Estructura de Datos

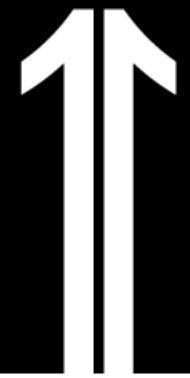
Del 24 al 28 de Junio del 2024

UPN.EDU.PE

Semana 06

RECORRIDO DE ÁRBOLES

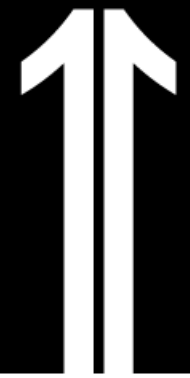
PRESENTACIÓN DE LA SESIÓN
Logro de la Sesión y Temario



Al término de la sesión, el estudiante aprende algoritmos con arboles, arboles binarios y diversas aplicaciones, usándolos con coherencia.

- Árboles: Generalidades. Arboles binarios.
- Operaciones: Raíz, hoja, tallo, recorrido inorden, postorden, preorden.

Reflexiona



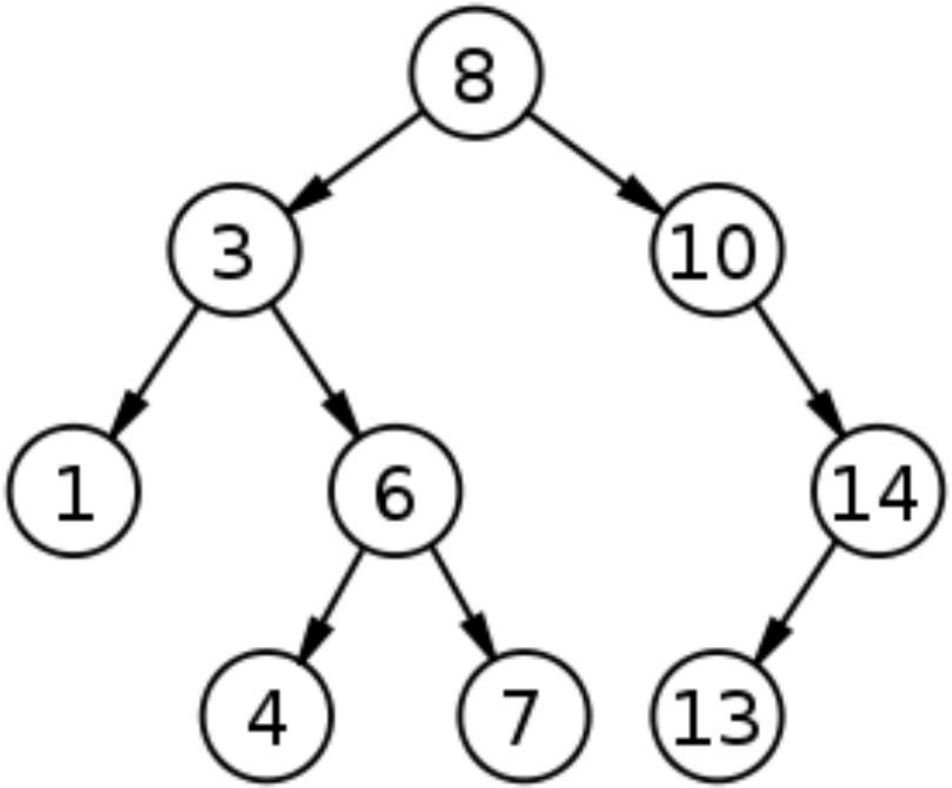
- ¿Quées árbol?
- ¿Qué es un árbol binario?

RECORRIDO EN INORDEN:

(izquierdo, raíz, derecho). Para recorrer un árbol binario no vacío en InOrden, hay que realizar las siguientes operaciones recursivamente en cada nodo:

- Atraviese el sub-árbol izquierdo
- Visite la raíz
- Atraviese el sub-árbol derecho

RECORRIDO EN INORDEN:



Secuencia:

1 – 3 – 4 – 6 – 7 – 8 – 10 – 13 – 14

DEFINIMOS LA FUNCIÓN INORDEN:

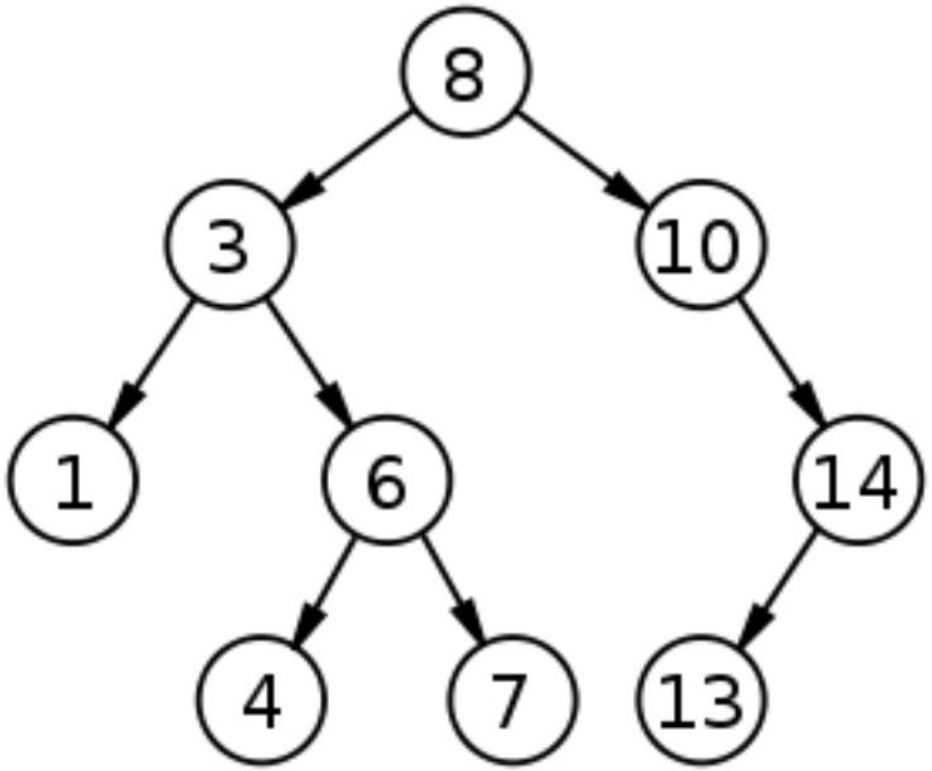
```
void InOrden(Nodo *arbol){
    if(arbol == NULL){
        return;
    }
    else{
        InOrden(arbol->izq);
        cout<<arbol->dato<<« - »;
        InOrden(arbol->der);
    }
}
```

RECORRIDO EN PREORDEN:

(raíz, izquierdo, derecho). Para recorrer un árbol binario no vacío en PreOrden, hay que realizar las siguientes operaciones recursivamente en cada nodo, comenzando con el nodo de raíz:

- Visite la raíz
- Atraviese el sub-árbol izquierdo
- Atraviese el sub-árbol derecho

RECORRIDO EN PREORDEN:



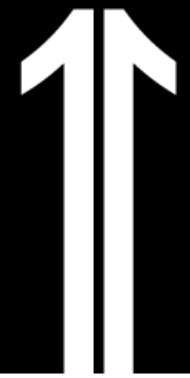
Secuencia:

8 – 3 – 1 – 6 – 4 – 7 – 10 – 14 – 13

DEFINIMOS LA FUNCIÓN PREORDEN:

```
void preOrden(Nodo *arbol){
    if(arbol == NULL){
        return;
    }
    else{
        cout<<arbol->dato<<« - »;
        preOrden(arbol->izq);
        preOrden(arbol->der);
    }
}
```

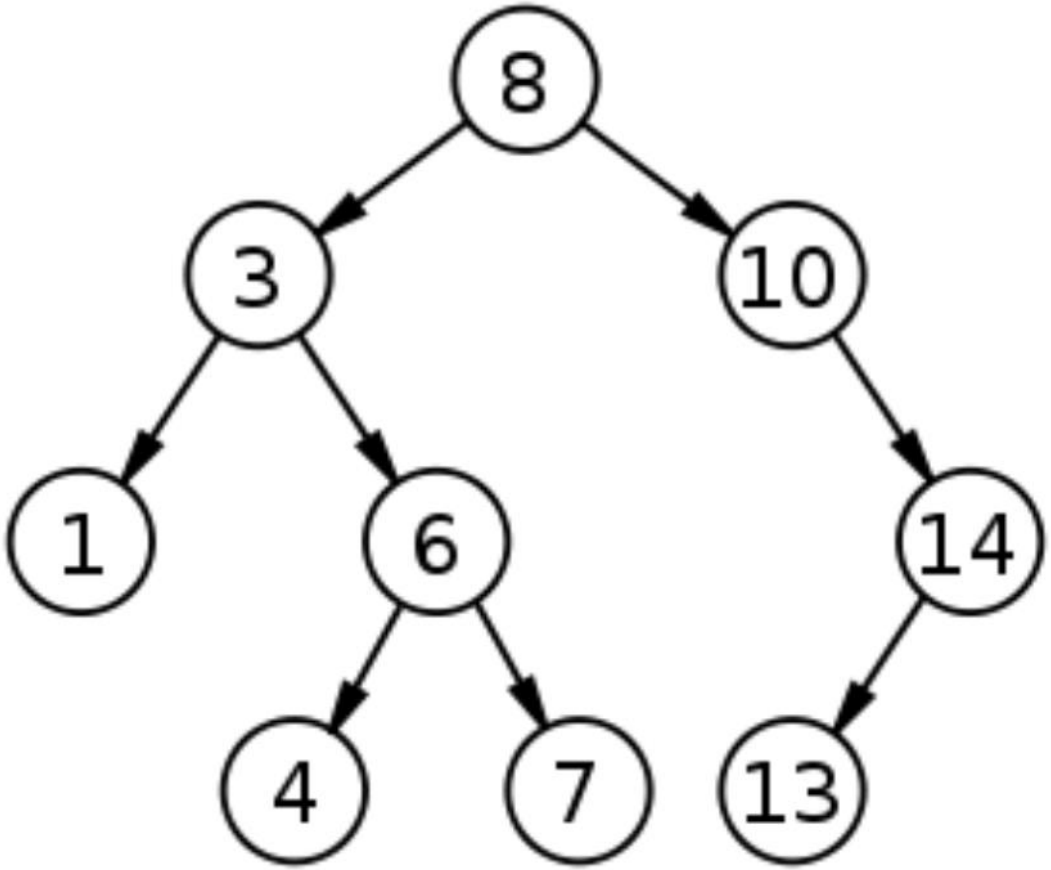
RECORRIDO EN POSTORDEN:



(izquierdo, derecho, raíz). Para recorrer un árbol binario no vacío en PostOrden, hay que realizar las siguientes operaciones recursivamente en cada nodo:

- Atraviese el sub–árbol izquierdo
- Atraviese el sub–árbol derecho
- Visite la raíz

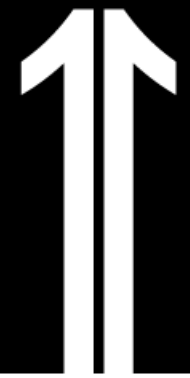
RECORRIDO EN POSTORDEN:



Secuencia:

1 – 4 – 7 – 6 – 3 – 13 – 14 – 10 – 8

DEFINIMOS LA FUNCIÓN POSTORDEN:



```
void postOrden(Nodo *arbol){
    if(arbol == NULL){
        return;
    }
    else{
        postOrden(arbol->izq);
        postOrden(arbol->der);
        cout<<arbol->dato<<« - »;
    }
}
```

Semana 12

GRAFOS

PRESENTACIÓN DE LA SESIÓN
Logro de la Sesión y Temario

Al término de la sesión, el estudiante aprende algoritmos para grafos, como representarlos y usarlos con eficacia.

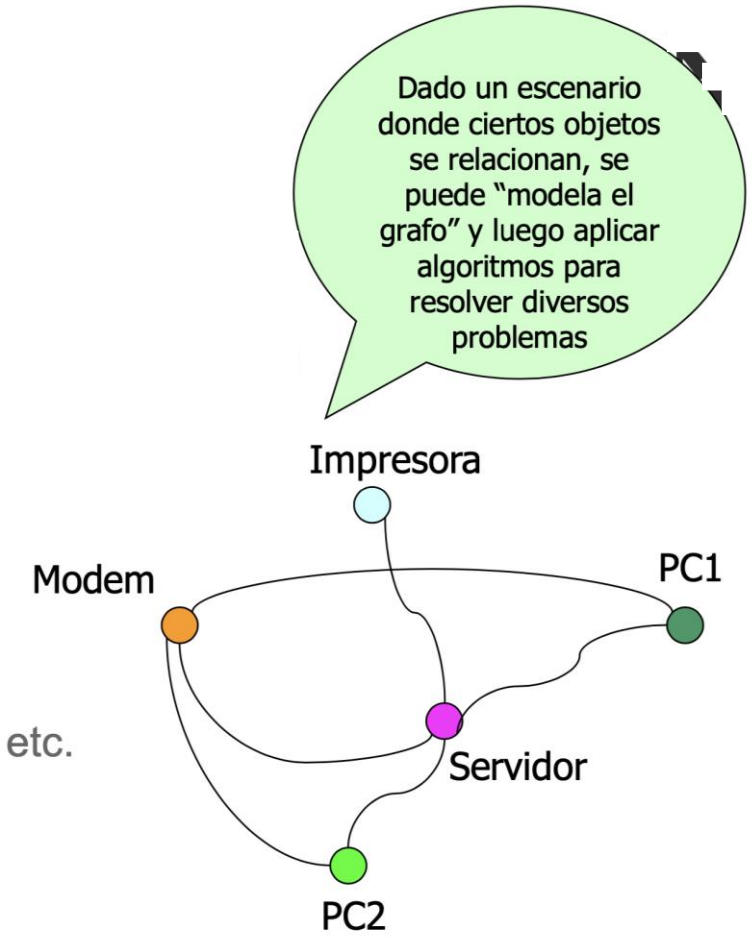
- Definiciones, grafos y grafos dirigidos, aplicaciones, representación, matriz de adyacencia, lista de adyacencia, matriz de costos. Recorrido: en amplitud (BFS), en profundidad (DFS), ordenamiento topológico y conectividad.

Reflexiona

- ¿Qué es un grafo?

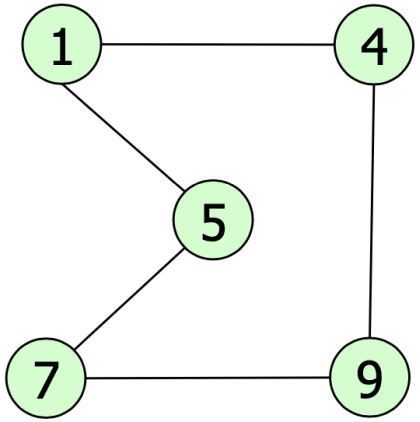
INTRODUCCION

- Los grafos son estructuras de datos
- Representan relaciones entre objetos
  - Relaciones arbitrarias, es decir
  - No jerárquicas
- Son aplicables en
  - Química
  - Geografía
  - Ing. Eléctrica e Industrial, etc.
  - Modelado de redes
    - De alcantarillado
    - Eléctricas
    - Etc.



DEFINICIÓN

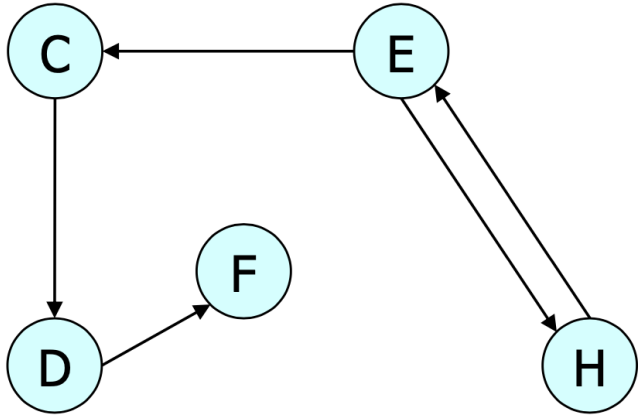
- Un grafo G = (V,A)
- V, el conjunto de vértices o nodos
  - Representan los objetos
- A, el conjunto de arcos
  - Representan las relaciones



V = {1, 4, 5, 7, 9}

A= {(1,4), (5,1), (7,9), (7,5), (4,9), (4,1), (1,5), (9,7), (5, 7), (9,4)}

TIPOS DE GRAFOS

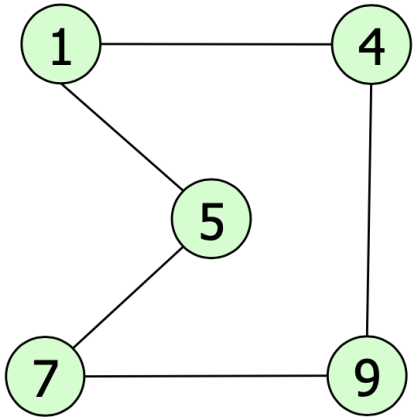


V = {C, D, E, F, H}

A= {(C,D), (D,F), (E,H), (H,E), (E,C)}

- Grafos no dirigidos
  - Si los pares de nodos de los arcos
  - No son ordenados Ej.: u-v

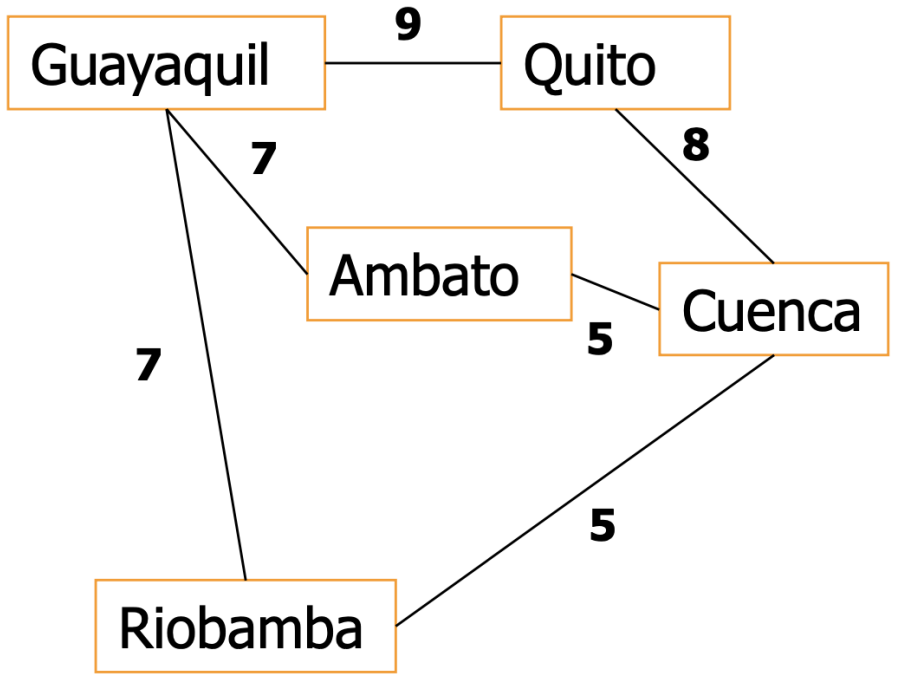
- Grafos dirigidos
  - Si los pares de nodos que forman arcos
  - Son ordenados. Ej.: (u->v)



Grafo del ejemplo anterior

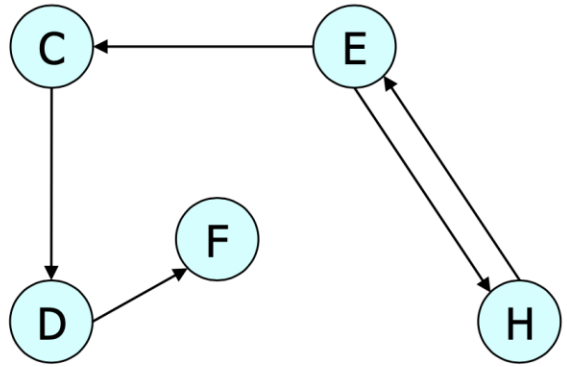
OTROS CONCEPTOS

- Arista
  - Es un arco de un grafo no dirigido
- Vertices adyacente
  - Vertices unidos por un arco
- Factor de Peso
  - Valor que se puede asociar con un arco
  - Depende de lo que el grafo represente
  - Si los arcos de un grafo tienen F.P.
    - Grafo valorado



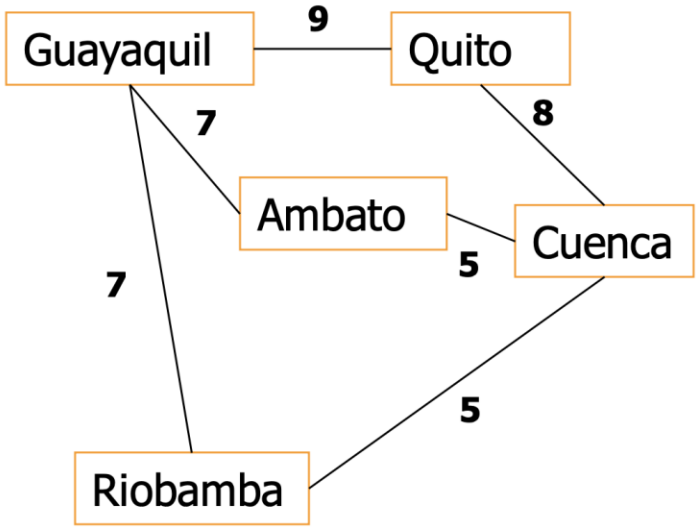
GRADOS DE UN NODO

- En Grafo No Dirigido
  - Grado(V)
    - Numero de aristas que contiene a V



Gradoent(D) = 1 y Gradsal(D) = 1

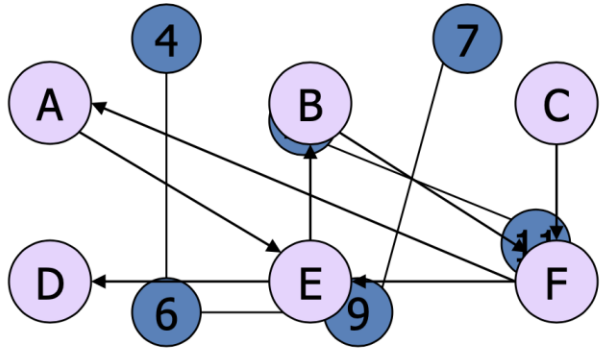
Grado(Guayaquil) = 3



- En Grafo Dirigido
  - Grado de entrada, Graden(V)
    - *Numero de arcos que llegan a V*
  - Grado de Salida, Gradsal(V)
    - *Numero de arcos que salen de V*

CAMINOS

- Definicion
  - Un camino P en un grafo G, desde V_0 a V_n
  - Es la secuencia de n+1 vertices
  - Tal que $(V_i, V_{i+1}) \in A$ para $0 \leq i \leq n$



Camino entre A y A

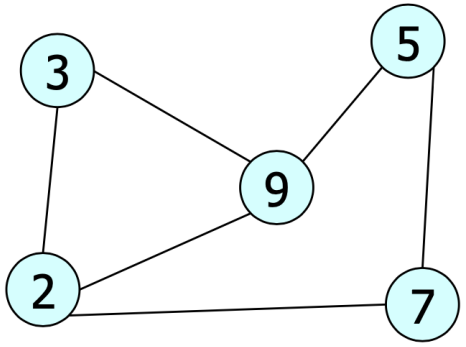
P = {A, E, B, F, A}

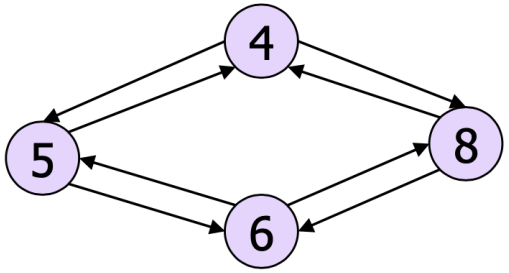
Longitud: 4 – 4ciclo

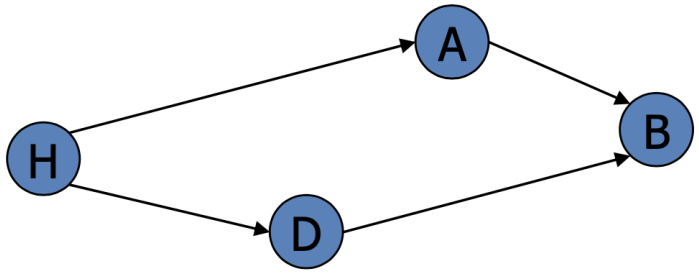
- Longitud de camino
  - El numero de arcos que lo forman
- Camino Simple
  - Todos los nodos que lo forman son distintos
- Ciclo
  - Camino simple cerrado de long. >= 2
  - Donde $V_0 = V_n$

CONECTIVIDAD

- Grafo No Dirigido
  - Conexo
    - Existe un camino entre cualquier par de nodos

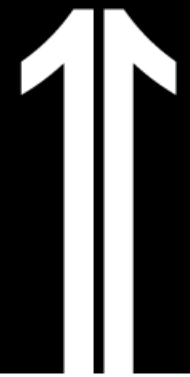






- Grafo Dirigido
  - Fuertemente Conexo
    - *Existe un camino entre cualquier par de nodos*
  - Conexo
    - *Existe una cadena entre cualquier par de nodos*

TDA GRAFO



- Datos
  - Vertices y
  - Arcos(relacion entre vertices)
- Operaciones
  - void AñadirVertice(Grafo G, Vertice V)
    - Añadir un nuevo vertice
  - void BorrarVertice(Grafo G, Generico clave)
    - Eliminar un vertice existente
  - void Union(Grafo G, Vertice V1, Vertice V2)
    - Unir dos vertices
  - Void BorrarArco(Grafo G, Vertice V1, Vertice V2)
    - Eliminar un Arco
  - bool EsAdyacente(Grafo G, Vertice V1, Vertice V2)
    - Conocer si dos vertices son o no adyacentes

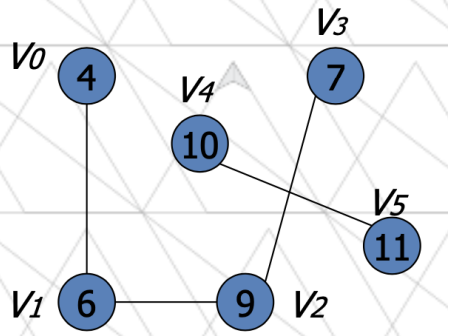
MATRIZ DE ADYACENCIA

Si el grafo fuese valorado, en vez de 1, se coloca el factor de peso

MATRIZ DE ADYACENCIA

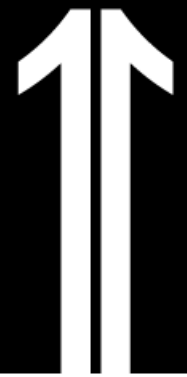
- Dado un Grafo G = (V, A)
- Sean los Vertices V = {V0, V1, … Vn}
  - Se pueden representar por ordinales 0,1,..n
- Como representar los Arcos?
  - Estos son enlaces entre vertices
- Puede usarse una matriz

$$a_{ij}\begin{cases}1, \text{si hay arco} & (Vi,Vj)\\ 0, \text{si no hay arco} & (Vi,Vj)\end{cases}$$



$$\begin{array}{c} \\ V_0 \\ V_1 \\ V_2 \\ V_3 \\ V_4 \\ V_5 \end{array}\begin{array}{c} \begin{array}{cccccc} V_0 & V_1 & V_2 & V_3 & V_4 & V_5 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

EL TIPO DE DATO



- Los Vertices
  - Se definen en un Arreglo
- Los Arcos
  - Se definen en una Matriz

```
#define MAX 20
typedef int [MAX][MAX] MatrizAdy;
typdef Generico[MAX] Vertices;
typedef struct Grafo{
        Vertices V;
        MatrizAdy A;
        int nvertices;
        bool Dirigido;
};
```

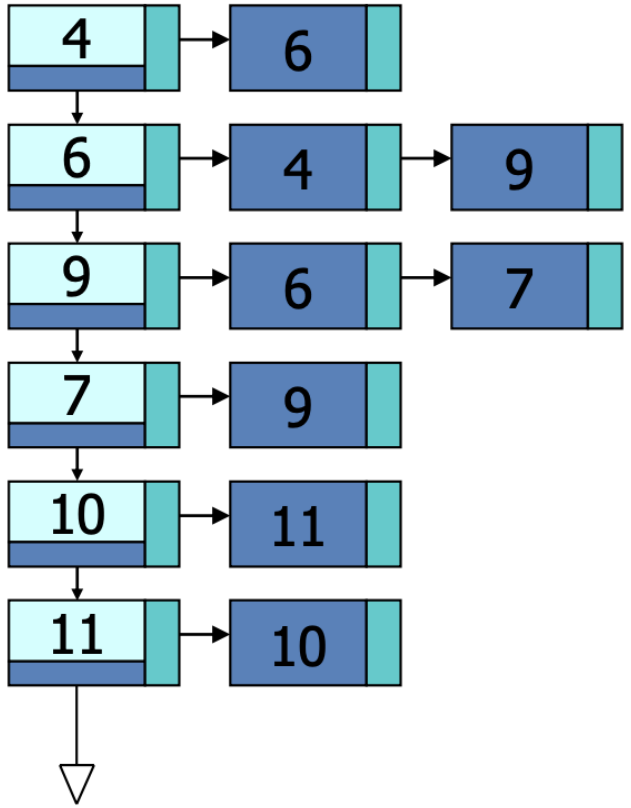
UNIR VERTICE

```
void Union(Grafo G, int v1, int v2){

    G->A[v1][v2] = 1;
    if(!G->dirigido)
        G->A[v2][v1] = 1;
}
```

LISTA DE ADYACENCIA

- Si una matriz
  - Tiene muchos vertices y
  - Pocos arcos
  - La Matriz de Adyacencia
    - Tendra demasiados ceros
    - Ocupara mucho espacio
- Los vertices
  - Pueden formar una lista, no un vector
- Los arcos
  - Son relaciones entre vertices
  - Se pueden representar con una lista x cada vertice



EL TIPO DE DATO

- Cadaverticetiene
  - Contenido
  - Siguiente
  - Una lista de adyacencia
- Cada nodo en la lista de adyacencia
  - Peso del arco
  - Siguiente
  - Una referencia al vertice(arco)

```
typedef struct Vertice{
        Generico contenido;
        LSE *LA;
};
typedef Vertice *Arco;
typedef struct Grafo{
        LSE LVertices;
        bool dirigido;
};
```

ALGUNAS IMPLEMENTACIONES

```
void Union(Grafo G, Vertice V1, Vertice V2){
      LSE_InsertarNodoFin(V1->Larcos, LSE_CrearNodo(V2));
      if(!G->dirigido)
            LSE_InsertarNodoFin(V2->Larcos,
                        LSE_CrearNodo(V1);
}
```

EJERCICIO

- Complete la implementacion de las operaciones del grafo con lista de adyacencia

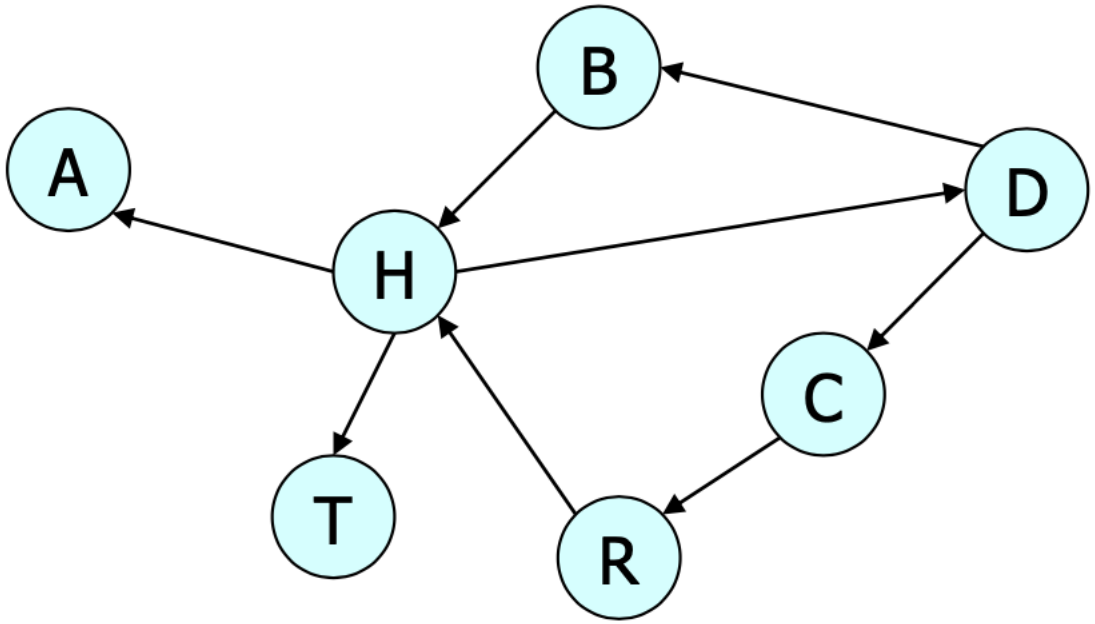
RECORRIDOS DEL GRAFO

- Se busca
  - Visitar todos los nodos posibles
  - Desde un vertice de partida D
  - Cualquiera
- Existe dos posibles recorridos
  - En Anchura y
  - En Profundidad

RECORRIDO EN ANCHURA

- Encolar vertice de partida
- Marcarlo como “visitado”
- Mientras la cola no este vacia • Desencolar vertice W
    - Mostrarlo
    - Marcar como visitados
        - Los vertices adyacentes de W
        - Que no hayan sido ya visitados
    - Encolarlos

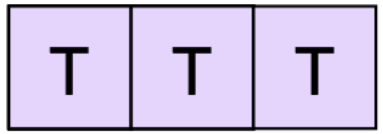
EJEMPLO



Se Muestra:

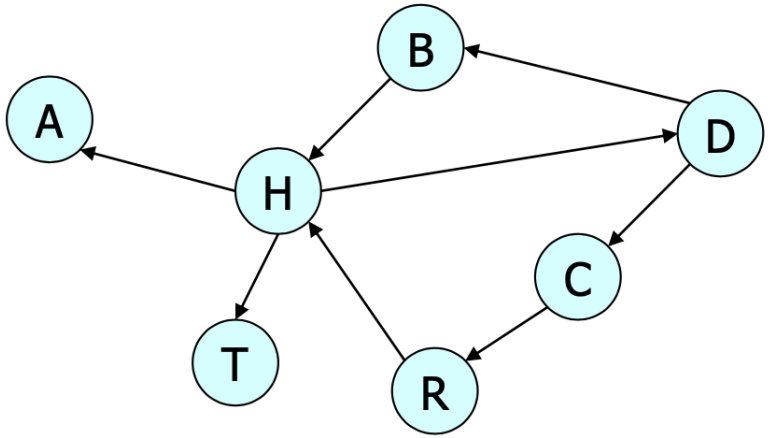
D B C H R A T

Cola



RECORRIDO EN PROFUNDIDAD

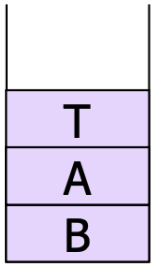
- Marcar vertice origen V como visitado
- Recorrer en Profundidad
  - Cada vertice adyacente de V
  - Que no haya sido visitado
- Ejemplo



Se Muestra:

D C R H T A B

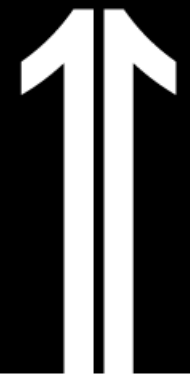
Pila



ACTIVIDAD

- Escriba la implementacion del recorrido en profundidad de un grafo a partir de un vertice inicial

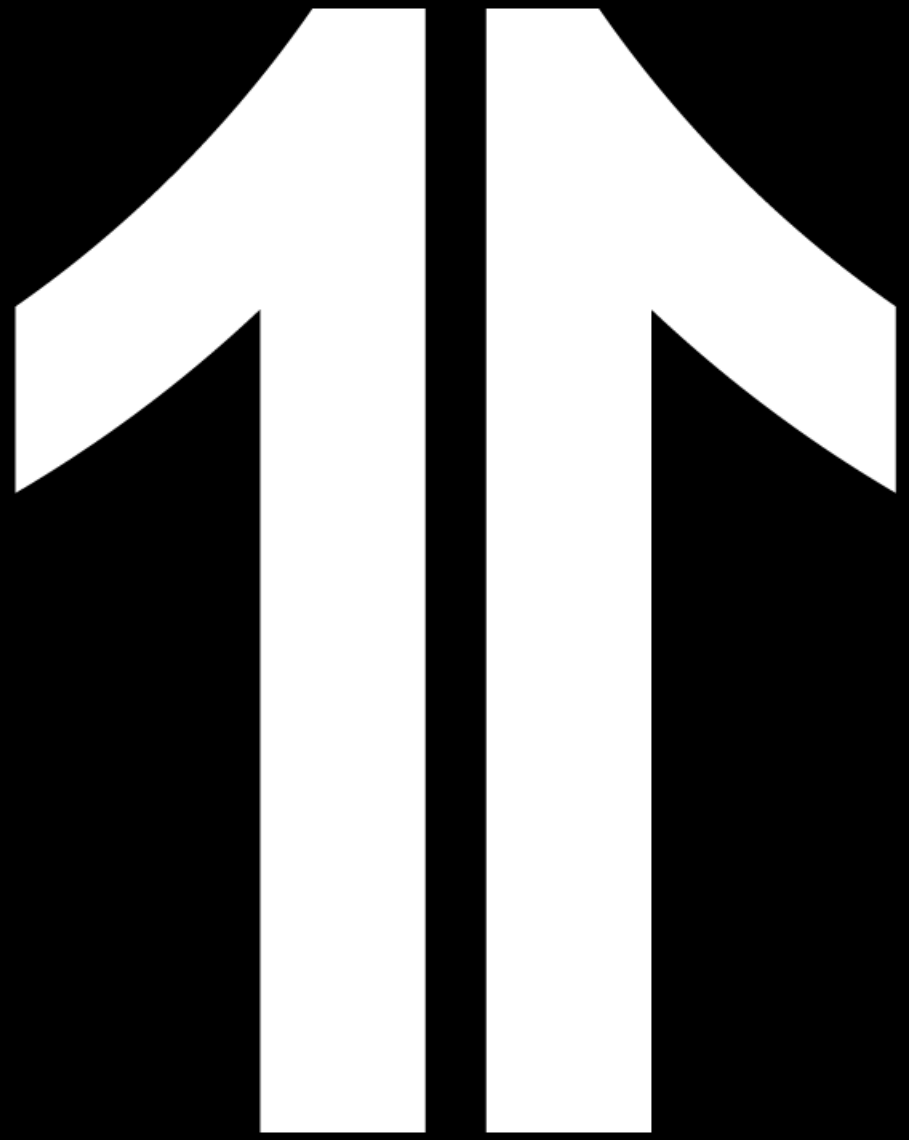
CONCLUSIONES



- Los árboles binarios son estructuras que permiten una gestión eficiente de la información.
- Las partes de un árbol binario son raíz, hoja y tallo.
- En el caso de la raíz y hojas, se generan de manera recursiva.
- Las operaciones que pueden realizarse son: recorrido, inorden, postorden, preorden.

BIBLIOGRAFÍA BÁSICA

- Ceballos Sierra, F. Microsoft C#: Curso de Programación (2a.ed.) 2014 https://elibronet.eu1.proxy.openathens.net/es/lc/upnorte/titulos/106417
- Cesar Liza Avila; Estructura de datos con C/C++



GRACIAS


UPN
UNIVERSIDAD
PRIVADA
DEL NORTE