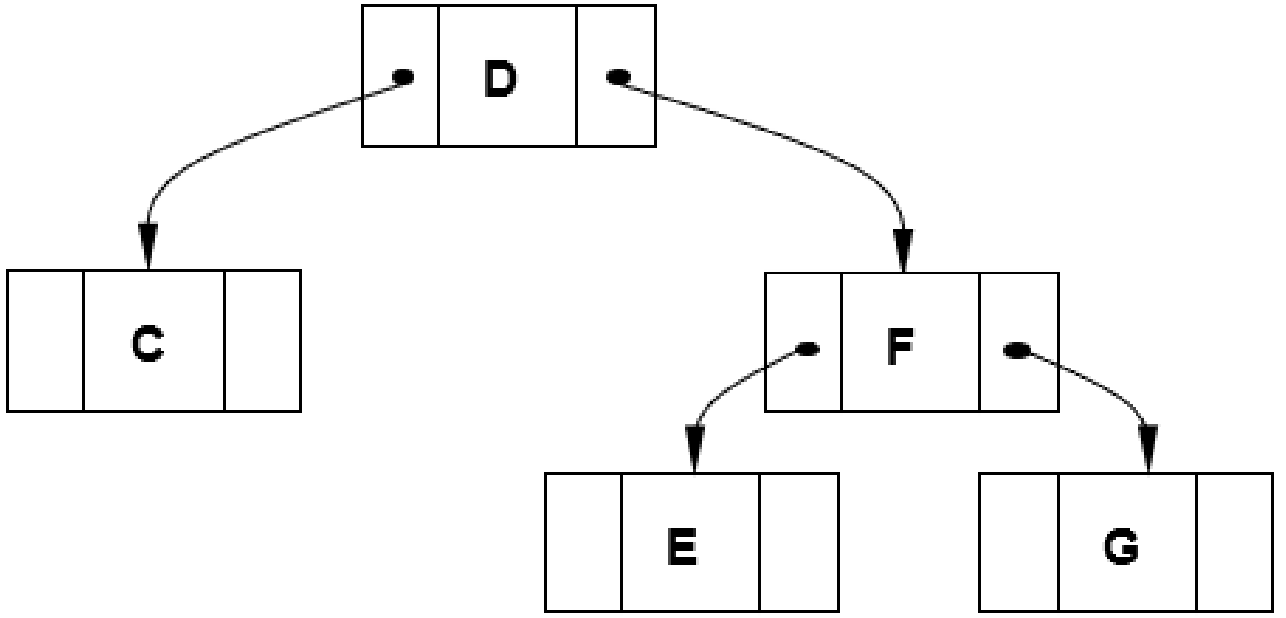
¿Cómo quedará el árbol al eliminar la B?

4.1.2 Operaciones Básicas sobre AB



4.1.2 Operaciones Básicas sobre AB

El sub-árbol con la raíz C sube un nivel.



4.1.2 Operaciones Básicas sobre AB

3. Supresión de un nodo con dos hijos

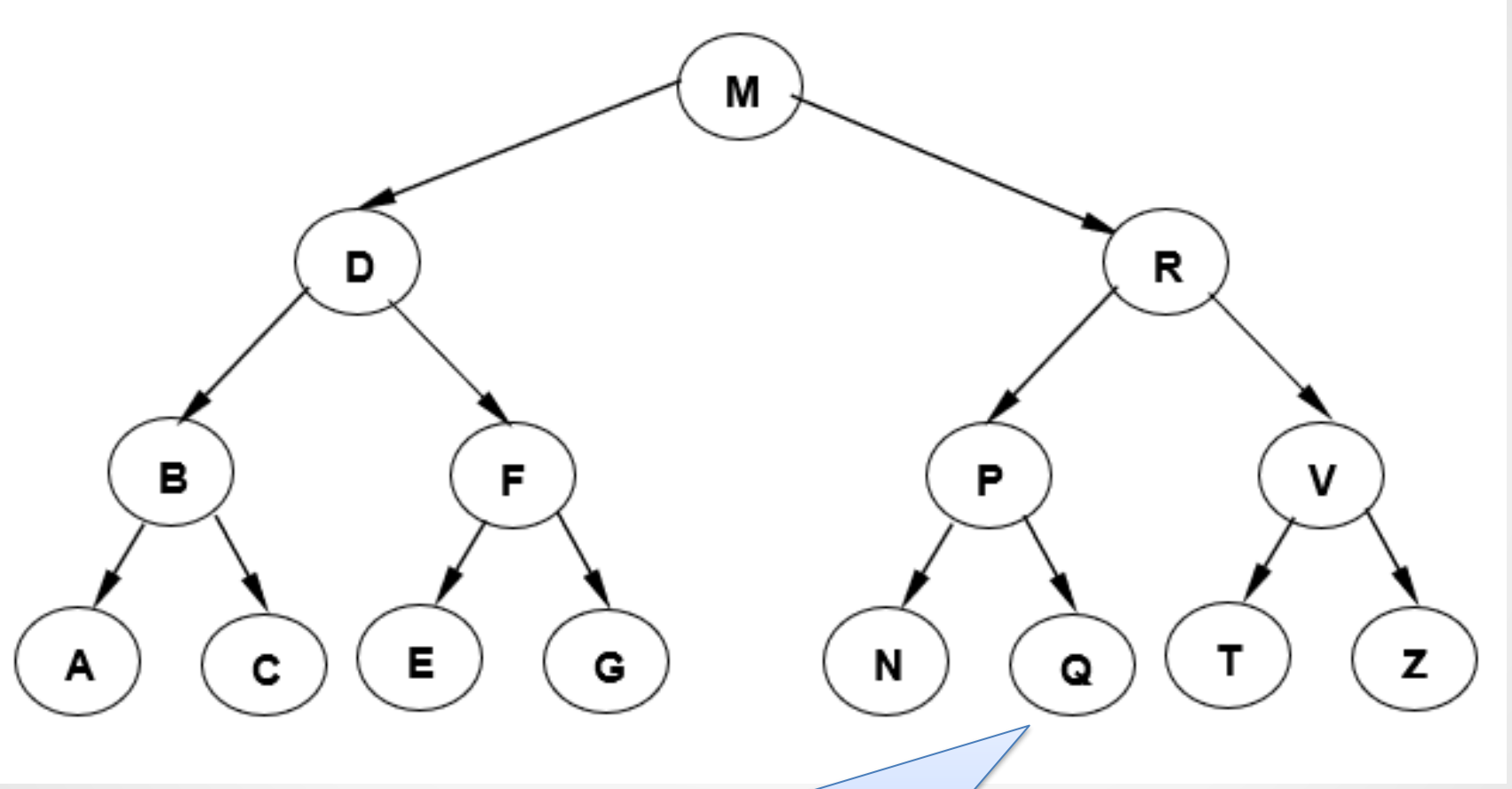
- **Caso más complejo que el anterior porque no se puede hacer que un nodo apunte a otros dos por la izquierda o la derecha.**
- **El método más común para resolver este caso es sustituir información.**
- **Se reemplazará el dato que se quiere eliminar con el dato más cercano del subárbol izquierdo (equivalente al predecesor inmediato).**

¿Cómo se encuentra el predecesor inmediato?

Se viaja (a partir del nodo a eliminar) una vez a la izquierda y tantas veces como se pueda hacia la derecha.

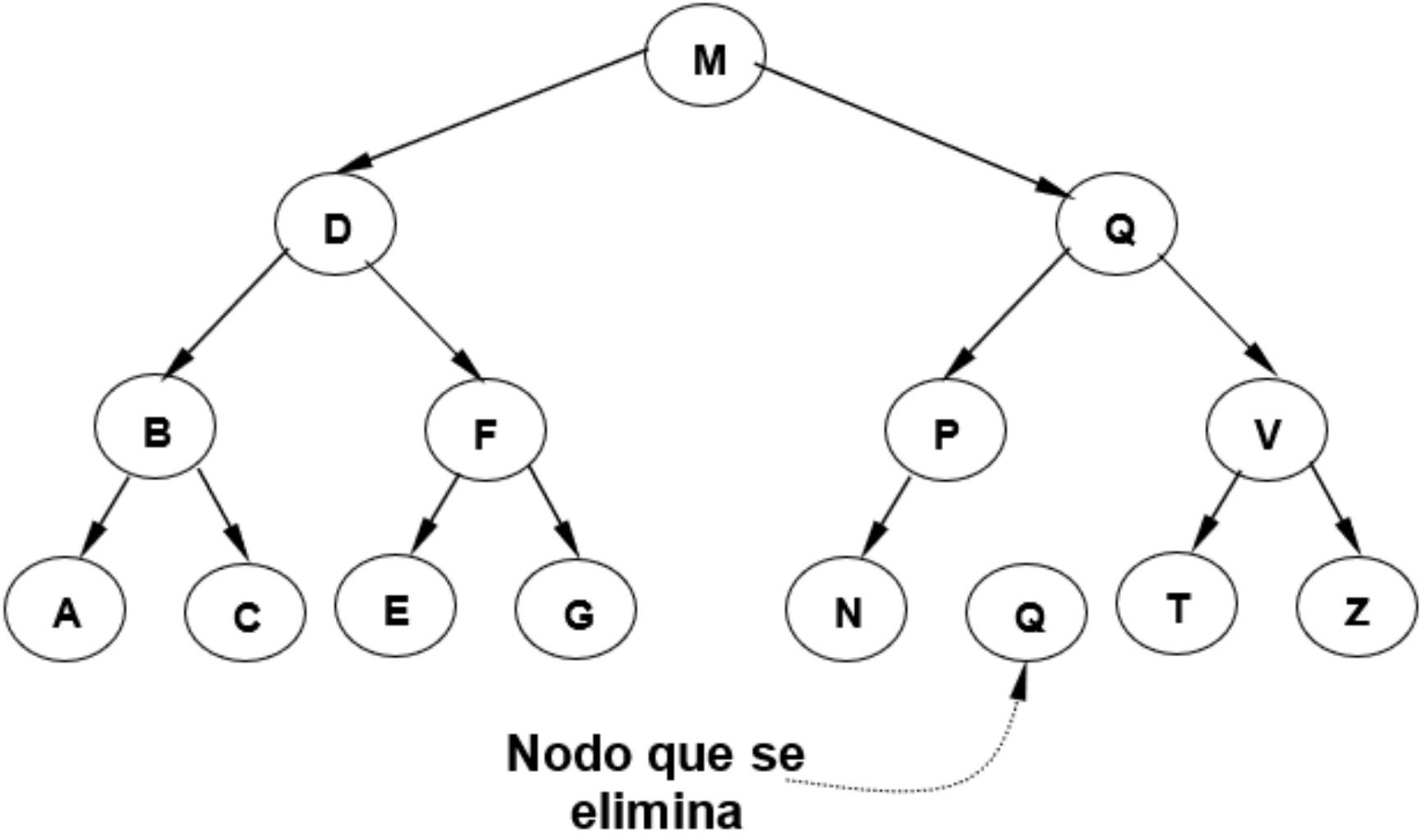
Ya encontrado el dato sustituto, se coloca en el lugar del nodo a eliminar y el nodo del dato sustituto se elimina (solo tendrá un hijo como máximo).

4.1.2 Operaciones Básicas sobre AB

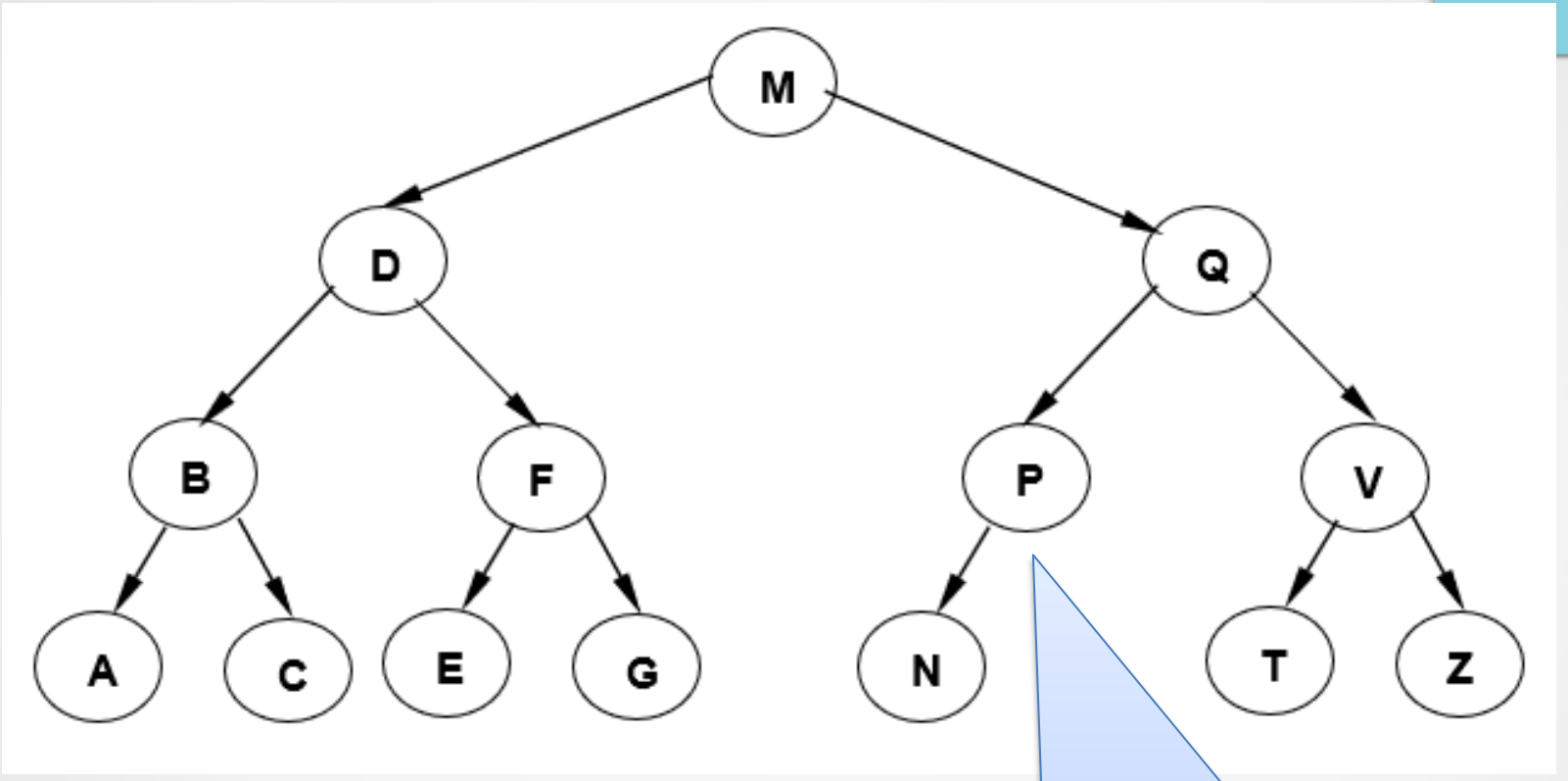


Si eliminamos la R, se reemplazará la información del nodo que contiene la **R** por el predecesor inmediato, es decir la **Q**, luego el nodo que contenía la Q deberá ser eliminado

4.1.2 Operaciones Básicas sobre AB

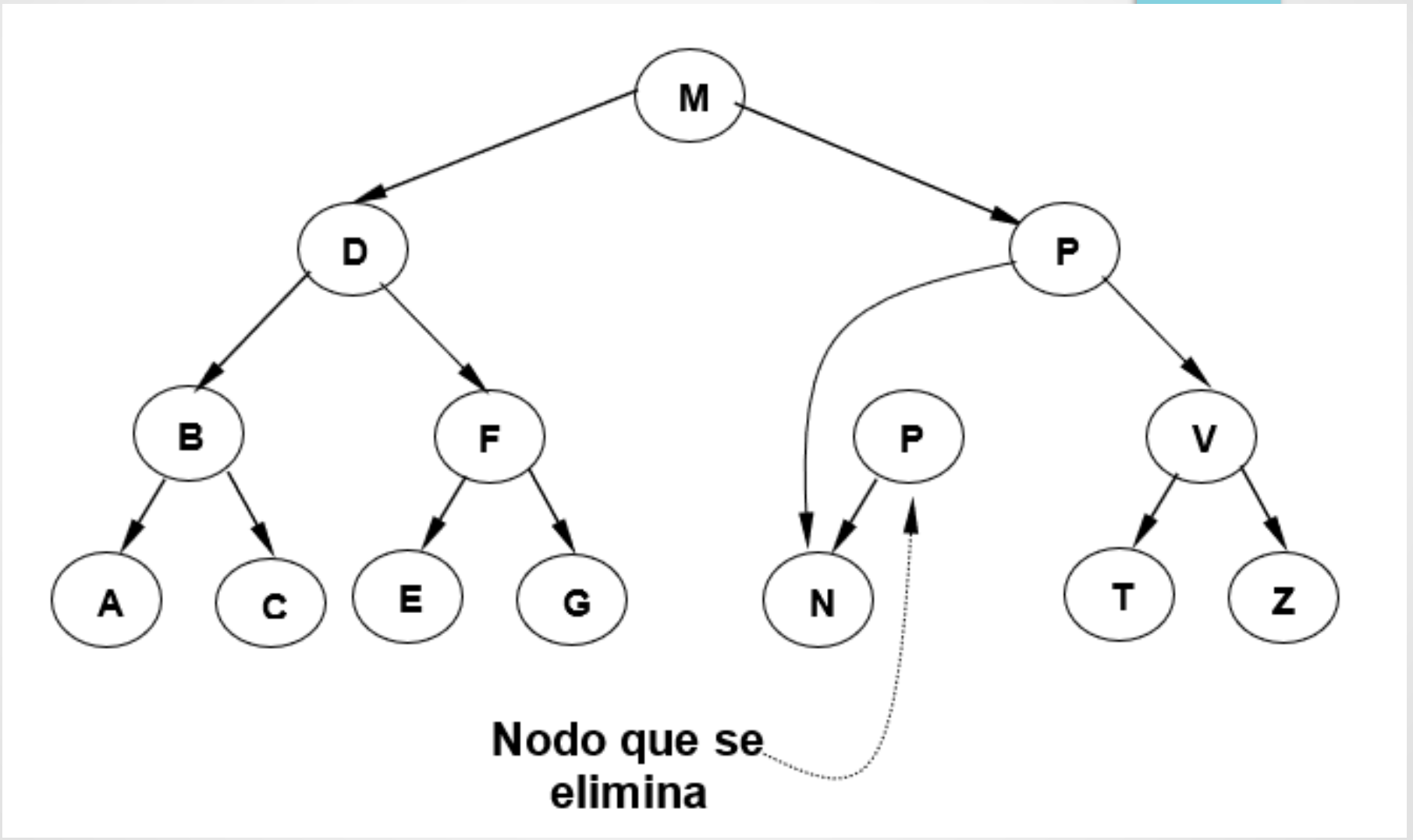


4.1.2 Operaciones Básicas sobre AB

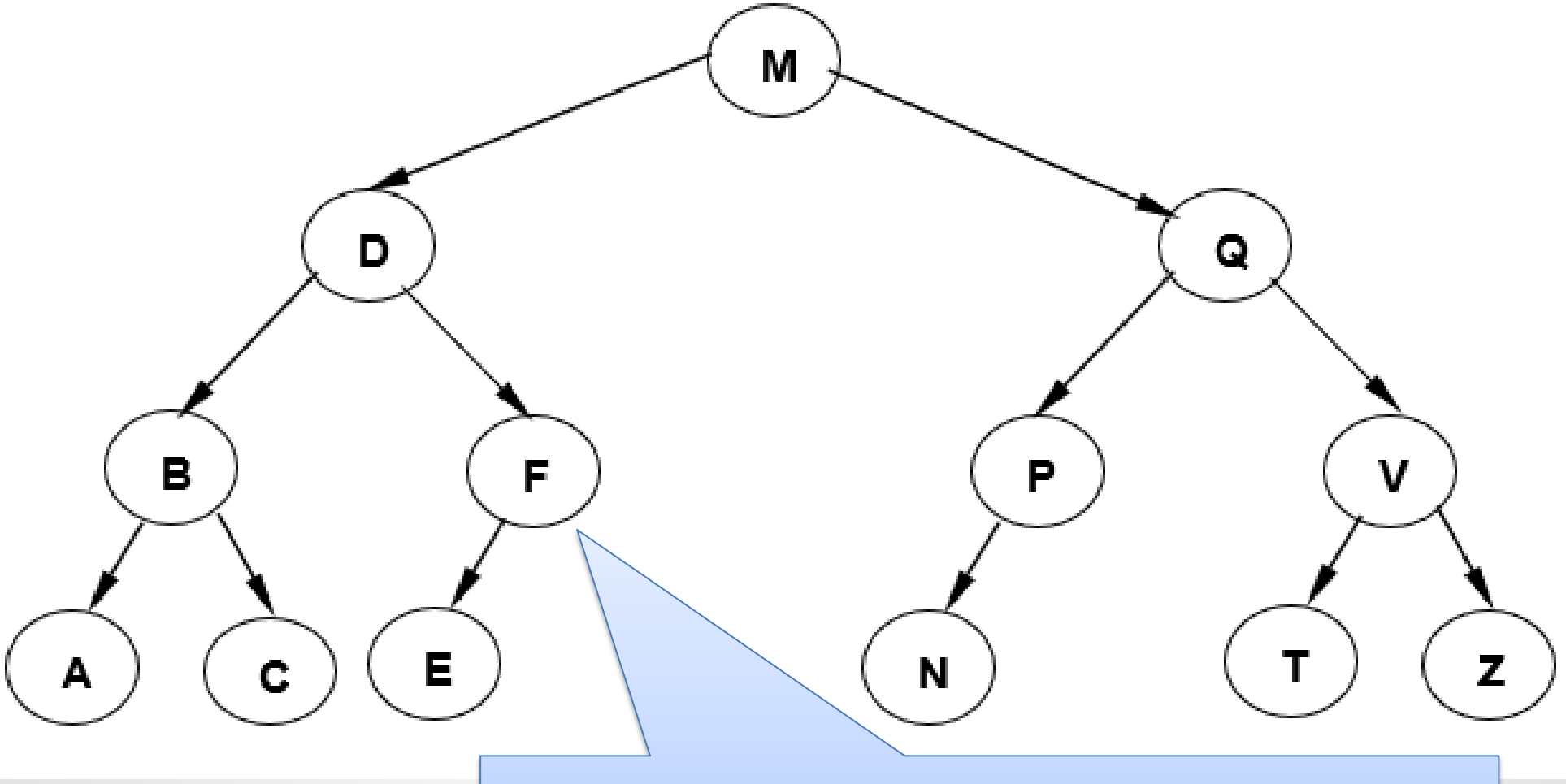


Si se elimina la Q, se reemplazará la información del nodo que contiene la **Q** por el predecesor inmediato, es decir la **P**, luego el nodo que **contenía** la P deberá ser eliminado.

4.1.2 Operaciones Básicas sobre AB

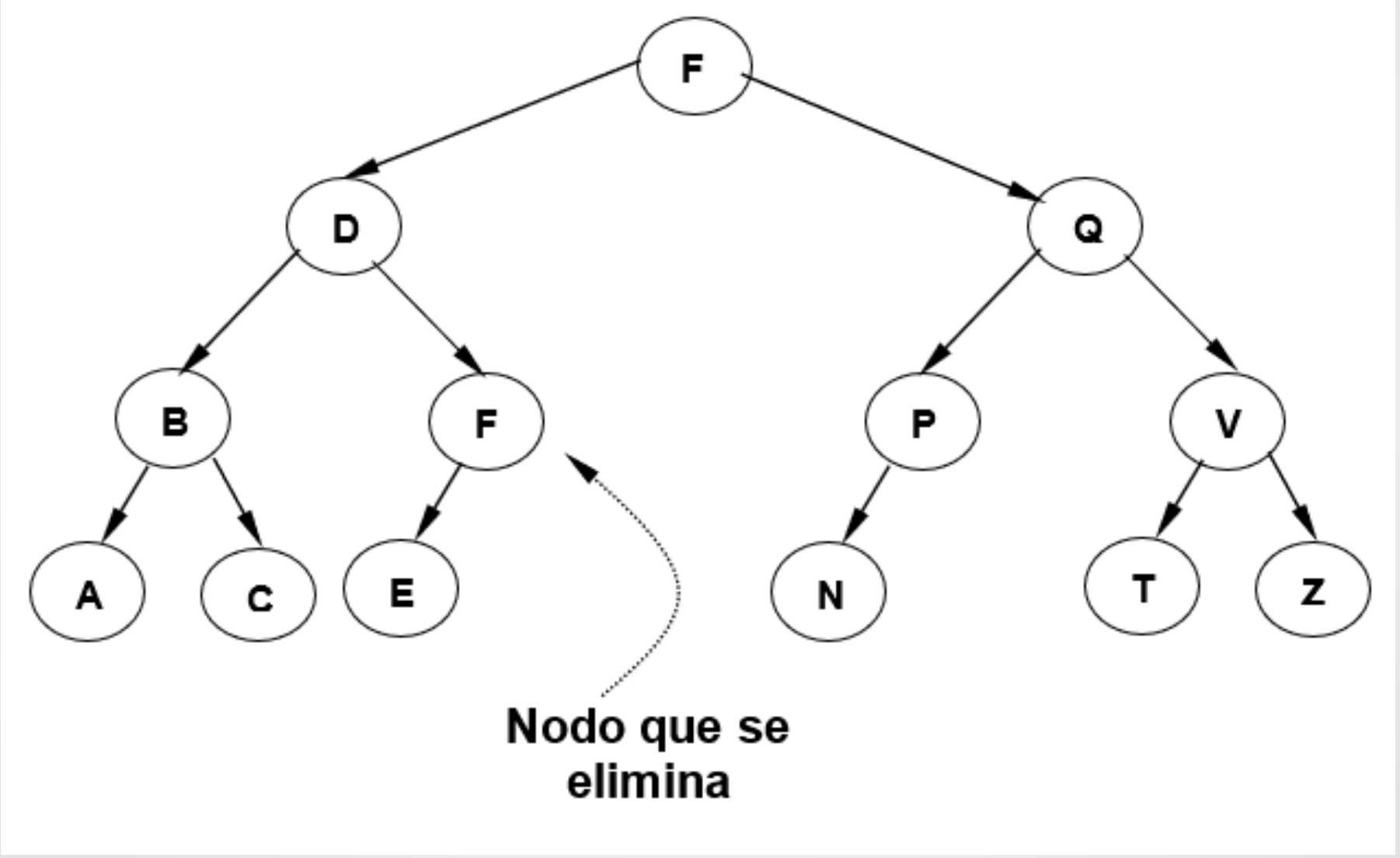


4.1.2 Operaciones Básicas sobre AB

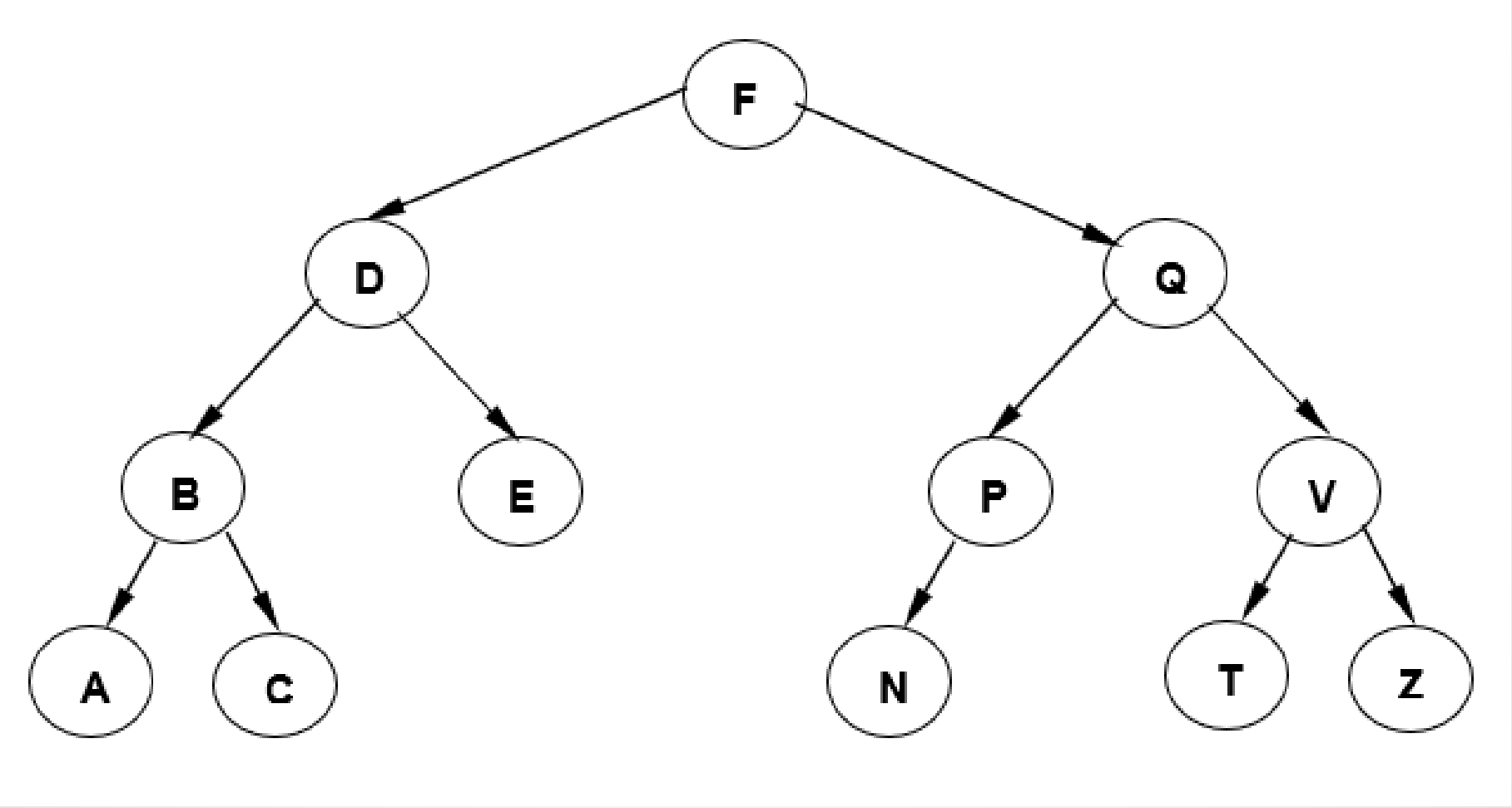


Si se elimina la M, se reemplazará la información del nodo que contiene la **M** por el predecesor inmediato, es decir la **F**, luego el nodo que **contenía** la **F** deberá ser eliminado.

4.1.2 Operaciones Básicas sobre AB



4.1.2 Operaciones Básicas sobre AB



4.1.2 Operaciones Básicas sobre AB

Ejercicio

- Construya un ABB con los datos que se muestran a la derecha.

- Luego elimine los siguientes valores del árbol:
  M H C W R

J	W	N	O	P
M	Q	U	X	R
Z	Y	K	L	H
C	A	D	F	G

Compruebe sus resultados con la URL que usamos antes
https://www.cs.usfca.edu/~galles/visualization/BST.html

Algoritmos:

P = dirección del nodo a eliminar.
PADRE = dirección del padre del nodo a eliminar.

PROCEDIMIENTO SUPRIME_NODO(P)

```
SI IZQ(P)=FIN Y DER(P)=FIN
        EJECUTA PROCEDIMIENTO CASO_1
DE LO CONTRARIO
        SI IZQ(P)<>FIN Y DER(P)<>FIN
                EJECUTA PROCEDIMIENTO CASO_3
        DE LO CONTRARIO
                EJECUTA PROCEDIMIENTO CASO_2
        FIN SI
FIN SI
```

Una vez ejecutado el algoritmo anterior, se sabrá cuantos hijos tiene el nodo a eliminar ....

CASO 1
(EL NODO A ELIMINAR ES UNA HOJA)

Hay que determinar si el nodo a eliminar es hijo izquierdo o derecho de su padre.

PROCEDIMIENTO CASO_1(P, PADRE)

```
SI PADRE=NULO
    RAIZ=NULO
NO
    SI IZQ(PADRE)=P
        IZQ(PADRE) ← NULO
    DE LO CONTRARIO
        DER(PADRE) ← NULO
    FIN SI
FIN SI
```

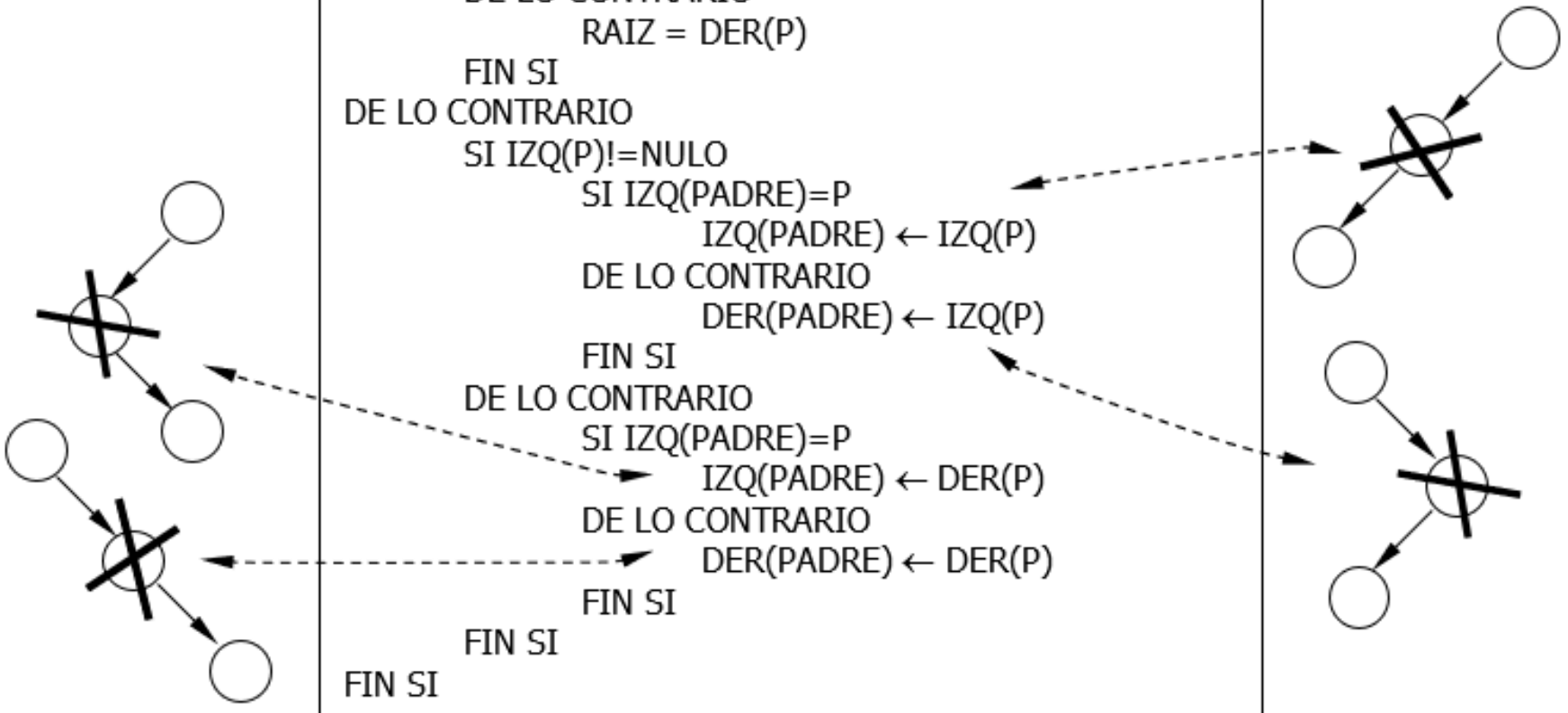
CASO 2

(EL NODO A ELIMINAR TIENE SOLO UN HIJO)

Hay que determinar si el nodo a eliminar es hijo izquierdo o derecho de su padre y si el único hijo que tiene es izquierdo o derecho.
Antes de lo anterior hay que verificar si el nodo a eliminar es la raíz.

PROCEDIMIENTO CASO_2 (P, PADRE)

```
SI PADRE=NULO
        SI IZQ(P) != NULO
                RAIZ = IZQ(P)
        DE LO CONTRARIO
                RAIZ = DER(P)
        FIN SI
DE LO CONTRARIO
        SI IZQ(P)!=NULO
                SI IZQ(PADRE)=P
                        IZQ(PADRE) ← IZQ(P)
                DE LO CONTRARIO
                        DER(PADRE) ← IZQ(P)
                FIN SI
        DE LO CONTRARIO
                SI IZQ(PADRE)=P
                        IZQ(PADRE) ← DER(P)
                DE LO CONTRARIO
                        DER(PADRE) ← DER(P)
                FIN SI
        FIN SI
FIN SI
```



CASO 3

(EL NODO A ELIMINAR TIENE DOS HIJOS)

P_SUST = Dirección del nodo sustituto
PADRE_SUST = Dirección del Padre del Nodo Sustituto.

PROCEDIMIENTO CASO_3(P)

```
PADRE_SUST ← P
P_SUST ← IZQ(P)
MIENTRAS DER(P_SUST) <> NULO
        PADRE_SUST ← P_SUST
        P_SUST ← DER(P_SUST)
FIN MIENTRAS

INFO(P) ← INFO(P_SUST)

SI PADRE_SUST = P
        IZQ(PADRE_SUST) ← IZQ(P_SUST)
DE LO CONTRARIO
        DER(PADRE_SUST) ← IZQ(P_SUST)
FIN SI
```

4.1.2 Operaciones Básicas sobre AB

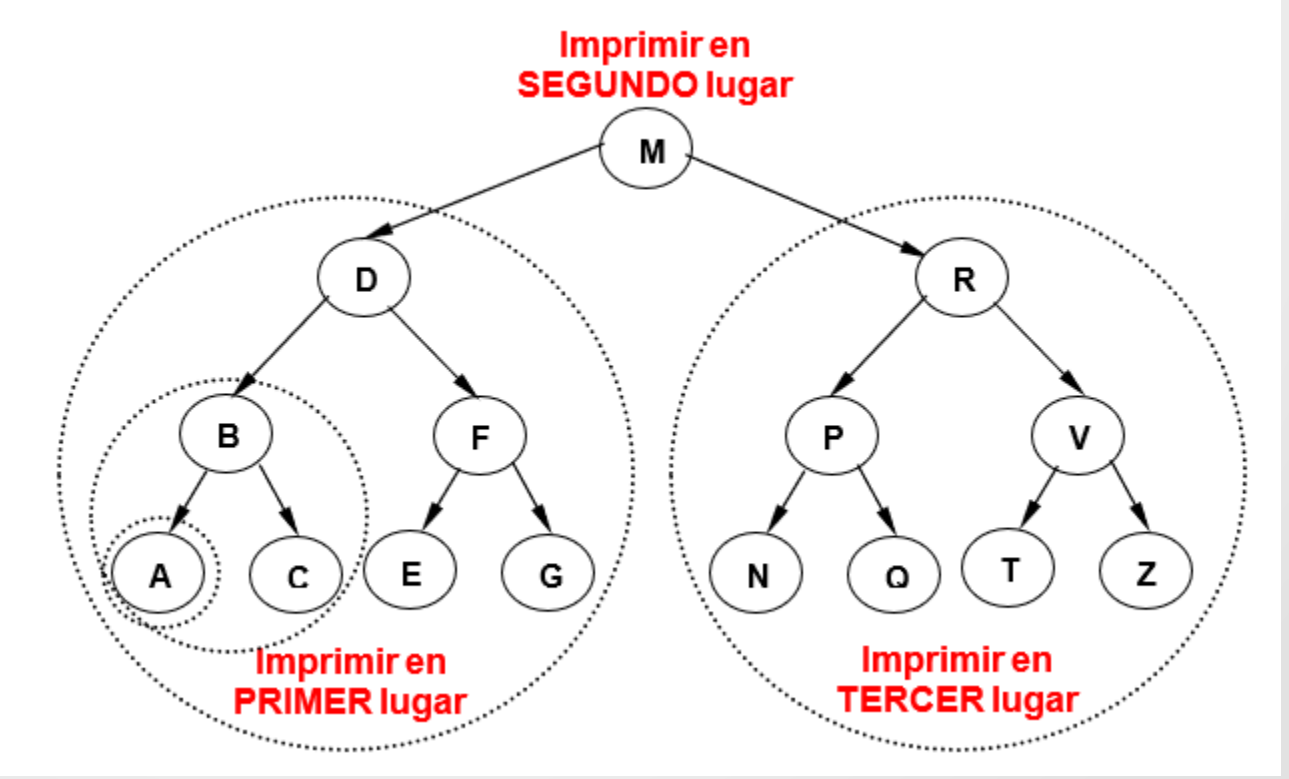
EJERCICIO 4

- Modificar la Clase ABB para que contenga un método que se llame “eliminar”.
  - Debe considerar por separado los tres casos dependiendo del número de hijos que tiene el nodo a eliminar.
- Modificar la aplicación para que haya una opción del menú para ejecutar la opción “eliminar”.

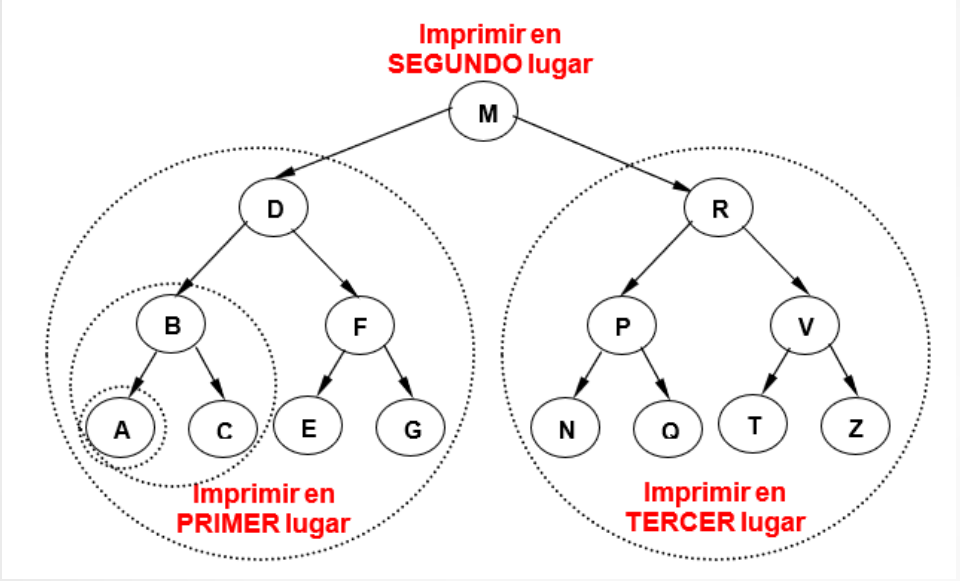
4.1.3 Recorridos

- Recorrer significa "visitar todos los nodos del árbol".
- En las listas lineales, solo hay un camino, pero en los Árboles Binarios, a partir de cada nodo hay dos posibilidades, izquierda o derecha.

Para imprimir los datos de un Árbol Binario de Búsqueda ordenados de menor a mayor:



4.1.3 Recorridos

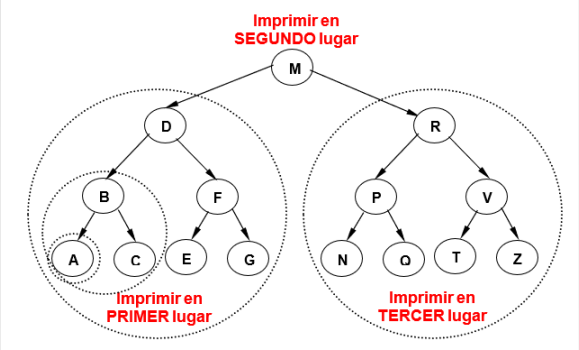


Para imprimir cada sub-árbol se repetirá el mismo proceso.

Este recorrido, para impresión ordenada, se conoce como **INORDEN** (IN=entre, es decir, la raíz ENTRE ambos subárboles).

Imprime en Inorden los datos del sub-árbol con raíz en el **hijo izquierdo de P**, luego imprime el dato del **nodo P** y finalmente, en Inorden, los datos del sub-árbol con raíz en el **hijo derecho de P**.

4.1.3 Recorridos



Esta tabla describe una metodología para conocer el resultado de la impresión Recursiva en **Inorden**.

							M		
			D				M		
	B		D				M		
A	B		D				M		
A	B	C	D				M		
A	B	C	D		F		M		
A	B	C	D	E	F		M		
A	B	C	D	E	F	G	M		
A	B	C	D	E	F	G	M	R	

Los recuadros vacíos muestran la posición para valores que se sabe estarán allí sin conocer aún su posición exacta.

Recorrido para impresión en INORDEN

Algoritmo Recursivo

```
PROCEDIMIENTO INORDEN(P)

SI P<>NULO
      EJECUTAR INORDEN(IZQ(P))
      IMPRIMIR INFO(P)
      EJECUTAR INORDEN(DER(P))
FIN SI
```

LLAMADA INICIAL: INORDEN(RAIZ)

Algoritmo Iterativo

```
P ← RAIZ
HASTA P=FIN Y PILAVACIA
      MIENTRAS P<>FIN
            PUSH(P)
            P ← IZQ(P)
      FIN MIENTRAS
      SI NO PILAVACIA
            P ← POP
            IMPRIME INFO(P)
            P ← DER(P)
      FIN SI
FIN HASTA
```

Recorridos para impresión en PREORDEN y POSTORDEN

PROCEDIMIENTO PREORDEN(P)

```
SI P<>FIN
      IMPRIMIR INFO(P)
      EJECUTAR PREORDEN(IZQ(P))
      EJECUTAR PREORDEN(DER(P))
FIN SI
```

Llamada Inicial: **PREORDEN(RAIZ)**

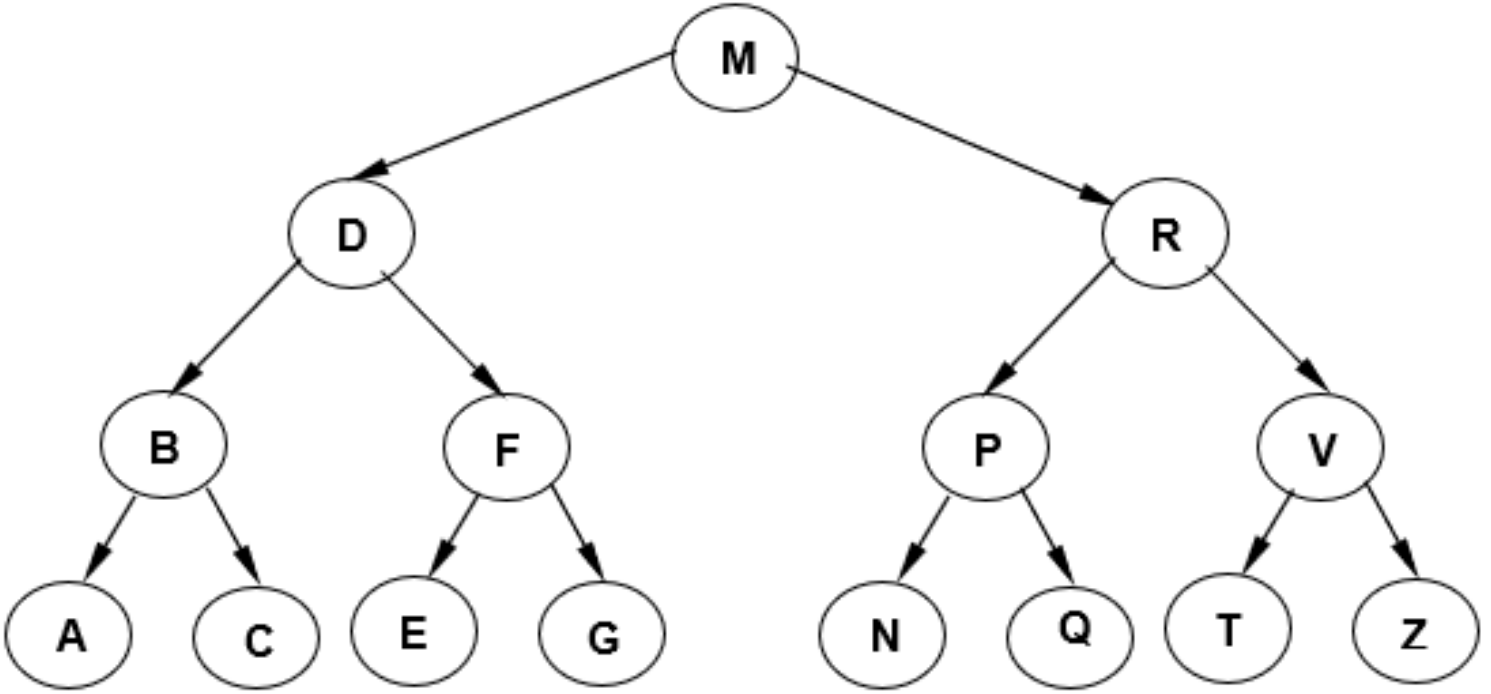
PROCEDIMIENTO POSTORDEN(P)

```
SI P<>FIN
      EJECUTAR POSTORDEN(IZQ(P))
      EJECUTAR POSTORDEN(DER(P))
      IMPRIMIR INFO(P)
FIN SI
```

Llamada Inicial: **POSTORDEN(RAIZ)**

4.1.3 Recorridos

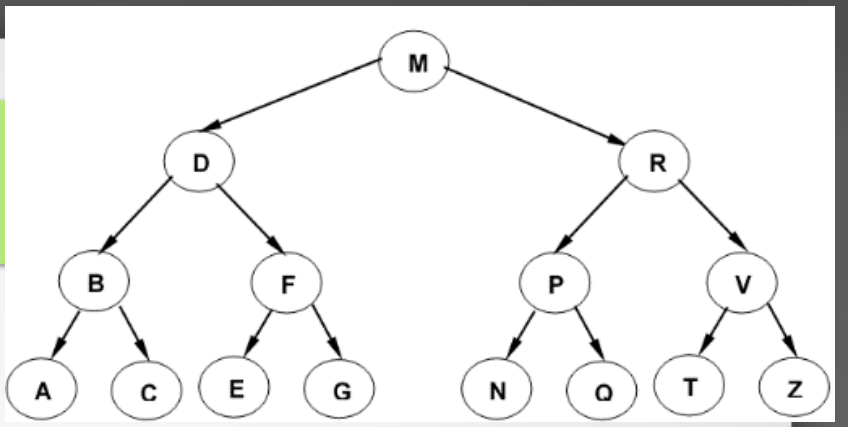
Impresiones en inorden y preorden para el árbol siguiente



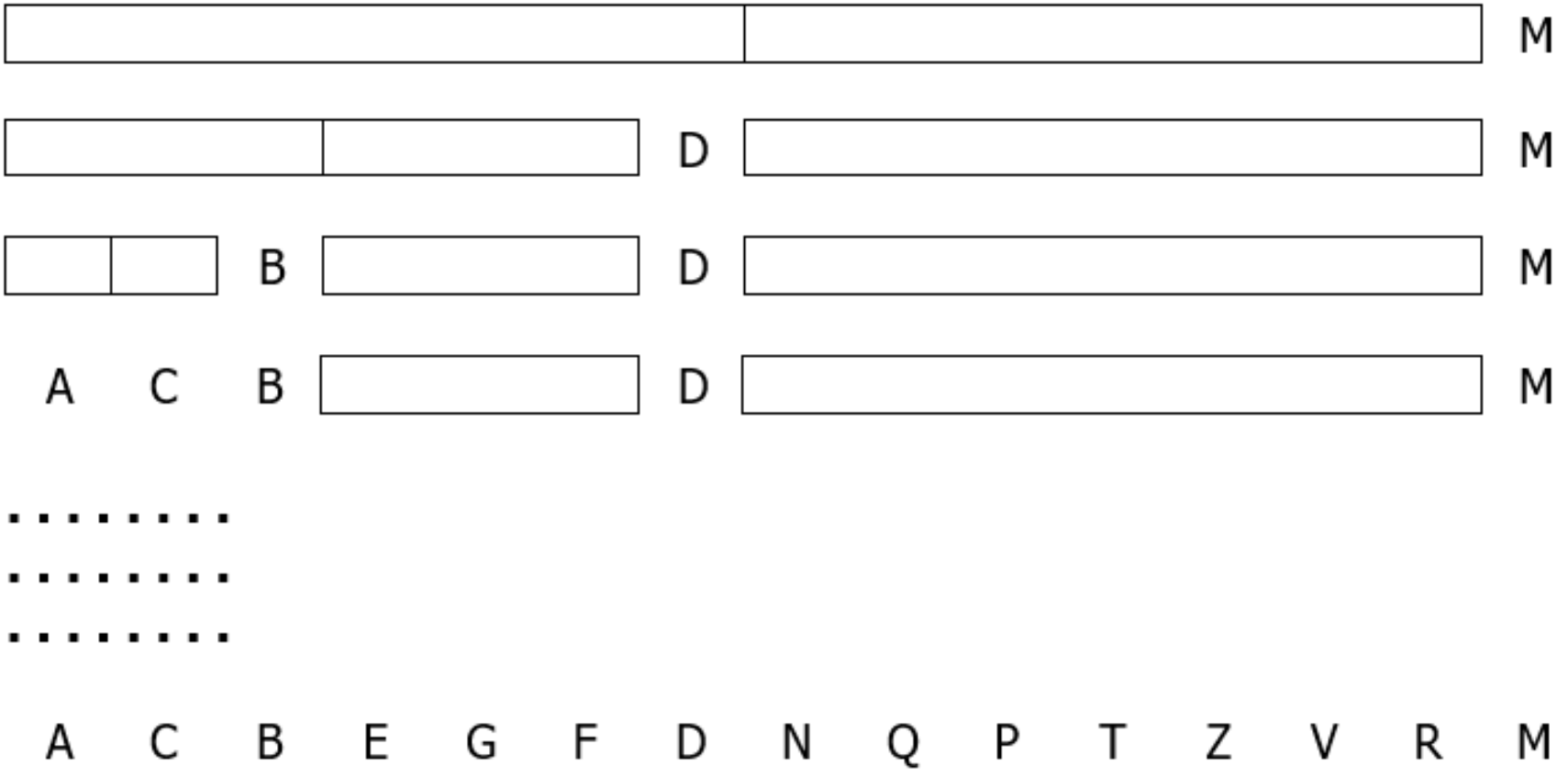
INORDEN: A B C D E F G M N P Q R T V Z

PREORDEN: M D B A C F E G R P N Q V T Z

4.1.3 Recorridos



POSTORDEN (Primero los sub-árboles y al final la raíz)**:**



Programa: EjemploABB.cpp

```
#include <iostream>
#include <fstream>
#include <cstring>
#include <conio.h>
using namespace std;
#include "ClaseABB.h"
int main() {
	char opcion;
	ifstream arch;
	string dato;
	ABB arbol;
	do {
		cout << "1.- Altas\n" << "2.- Consultar\n";
		cout << "3.- Eliminar\n" << "4.- Inorden\n";
		cout << "5.- Preorden\n" << "6.- Postorden\n";
		cout << "9.- Altas masivas\n" << "0.- Terminar\n\n";
		cout << "Opcion: ";opcion = getch(); cout << opcion;
		cout << endl;
		switch (opcion){
		case '1':
			cout<<"Nuevo Dato: ";getline(cin,dato);
			arbol.Anadir(dato);
				break;
		 case '2':
			cout<<"Dato a buscar: ";getline(cin,dato);
			arbol.Consultar(dato);
			break;
		 case '3':
			cout<<"Dato a Eliminar: ";getline(cin,dato);
			arbol.Eliminar(dato);
			break;
		 case '4':
			cout<<"----------------------"<<endl;
			cout<<"Impresion en Inorden"<<endl;
			cout<<"----------------------"<<endl;
			arbol.Inorden();
			cout<<endl;
			cout<<"----------------------"<<endl;
			cout<<endl;
			break;
		 case '5':
			cout<<"----------------------"<<endl;
			cout<<"Impresion en Preorden"<<endl;
			cout<<"----------------------"<<endl;
			arbol.Preorden();
			cout<<endl;
			cout<<"----------------------"<<endl;
			cout<<endl;
			break;
		 case '6':
			cout<<"----------------------"<<endl;
			cout<<"Impresion en Postorden"<<endl;
			cout<<"----------------------"<<endl;
			arbol.Postorden();
			cout<<endl;
			cout<<"----------------------"<<endl;
			cout<<endl;
			break;
		case '9':
			ifstream arch("DatosArbol.txt");
			if(!arch){
				cout<<"---------------------------"<<endl;
				cout<<"No existe o no se pudo abrir"<<endl;
				cout<<"el archivo DatosArbol.txt"<<endl;
				cout<<"---------------------------"<<endl;
				}
			else{
				cout<<"Datos importados: ";
				while (getline(arch, dato)) {
					cout << dato;
					arbol.Anadir(dato);
				}
				cout << endl;
				arch.close();
			}
			break;
		}
	   } while (opcion!='0');
	return 0;
}
```

Archivo: ClaseABB.h

```
class ABB{
	private:
		struct Nodo {
			Nodo* izq;
			string info;
			Nodo* der;
		};
		Nodo* raiz;
		void Inorden(Nodo*);
		void Preorden(Nodo*);
		void Postorden(Nodo*);
		void BorraNodos(Nodo*);
	public:
		ABB(); // Constructor
		void Anadir(string);
		void Consultar(string);
		void Eliminar(string);
		void Inorden();
		void Preorden();
		void Postorden();
		~ABB(); // Destructor
};

ABB::~ABB(){
	cout << "Datos Borrados: ";
	BorraNodos(raiz);
	cout << endl;
}
void ABB::BorraNodos(Nodo* p){
	// Borrado con recorrido en INORDEN
	Nodo* HijoIzq;
	Nodo* HijoDer;
	if (p!= NULL){
		HijoIzq=p->izq;
		HijoDer=p->der;
		BorraNodos(HijoIzq);
		cout << p->info;
		delete(p);
		BorraNodos(HijoDer);
	}
}
ABB::ABB(){
	raiz=NULL;
};
void ABB::Anadir(string x){
	Nodo *nn,*temp,*pnn;
	nn = new Nodo;
	nn->info = x;
	nn->izq = NULL;
	nn->der = NULL;
	temp = raiz;
	pnn  = NULL;
	while (temp!=NULL){
		pnn = temp;
		if (nn->info>temp->info)
			temp = temp->der;
		else
			temp = temp->izq;
	};
	if (pnn==NULL)
		raiz = nn;
	else
		if (nn->info > pnn->info)
			pnn->der = nn;
		else
			pnn->izq = nn;
	};

void ABB::Consultar(string x){
	Nodo *p,*padre;
	int nivel = 0;
	p = raiz;
	padre  = NULL;
	while ((p!=NULL) and (p->info != x)){
		nivel++;
		padre = p;
		if (x>p->info)
			p = p->der;
		else
			p = p->izq;
	};
	if (p==NULL)
		cout << "[" << x << "] no existe" << endl;
	else{
		if (padre==NULL)
			cout << "[" << x << "] es la raiz" << endl;
		else
			cout << "El padre de " << "[" << x << "] es: “
			<< padre->info << endl;
		if ((p->izq==NULL) and (p->der==NULL))
			cout << "[" << x << "] no tiene hijos" << endl;
		else if ((p->izq!=NULL) and (p->der!=NULL))
			cout << "[" << x << "] tiene DOS hijos" << endl;
		else
			cout << "[" << x << "] tiene UN hijo" << endl;
		cout << "El nivel del nodo es " << nivel << endl;
	}
}

void ABB::Eliminar(string x){
	Nodo*p,*padre,*p_sust,*padre_sust;
	// busca el valor a eliminar
	p = raiz;
	padre = NULL;
	while ((p!=NULL) and (p->info != x))  {
		padre = p;
		if (x>p->info)
			p = p->der;
		else
			p = p->izq;
	};
	if (p==NULL)
		cout << "No existe \"" << x << "\" en el arbol" <<
endl;
	else{
		if ((p->izq==NULL) and (p->der==NULL)){
			// caso 1 (es una hoja)
			if (padre==NULL)
				raiz = NULL;
			else if (padre->izq==p)
				padre->izq = NULL;
			else
				padre->der = NULL;
			delete(p);
			}
		else if ((p->izq!=NULL) and (p->der!=NULL)){
			// caso 3 (tiene 2 hijos)
			padre_sust = p;
			p_sust = p->izq;
			while (p_sust->der!=NULL){
				padre_sust = p_sust;
				p_sust = p_sust->der;
			}
			p->info = p_sust->info;
			if (padre_sust==p)
				padre_sust->izq = p_sust->izq;
			else
				padre_sust->der = p_sust->izq;
			delete(p_sust);
		}
		else{
			// caso 2 (solo tiene un hijo)
			if (padre==NULL){
				if (p->izq != NULL)
					raiz = p->izq;
				else
					raiz = p->der;
				}
			else if (p->izq != NULL)
				if (padre->izq==p)
					padre->izq = p->izq;
				else
					padre->der = p->izq;
			else
				if (padre->izq==p)
					padre->izq = p->der;
				else
					padre->der = p->der;
			delete(p);
			}
		cout << "'" << x << "' ha sido borrado" << endl;
	};
}
void ABB::Inorden(Nodo* p){
	if (p!= NULL){
		Inorden(p->izq);
		cout << p->info;
		Inorden(p->der);
	}
}
void ABB::Inorden(){
	Inorden(raiz);
}
void ABB::Preorden(Nodo* p){
	if (p!= NULL){
		cout << p->info;
		Preorden(p->izq);
		Preorden(p->der);
	}
}
void ABB::Preorden(){
	Preorden(raiz);
}
void ABB::Postorden(Nodo* p){
	if (p!= NULL){
		Postorden(p->izq);
		Postorden(p->der);
		cout << p->info;
	}
}
void ABB::Postorden(){
	Postorden(raiz);
}
```

4.2 Grafos

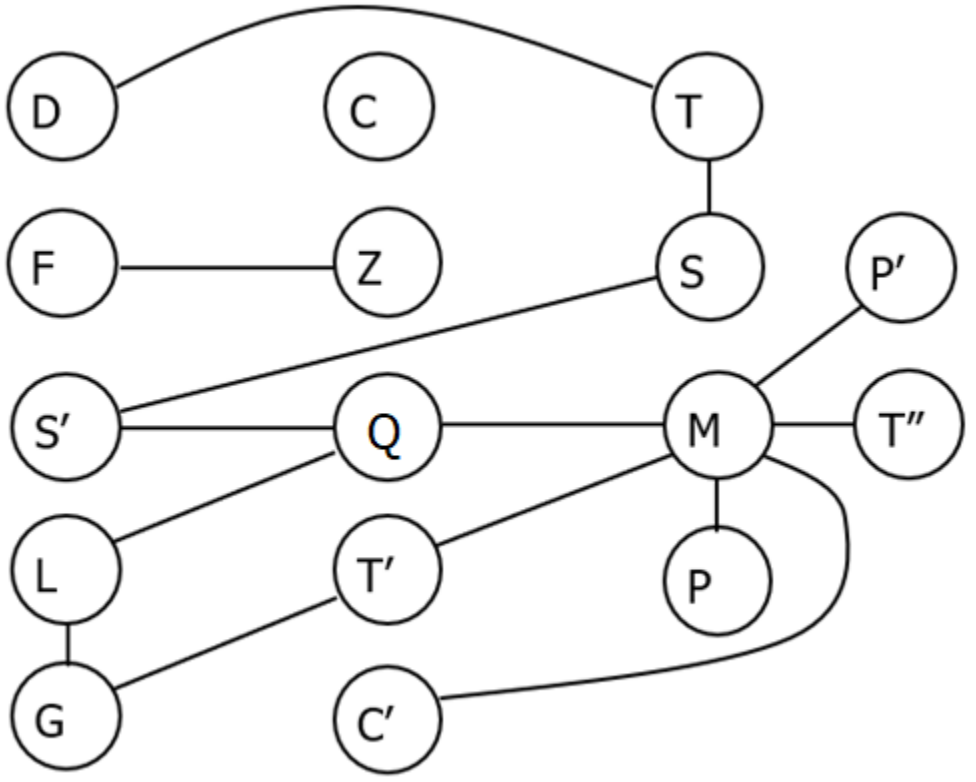
4.2.1 Terminología de Grafos

Un ejemplo típico de un **grafo** es un mapa de carreteras.

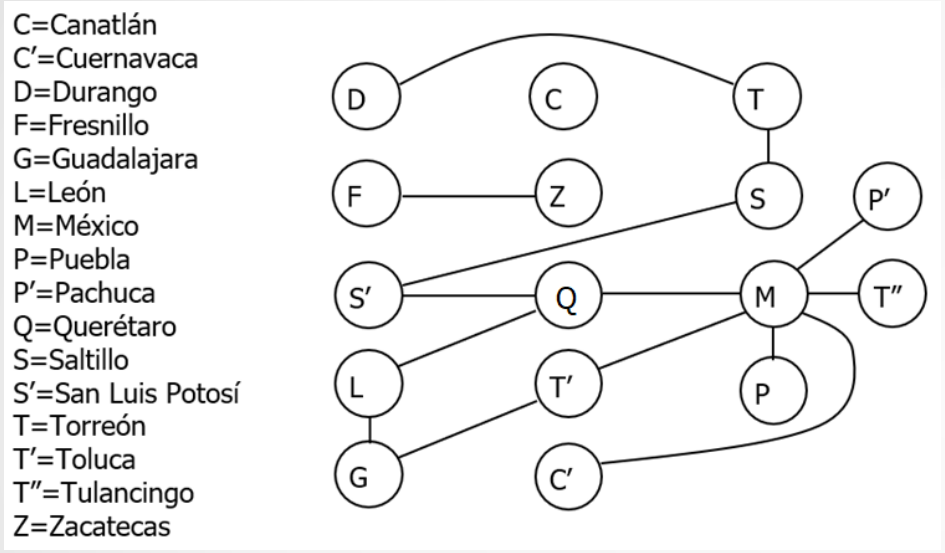
- Las ciudades equivalen a los **nodos** o vértices.
- Las carreteras corresponden con los **arcos** o aristas.

El siguiente grafo representa a una **mapa de carreteras de 4 carriles**.

C=Canatlán
C′=Cuernavaca
D=Durango
F=Fresnillo
G=Guadalajara
L=León
M=México
P=Puebla
P′=Pachuca
Q=Querétaro
S=Saltillo
S′=San Luis Potosí
T=Torreón
T′=Toluca
T″=Tulancingo
Z=Zacatecas



4.2 Grafos
4.2.1 Terminología



Este grafo se puede definir como el conjunto de nodos:

{ C, C′, D, F, G, L, M, P, P′, Q, S, S′, T, T′, T″, Z }

y el conjunto de arcos:

{D,T}, {T,S}, {F,Z}, {S,S′}, {S′,Q}, {Q,M}, {M,P′}, {M,T″}, {L,Q}, {T′,M}, {M,P}, {L,G}, {G,T′}, {C′,M}

No tienen significado alguno las siguientes características de un grafo:

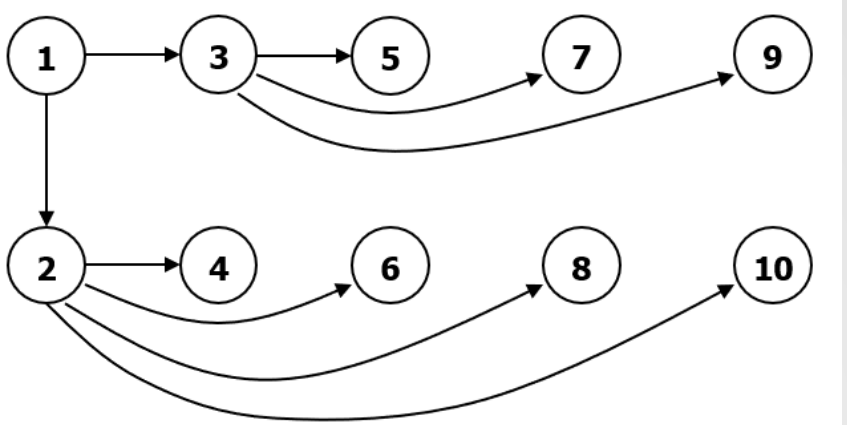
- Posición de un nodo con respecto a otro.
- La longitud de un arco.
- La forma de los arcos (pueden ser líneas curvas o rectas).
- El grueso de los arcos.

Observe el grafo y verá que las distancias entre Guadalajara-Toluca y Toluca-México parecen ser las mismas, pero no es así en la realidad.

4.2 Grafos
4.2.1 Terminología

Otro ejemplo:
Diagrama de tuberías de agua potable de una ciudad.

1 = Depósito Principal.
2,3 = Puntos de Distribución.
4-10 = Tomas de consumo (puntos finales).



Este grafo estaría definido como el conjunto de nodos:

{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 }

Y el conjunto de arcos:

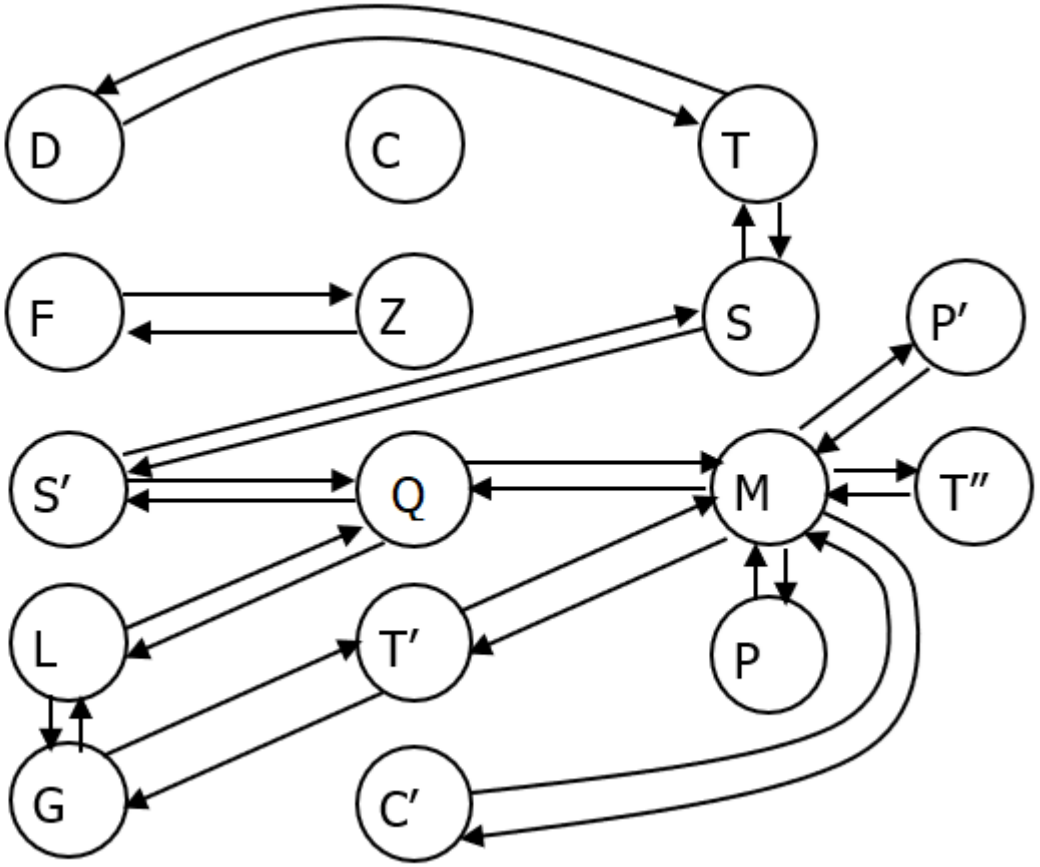
(1,2), (1,3), (3,5), (3,7), (3,9), (2,4), (2,6), (2,8), (2,10)

Este es un **grafo dirigido**; existe un arco (1,2) pero no un arco (2,1) ya que el agua **no puede fluir hacia** el depósito principal, por lo tanto, los pares de nodos son **pares ordenados** y la punta de flecha debe señalar al segundo nodo del par.

4.2 Grafos
4.2.1 Terminología

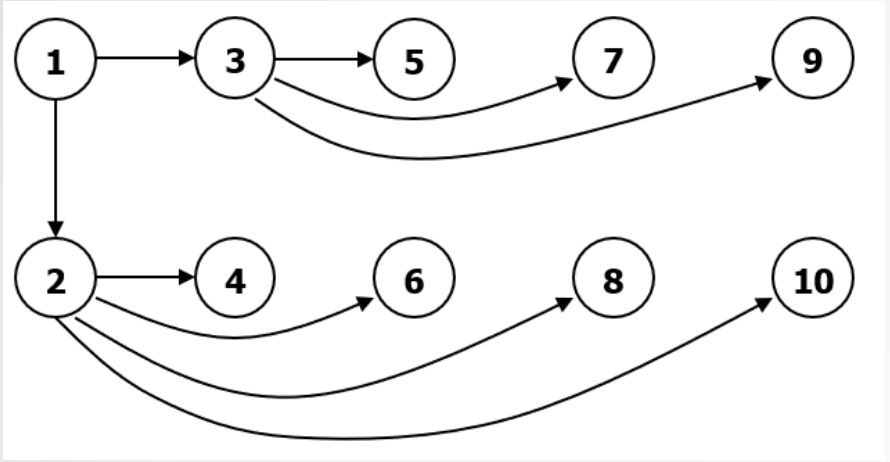
El grafo de las autopistas es un **grafo no dirigido**, el arco {D,T} representa a un par de arcos (D,T) y (T,D) viéndolo como un grafo dirigido.

C=Canatlán
C′=Cuernavaca
D=Durango
F=Fresnillo
G=Guadalajara
L=León
M=México
P=Puebla
P′=Pachuca
Q=Querétaro
S=Saltillo
S′=San Luis Potosí
T=Torreón
T′=Toluca
T″=Tulancingo
Z=Zacatecas



Esta representación es innecesaria, simplemente se representa como **no dirigido**.

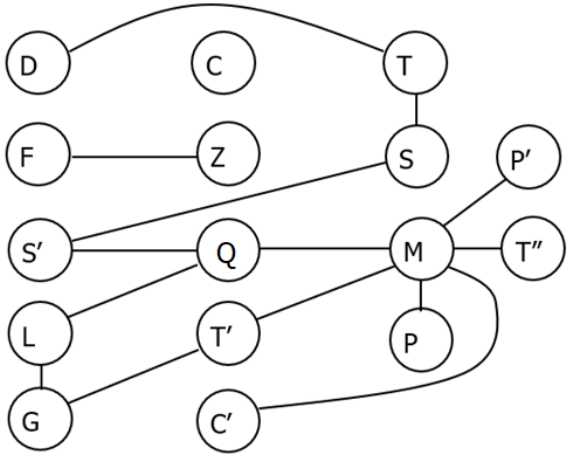
4.2 Grafos
4.2.1 Terminología



El **nodo 1 es incidente** a los arcos (1,2) y (1,3) o los **arcos (1,2) y (1,3) son incidentes al nodo 1**.
Sobre el **nodo 3** inciden **4 arcos** por lo tanto tiene un grado 4.

- **Grado de entrada** 1.
- **Grado de salida** 3.

C=Canatlán
C'=Cuernavaca
D=Durango
F=Fresnillo
G=Guadalajara
L=León
M=México
P=Puebla
P'=Pachuca
Q=Querétaro
S=Saltillo
S'=San Luis Potosí
T=Torreón
T'=Toluca
T''=Tulancingo
Z=Zacatecas

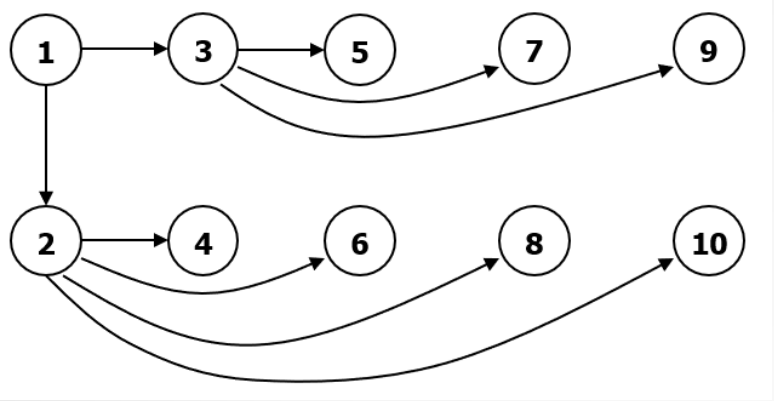


El nodo M del mapa de carreteras tiene un **grado 6**, pero no tiene grado de entrada ni de salida porque no se trata de un grafo dirigido.

Nodos adyacentes son aquellos que tienen un arco que los une.

Camino de longitud **k** entre 2 nodos **a** y **b** es aquella secuencia de **k+1** nodos, tal que $\mathbf{n_1}$ es el nodo **a**, $\mathbf{n_{k+1}}$ es el nodo **b** y todos los nodos entre **1** y **k** son adyacentes.

4.2 Grafos
4.2.1 Terminología

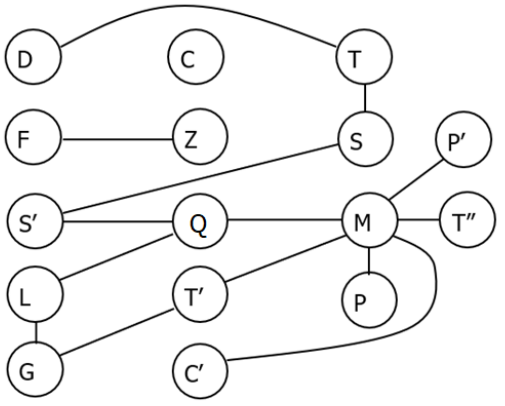


A los grafos que no contienen ningún ciclo se les llama **acíclicos**.
A los grafos dirigidos acíclicos se les llama **DAG (*directed acyclic graph*).**
El diagrama de tuberías es un DAG.

- De hecho es un árbol.
- Todos los árboles son DAG's.
- En los grafos dirigidos, como en este ejemplo, el nodo 2 es sucesor del nodo 1 y predecesor del nodo 4.

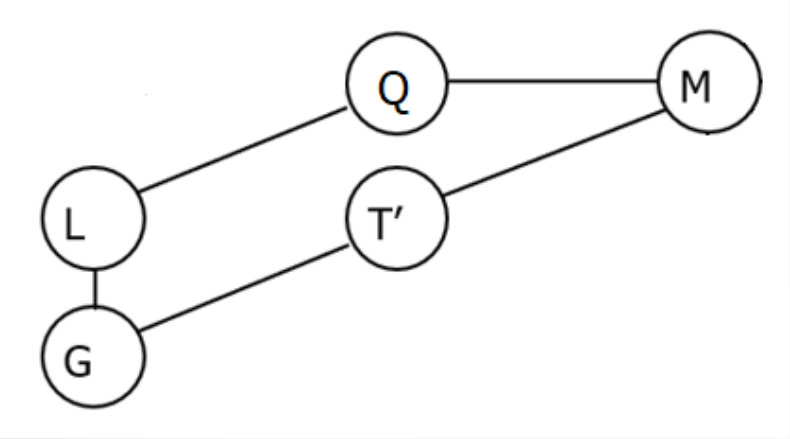
- Un camino de un nodo a sí mismo es un **ciclo**. A los grafos que contienen al menos un ciclo se les llama **cíclicos**.
- El mapa de autopistas es un **grafo cíclico**, observe que partiendo de casi cualquier ciudad podemos llegar a la misma (con algunas excepciones para aquellas ciudades en las que solo incide un arco).
- En los grafos no dirigidos, un mismo nodo puede actuar a veces como predecesor y en otras ocasiones como sucesor.

C=Canatlán
C'=Cuernavaca
D=Durango
F=Fresnillo
G=Guadalajara
L=León
M=México
P=Puebla
P'=Pachuca
Q=Querétaro
S=Saltillo
S'=San Luis Potosí
T=Torreón
T'=Toluca
T''=Tulancingo
Z=Zacatecas



4.2 Grafos
4.2.2 representación en memoria

Consideremos una parte de nuestro ejemplo de las carreteras:



Asignemos un entero a cada una de las ciudades

0=México
1=Toluca
2=Guadalajara
3=León
4=Querétaro

	M	T′	G	L	Q
M	0	1	0	0	1
T′	1	0	1	0	0
G	0	1	0	1	0
L	0	0	1	0	1
Q	1	0	0	1	0

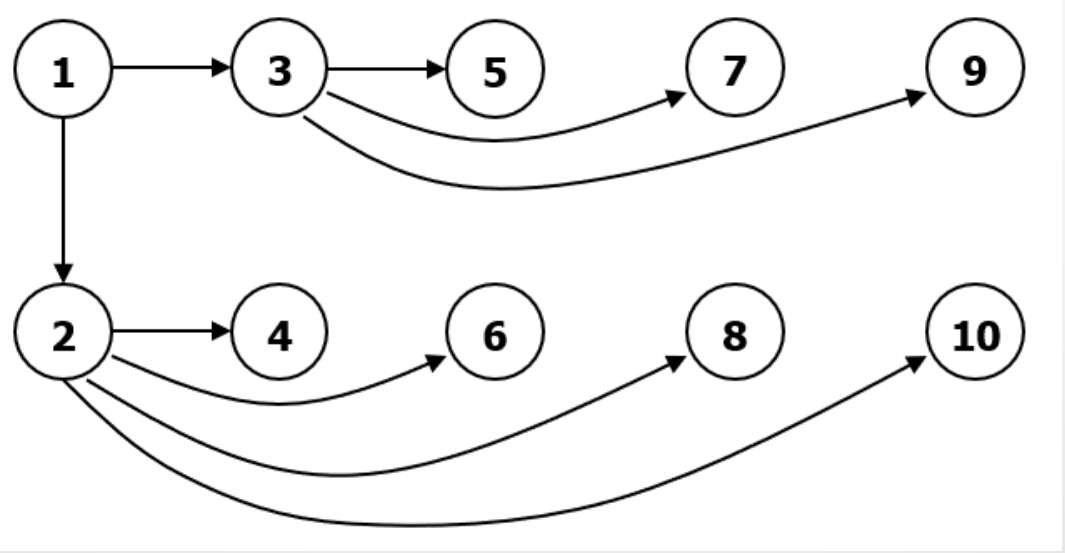
La matriz de adyacencia de la izquierda, siendo 0=False y 1=True, representa fielmente el grafo de esas 5 ciudades.

Esta matriz es simétrica ya que el grafo es no dirigido y por lo tanto, las coordenadas (2,1) y (1,2), representan al mismo arco. Lo mismo para otras parejas de coordenadas.

4.2 Grafos
4.2.2 representación en memoria

Ejercicio 1

Crear la matriz de adyacencia para el grafo dirigido de la distribución de agua potable. Recuerde que en los grafos dirigidos, (3,5) y (5,3) no representan al mismo arco y por lo tanto la matriz no será simétrica.



4.2 Grafos
4.2.2 representación en memoria

Ejercicio 2

Represente mediante un grafo:

- Un ejemplo de una pequeña parte de la red social *Facebook* considerando solo la relación de amistad en la que su cuenta personal es uno de los nodos, otros nodos son sus amigos, otros aunque no lo sean, son amigos de sus amigos y otros no están relacionados con usted de ninguna forma.
- Identifique si se trata de un grafo dirigido o no dirigido.
- Haga la matriz de adyacencia.

4.2 Grafos
4.2.2 representación en memoria

Ejercicio 3

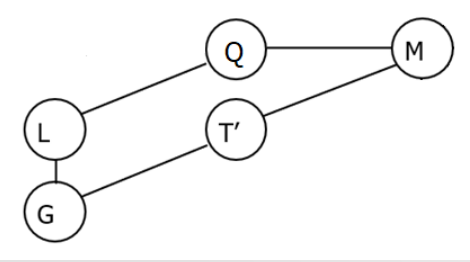
Represente mediante un grafo:

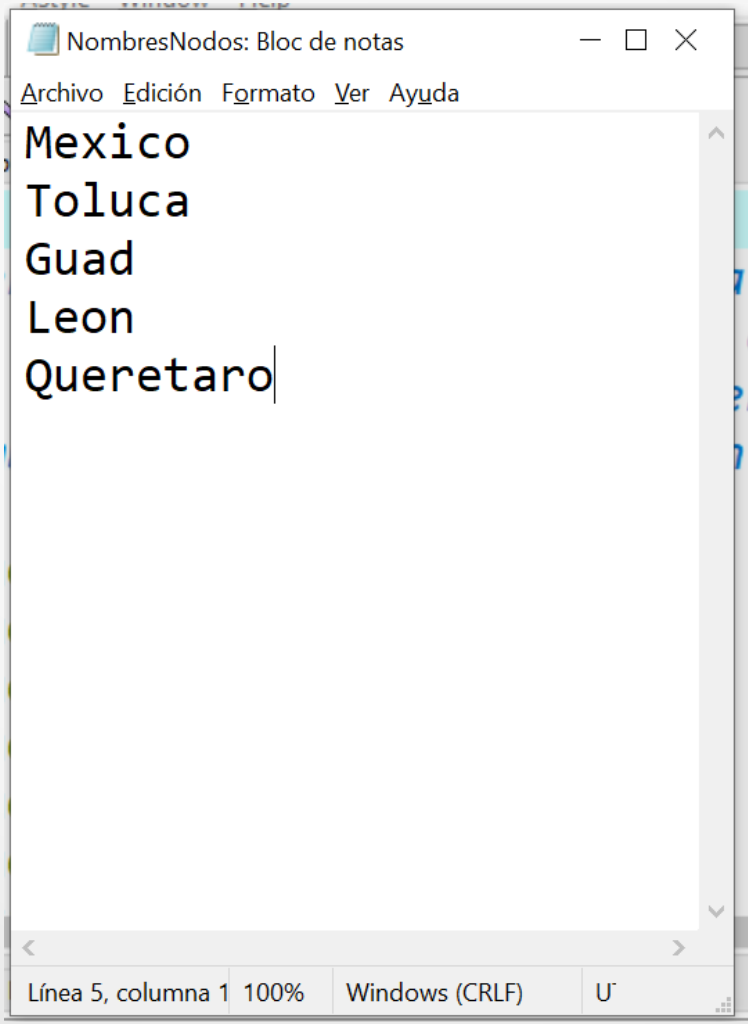
- Un ejemplo de una pequeña parte de la red de X (Twitter) en la que su cuenta personal es uno de los nodos, otros nodos son sus seguidores, otros a quien usted sigue y otros que no sigue y que tampoco lo siguen.
- Identifique si se trata de un grafo dirigido o no dirigido.
- Haga la matriz de adyacencia.

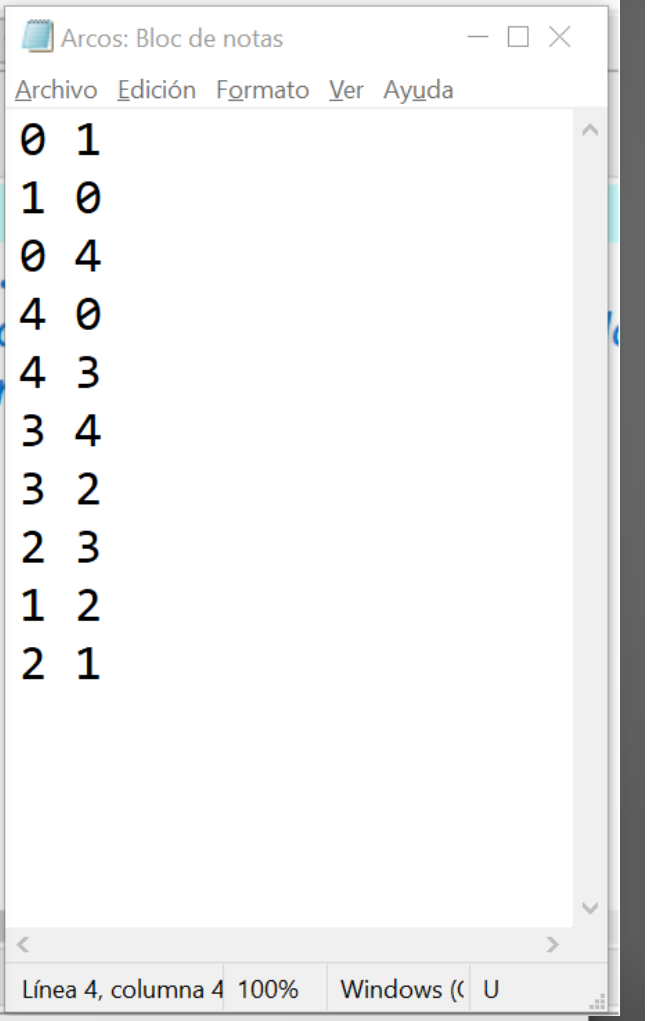
4.2 Grafos
4.2.2 representación en memoria

Ejercicio 4

Crear los archivos de texto **NombresNodos.txt** y **Arcos.txt** con los datos de algunos de los grafos que haya hecho. Aquí se muestra una parte del grafo de carreteras de 4 carriles.







Este programa crea la matriz de adyacencia de un grafo cualquiera a partir de los archivos NombresNodos.txt y Arcos.txt.

```
#include <iostream>
#include <fstream>
#include <cstring>
#include <conio.h>
#include <vector>
#include <string>
#include <sstream>
using namespace std;
int main()
{
		char Opcion;
		int NumeroNodos,i,j;
		string NombreNodo, Linea;
		int MaxLong=0;
		vector<string> NombresNodos;
		//--Se leen los Nombres de los Nodos
		ifstream archivo("NombresNodos.txt");
		if(!archivo){
				cout<<"No existe o no se pudo abrir el archivo NombresNodos.txt"<<endl;
				return 0;
				}
		else{
				while (getline(archivo, NombreNodo)) {
						NumeroNodos++;
						NombresNodos.push_back(NombreNodo);
						if(NombreNodo.size()>MaxLong) MaxLong=NombreNodo.size();
				}
		};
		MaxLong++; // Para que ninguna columna quede junto a otra
		archivo.close();
		//---- Se declara la matriz y se inicializa con ceros ----
		int Matriz[NumeroNodos][NumeroNodos];
		for(int i=0;i<NumeroNodos;i++)
				for(int j=0;j<NumeroNodos;j++)
						Matriz[i][j]=0;
		//--Se leen los Arcos
		ifstream archivo2("Arcos.txt");
		if(!archivo2){
				cout<<"No existe o no se pudo abrir el archivo Arcos.txt"<<endl;
				return 0;
				}
		else{
				while (getline(archivo2, Linea)) {
						sscanf(Linea.c_str(), "%d %d", &i, &j);
						Matriz[i][j]=1;
				}
		};
		archivo.close(); //------------- Fin de lectura -------------
```

```
            do {
                cout << "1.- Consultar Nodos\n";
                cout << "2.- Consultar Matriz de Adyacencia\n";
                cout << "0.- Terminar\n\n";
                cout << "Opcion: ";Opcion = getch(); cout << Opcion;
                cout << endl;
                switch (Opcion){
                    case '1':
                         cout << "-----------------" << endl;
                         cout << "      Nodos      " << endl;
                         cout << "-----------------"<<endl;
                         for(int i=0;i<NombresNodos.size();i++)
                                     cout << NombresNodos[i] << endl;
                                     cout << "Numero de Nodos: " << NumeroNodos << endl;
                                     cout << "-------------------"<<endl;
                                     break;
                          case '2':
                                     cout << string(MaxLong*2,' ');
                                     for(int i=0;i<NumeroNodos;i++)
                                                 cout << NombresNodos[i] << string(MaxLong-NombresNodos[i].size(), ' ');
                                     cout << endl;
                                     for(int i=0;i<NumeroNodos;i++){
                                                 cout << NombresNodos[i] << string(MaxLong-NombresNodos[i].size(), ' ') << " ";
                                                 for(int j=0;j<NumeroNodos;j++){
                                                             stringstream ssDato;
                                                             ssDato << Matriz[i][j];
                                                             cout << string(MaxLong-1,' ')+ssDato.str();
                                                 }
                                                 cout << endl;
                                     }
                                     cout<<endl;
                                     cout<<"----------------------"<<endl;
                                     cout<<endl;
                                     break;
                    }
                }
            while (Opcion!='0');
    return 0;
}
```

Ejercicio 5

Crear los archivos de texto **NombresNodos.txt** y **Arcos.txt** con los datos de los ejercicios 2 y 3 que usted realizó. Posteriormente hay que ejecutar el programa que muestra la matriz de adyacencia y compruebe que corresponde a los grafos de los ejercicios.

4.2 Grafos
4.2.3 Recorridos

BFS (*Breadth-First Search*)
Recorrido en anchura

Es una técnica de recorrido para grafos, para obtener los nodos hacia los cuales hay un camino partiendo de cierto nodo. El BFS se realiza explorando los vecinos de un nodo antes de profundizar con los vecinos de sus vecinos.

Se usa una Cola para garantizar que los nodos más cercanos al nodo origen, se visiten antes que los mas lejanos.

4.2 Grafos
4.2.3 Recorridos

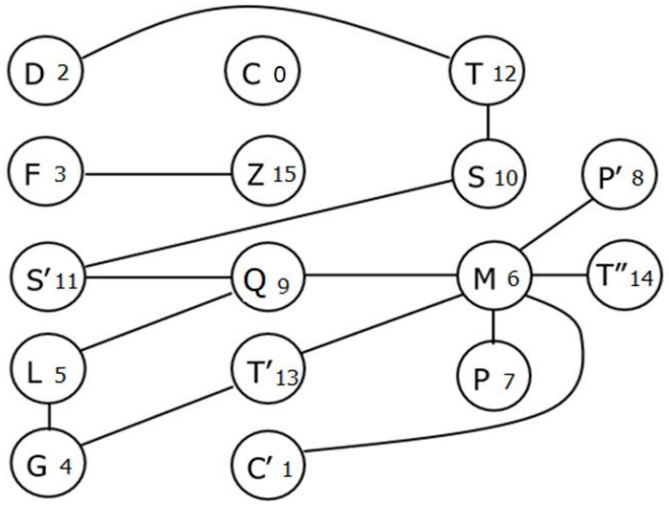
Ejercicio 6

Compruebe el algoritmo y el programa de recorrido en anchura considerando el grafo de autopistas de la pagina siguiente para obtener todos las ciudades conectadas a través de una autopista con la ciudad de San Luis Potosí.

Algoritmo para BFS
(se usa la matriz de adyacencia)

1. Etapa inicial:
   - Se elige un nodo inicial (llamado nodo fuente).
   - Se marca como visitado.
   - Se coloca en la cola.
2. Exploración de vecinos:
   - Mientras la cola no esté vacía:
     - Se extrae el siguiente nodo de la cola y se imprime.
     - Se inspeccionan todos sus vecinos en el orden de la matriz de adyacencia (los vecinos de cada nodo son los 1s del renglón correspondiente al nodo)..
       - Cada uno de los vecinos que no ha sido visitado:
         - Se marca como visitado.
         - Se agrega a la cola.
   - Se repite ciclo
3. Final:
   - El proceso termina cuando la cola queda vacía.

C=Canatlán	0
C′=Cuernavaca	1
D=Durango	2
F=Fresnillo	3
G=Guadalajara	4
L=León	5
M=México	6
P=Puebla	7
P′=Pachuca	8
Q=Querétaro	9
S=Saltillo	10
S′=San Luis Potosí	11
T=Torreón	12
T′=Toluca	13
T″=Tulancingo	14
Z=Zacatecas	15



	Can	Cue	Dgo	Fre	Gua	Leo	Mex	Pue	Pch	Qro	Sal	SLP	Trr	Tol	Tul	Zac
Can	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Cue	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
Dgo	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
Fre	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Gua	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0
Leo	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0
Mex	0	1	0	0	0	0	0	1	1	1	0	0	0	1	1	0
Pue	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
Pch	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
Qro	0	0	0	0	0	1	1	0	0	0	0	1	0	0	0	0
Sal	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
SLP	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
Trr	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0
Tol	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0
Tul	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
Zac	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0

4.2 Grafos
4.2.3 Recorridos

Aplicaciones de BFS:

- Búsqueda de rutas más cortas.
- Encontrar conexiones entre cuentas de redes sociales.
- Algoritmos de flujo en redes (tuberías de agua, tráfico en carreteras).
- Juegos de mesa como el ajedrez, por ejemplo (para explorar los movimientos válidos de una pieza).

4.2 Grafos
4.2.3 Recorridos

Código nuevo para nuestro ejemplo en C++

```
#include <algorithm>
#include "ColaDinamicaPlantilla.h“
-------
-------
cout << "3.- Recorrido en anchura (BFS)\n";
-------
-------
```

Los cambios continúan en las diapositivas siguientes

4.2 Grafos
4.2.3 Recorridos

```
case '3':{
          int n = NombresNodos.size();
          vector<bool> visitados(n, false);
          int NodoInicio;
          Cola<int> cola; // Cola para manejar la exploración BFS
          cout << "Busqueda en Anchura (BFS)\n";
          cout << "-------------------------\n";
          for(int i=0;i<NombresNodos.size();i++)
                    cout << i << ".- " << NombresNodos[i] << endl;
          cout << "Elige el Nodo de Inicio: ";
          cin >> NodoInicio;
          visitados[NodoInicio] = true;
          cola.entra(NodoInicio);
          cout << "Recorrido en anchura desde el nodo " << NombresNodos[NodoInicio] << ": ";
          while (!cola.vacia()) {
              int nodo = cola.sale();
              cout << NombresNodos[nodo] << " ";
              for (int vecino = 0; vecino < n; ++vecino) {
                   if (Matriz[nodo][vecino] == 1 && !visitados[vecino]) {
                             visitados[vecino] = true;
                             cola.entra(vecino);
                   }
              }
          }
          cout << endl;
          break;
}
```

ColaDinamicaPlantilla.h

```
template <class Tipo>
class Cola
    {
       private:
           struct Nodo
              {
              Tipo info;
              Nodo* sucesor;
              };
           Nodo* cabeza;
                        Nodo* ultimo;
       public:
           Cola();
           void entra(Tipo x);
           Tipo sale();
           bool vacia();
           bool llena();
           ~Cola();
     };
template <class Tipo>
Cola<Tipo>::Cola()
    {
    cabeza=NULL;
    ultimo=NULL;
    };
template <class Tipo>
void Cola<Tipo>::entra(Tipo x)
    {
    Nodo* temp;
    temp=new Nodo;
    temp->info = x;
    temp->sucesor = NULL;
    if (cabeza==NULL)
       cabeza=temp;
    else
       ultimo->sucesor=temp;
    ultimo=temp;
    };
template <class Tipo>
Tipo Cola<Tipo>::sale()
    {
    Nodo* temp=cabeza;
    Tipo result=cabeza->info;
    cabeza=cabeza->sucesor;
    if (cabeza==NULL)
       ultimo==NULL;
    delete temp;
    return result;
    };
template <class Tipo>
bool Cola<Tipo>::vacia()
    {
    return (cabeza==NULL);
    };
template <class Tipo>
bool Cola<Tipo>::llena()
    {
    return (false); // asumimos que siempre habrá
espacio en el Heap
    };
template <class Tipo>
Cola<Tipo>::~Cola()
    {
    Nodo *temp;
    while (cabeza!=NULL)
        {
        temp=cabeza;
        cabeza=cabeza->sucesor;
        delete temp;
        };
    };
```

Ejercicio 7

Compruebe con el grafo de autopistas y con el grafo de sus redes sociales el algoritmo de recorrido en anchura (BFS) para obtener los nodos que están conectados con el nodo correspondiente a su cuenta (puede verificar con cualquiera otros ejercicios que se hicieron).

- Modifique los archivos **NombresNodos.txt** y **Arcos.txt** de acuerdo al grafo que desea comprobar.

Algoritmo para camino más corto (basado en BFS)

1. Etapa inicial:
   - Se eligen los nodos origen y destino.
   - Se marca el origen como visitado.
   - Se coloca en la cola.
2. Exploración de vecinos:
   - Mientras la cola no esté vacía y no se llegue al nodo destino:
     - Se extrae el primer nodo de la cola
     - Se inspeccionan todos sus vecinos.
       - Si un vecino aún no ha sido visitado:
       - Se marca como visitado.
       - Se agrega a la cola.
       - Se guarda el nodo anterior en el vector NodoAnterior.
3. Reconstrucción del camino
   - Empezando en *NodoDestino* se reconstruye el camino hacia el *NodoOrigen* usando el vector *NodoAnterior*.
     - Se “viaja” a cada nodo anterior hasta llegar a un -1 (que es “el nodo anterior” al nodo origen)..
4. Como la ruta se reconstruye partiendo del destino, se debe invertir el vector “camino” para que se muestre la ruta en el orden correcto.

Como el recorrido es en anchura, en cuanto se llegue al *NodoDestino*, significa que se llegó por el camino más corto (si se siguiera buscando podría llegarse al destino por otro camino pero sería igual o más largo). Las diferencias con BFS son las marcas azules.

```
case '4': {int NodoInicio, NodoDestino;
          for(int i=0;i<NombresNodos.size();i++)
                    cout << i << ".- " << NombresNodos[i] << endl;
          cout << "Elige el Nodo de Inicio: ";
          cin >> NodoInicio;
          cout << "Elige el Nodo Destino: ";
          cin >> NodoDestino;
          int n = NombresNodos.size();
          vector<bool> visitados(n, false);
          vector<int> NodoAnterior(n, -1); // Para reconstruir el camino
          Cola<int> cola;
          visitados[NodoInicio] = true;
          cola.entra(NodoInicio);
          while (!cola.vacia()) {
                    int nodo = cola.sale();
                    if (nodo == NodoDestino)
                        break; // Encontramos el destino
                    for (int vecino = 0; vecino < n; ++vecino) {
                        if (Matriz[nodo][vecino] == 1 && !visitados[vecino]) {
                               visitados[vecino] = true;
                               NodoAnterior[vecino] = nodo; // Recordamos cómo llegamos
                               cola.entra(vecino);
                        }
                    }
          }
          vector<int> camino;
          cout << "--------- Camino mas corto -----------\n";
          if (visitados[NodoDestino]) {
              cout << "Camino mas corto: ";
              for (int x = NodoDestino; x != -1; x = NodoAnterior[x])
                     camino.push_back(x);
              reverse(camino.begin(), camino.end());
              for (int i=0;i<camino.size();i++)
                     cout << NombresNodos[camino[i]] << " ";
              cout << endl;
              }
          else
              cout << "No hay camino\n";
          cout << "--------------------------------------\n";
          break;
}
```

Añada al menú la siguiente opción:
cout << "4.- Camino mas corto usando BFS\n";

Ejercicio 8

Compruebe el funcionamiento de la opción 4 del programa (camino mas corto) para cualquier par de nodos, origen y destino, con los diferentes grafos con que hemos practicado.

4.3 Redes

A los grafos que tienen un número asociado a cada arco se les llama **redes**.

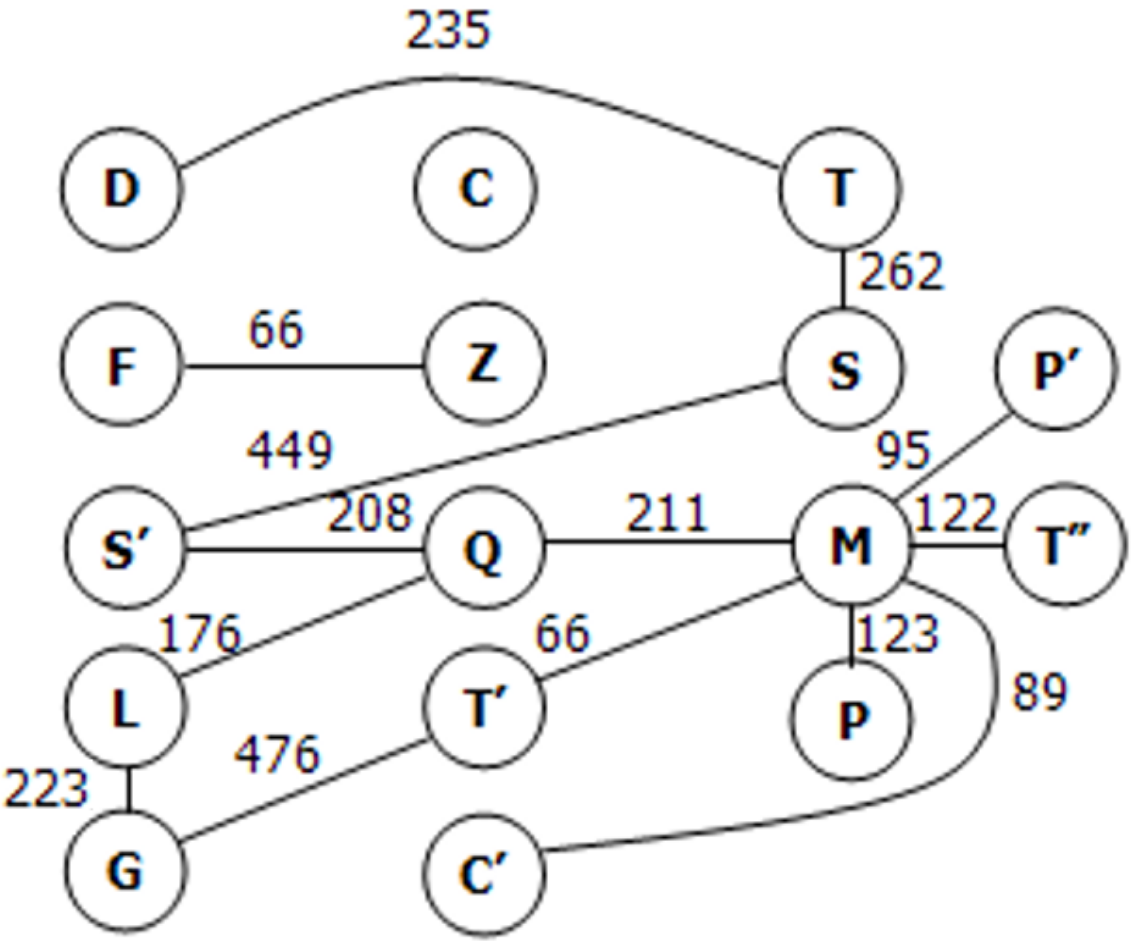
- Al número se le llama **peso**.
- Las redes son también conocidas como grafos ponderados.

Ejemplo: **Red de autopistas** de México.

- Si el peso es la distancia en kilómetros entre un par de ciudades, puede obtenerse la ruta mas corta ya no solo considerando el numero de arcos sino en kilómetros.
- Si el peso es el precio de las casetas de pago, se podría obtener la ruta que sea menos costosa.
- El peso puede ser cualquier cosa, por ejemplo, el número de accidentes que históricamente han ocurrido en esa carretera.

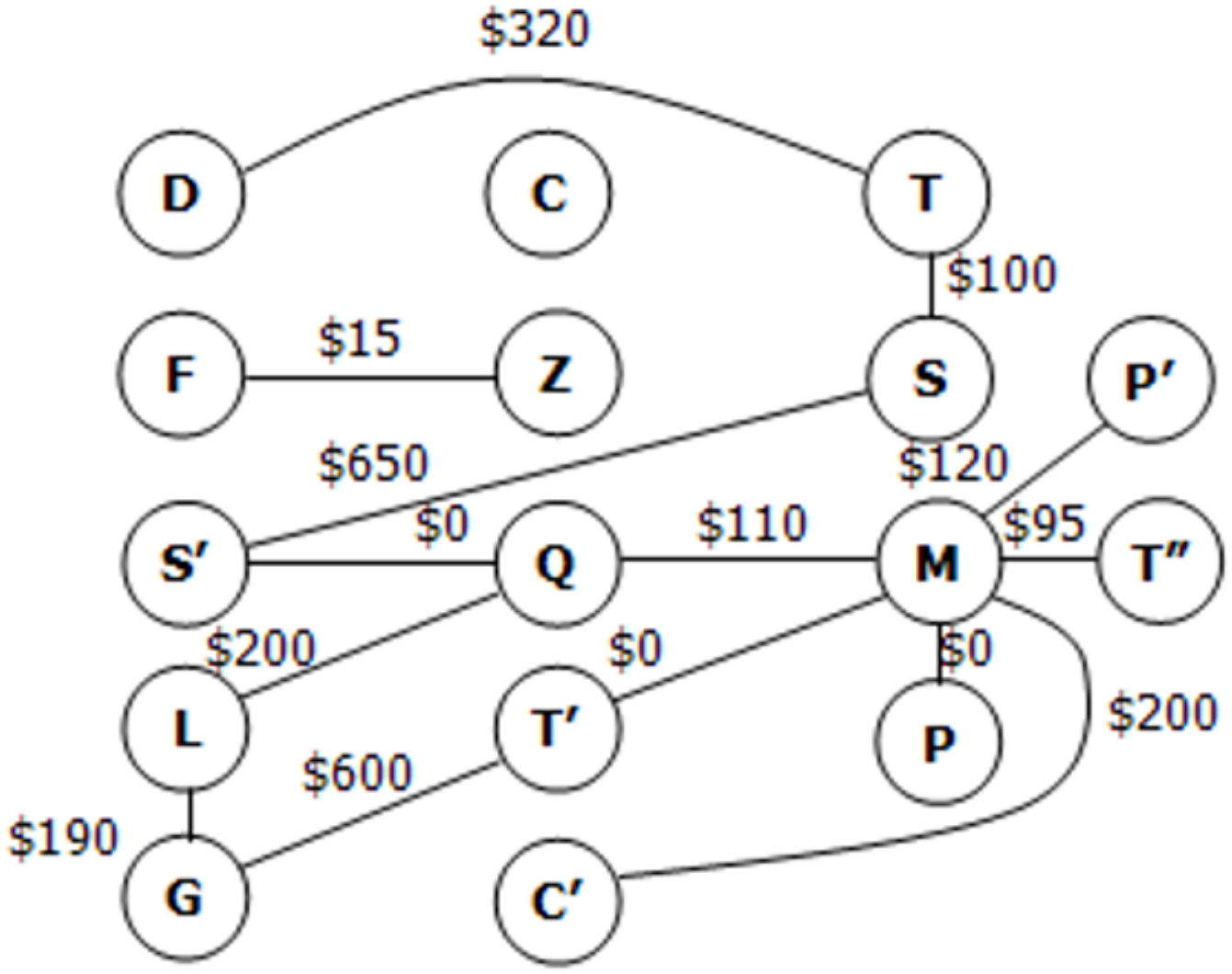
4.3 Redes
Ejemplo de una red de distancias en km de autopistas

C=Canatlán
C′=Cuernavaca
D=Durango
F=Fresnillo
G=Guadalajara
L=León
M=México
P=Puebla
P′=Pachuca
Q=Querétaro
S=Saltillo
S′=San Luis Potosí
T=Torreón
T′=Toluca
T″=Tulancingo
Z=Zacatecas



4.3 Redes
Ejemplo de una red de precios de autopistas

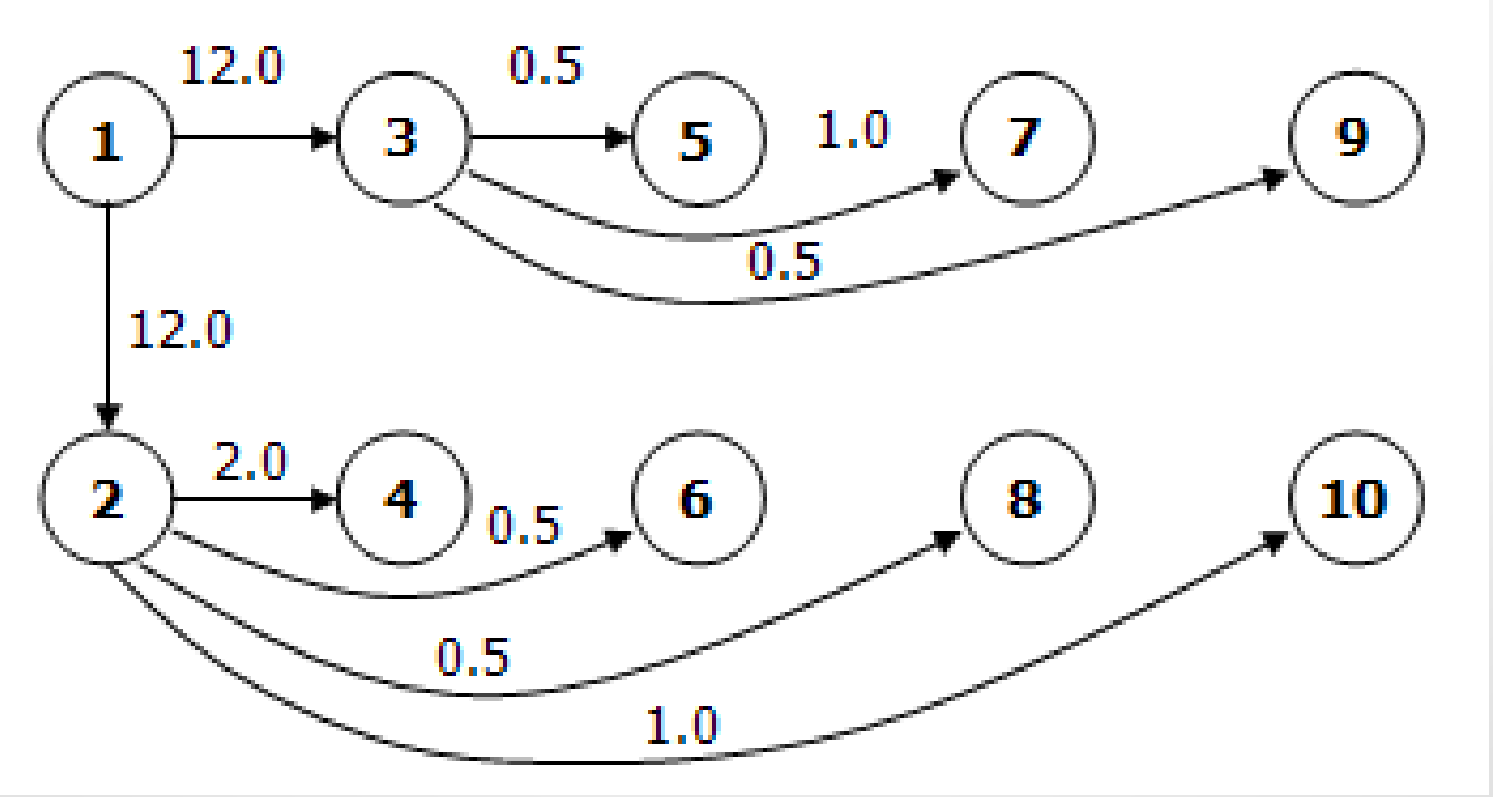
C=Canatlán
C′=Cuernavaca
D=Durango
F=Fresnillo
G=Guadalajara
L=León
M=México
P=Puebla
P′=Pachuca
Q=Querétaro
S=Saltillo
S′=San Luis Potosí
T=Torreón
T′=Toluca
T″=Tulancingo
Z=Zacatecas



4.3 Redes

La **red de agua potable** de una ciudad.

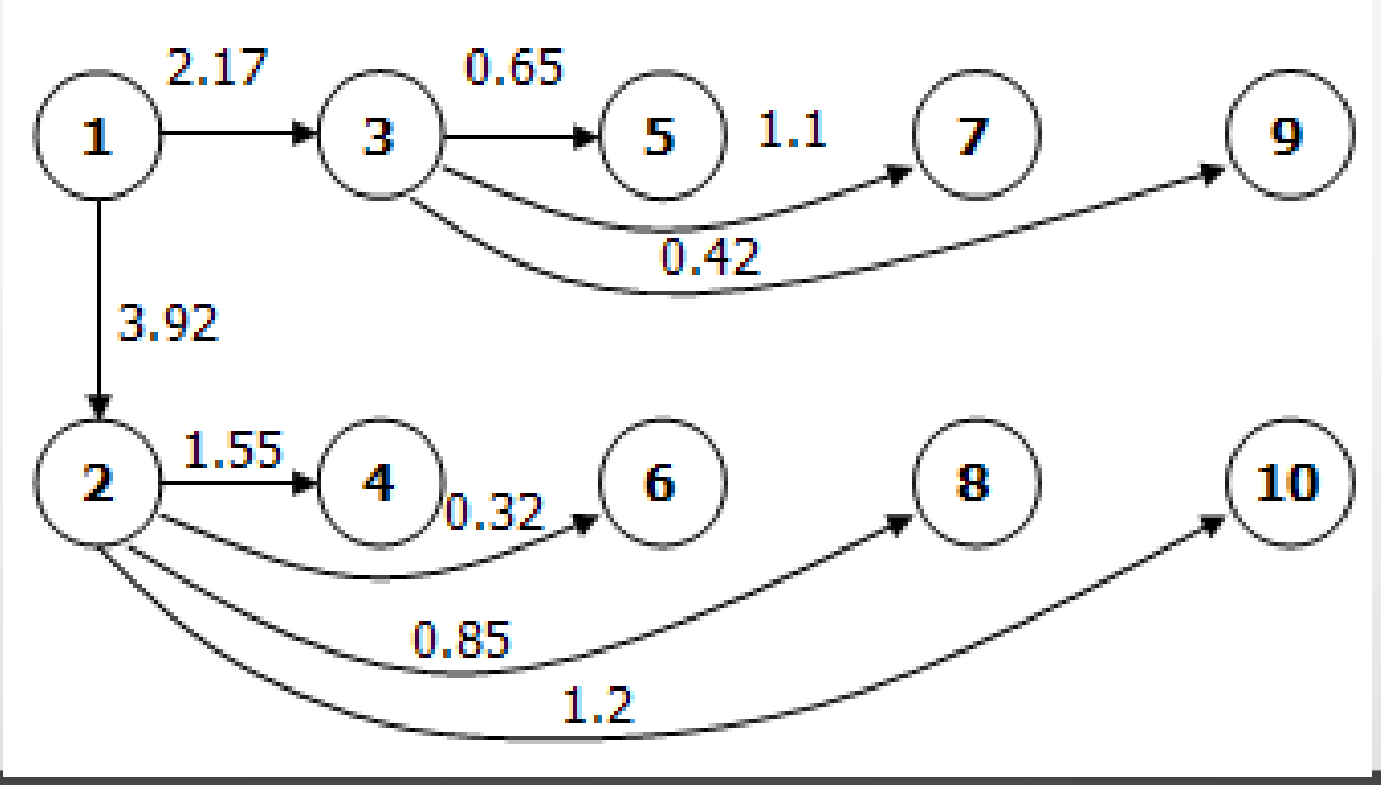
- El peso podría ser el diámetro de la tubería en pulgadas.
  - Esto nos informaría cuantas tomas pueden ser alimentadas por un tanque de distribución.
- ¿Cuántas tomas de **1”** pueden ser alimentadas por el nodo 3 si la entrada es un tubo de **12”**?



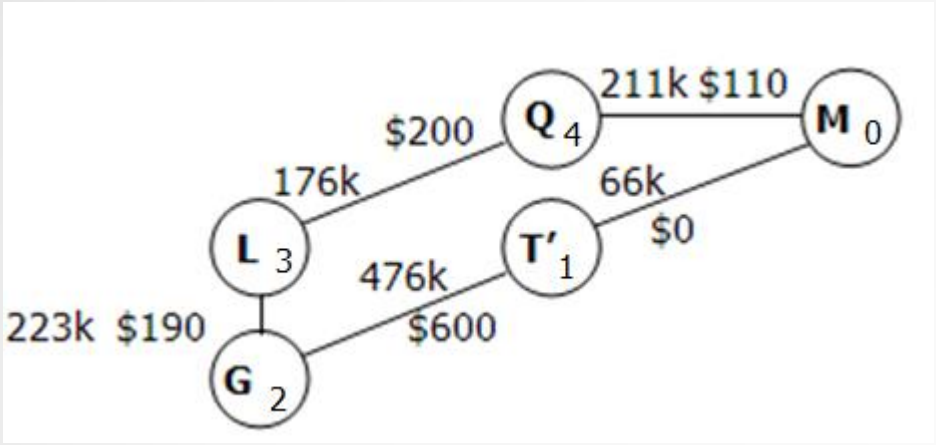
4.3 Redes

Otro ejemplo de un grafo ponderado de la red de agua potable:

- El peso representaría el gasto (consumo) expresado en metros cúbicos diarios (1 m3=1000 litros).
- El gasto de entrada a un nodo debe ser igual a la suma de los gastos de salida (asumiendo que no hay fugas en el sistema).
- De esta forma se puede calcular el volumen de los depósitos de agua así como la capacidad de las bombas que transportan el agua hacia los depósitos.



4.3 Redes
Representación en memoria



Matriz de precio

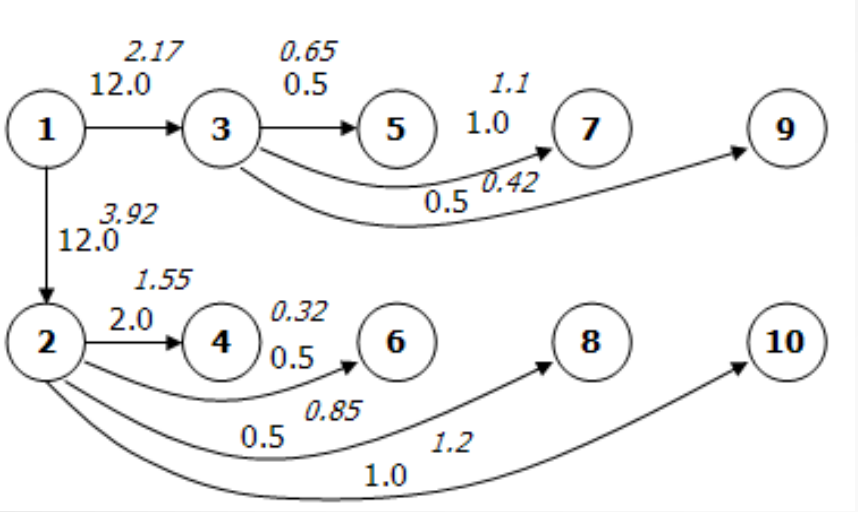
	M	T′	G	L	Q
M	∞	**0**	∞	∞	**110**
T′	**0**	∞	**600**	∞	∞
G	∞	**600**	∞	**190**	∞
L	∞	∞	**190**	∞	**200**
Q	**110**	∞	∞	**200**	∞

Las matrices de la derecha representan los 2 diferentes pesos de la red.
Observe que el símbolo ∞ representa un peso enormemente grande, lo que significa la imposibilidad práctica de ir de un nodo a otro porque no existe un arco entre ellos.

Matriz de distancia

	M	T′	G	L	Q
M	∞	**66**	∞	∞	**211**
T′	**66**	∞	**476**	∞	∞
G	∞	**476**	∞	**223**	∞
L	∞	∞	**223**	∞	**176**
Q	**211**	∞	∞	**176**	∞

4.3 Redes
Representación en memoria



Matriz de diámetro de tubería

PREDECESORES \ SUCESORES	1	2	3	4	5	6	7	8	9	10
1	0	**12.0**	**12.0**	0	0	0	0	0	0	0
2	0	0	0	**2.0**	0	**0.5**	0	**0.5**	0	**1.0**
3	0	0	0	0	**0.5**	0	**1.0**	0	**0.5**	0
4	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0

Matriz de consumo

PREDECESORES \ SUCESORES	1	2	3	4	5	6	7	8	9	10
1	0	**3.92**	**2.17**	0	0	0	0	0	0	0
2	0	0	0	**1.55**	0	**0.32**	0	**0.85**	0	**1.2**
3	0	0	0	0	**0.65**	0	**1.1**	0	**0.42**	0
4	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0

Observe que **los pesos** en nodos no adyacentes en este problema se representan con **cero** a diferencia de las matrices de distancia y precios de las autopistas en las que se representa con **infinito**.

Ejercicio 9

Identifique los posibles diferentes pesos para cada uno de sus grafos personales de redes sociales (Facebook y de Twitter).