TSP - PROGRAMACION II

ESTRUCTURAS DE DATOS NO LINEALES

Árboles y Grafos

ESTRUCTURA DE DATOS NO LINEALES

- Árboles
  - Binarios
  - Multi Camino o N-arios
- Grafos
  - Dirigidos
  - No Dirigidos
  - Conexos
  - No conexos

TSP - PROGRAMACION II

ÁRBOLES

ESTRUCTURA DE DATOS NO LINEALES - Árboles

Árbol

- Un árbol es una estructura de datos ampliamente formada por un conjunto de nodos conectados.
- Un nodo es la unidad sobre la que se construye el árbol y puede tener cero o más nodos hijos conectados a él.
- Se dice que un nodo **A** es padre de un nodo **B** si existe un enlace desde a hasta **B** (en ese caso, también decimos que **B** es hijo de **A**).

Árbol Binario

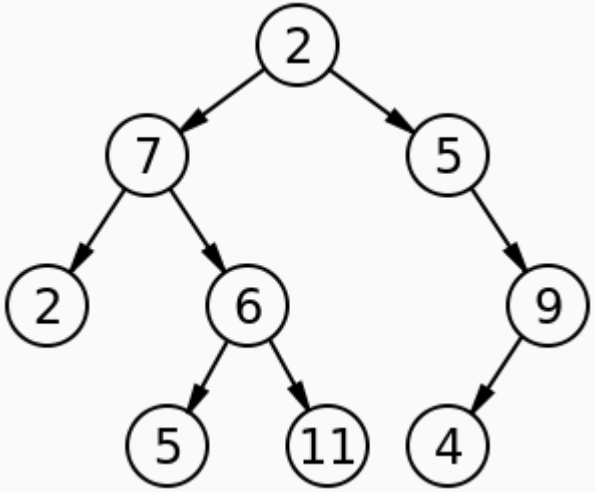
- Es una estructura de datos en la cual cada nodo puede tener un hijo izquierdo y un hijo derecho. No pueden tener más de dos hijos

Árbol N-ario

- Posee un grado **g** mayor a dos, donde cada nodo de información del árbol tiene un máximo de **g** hijos.

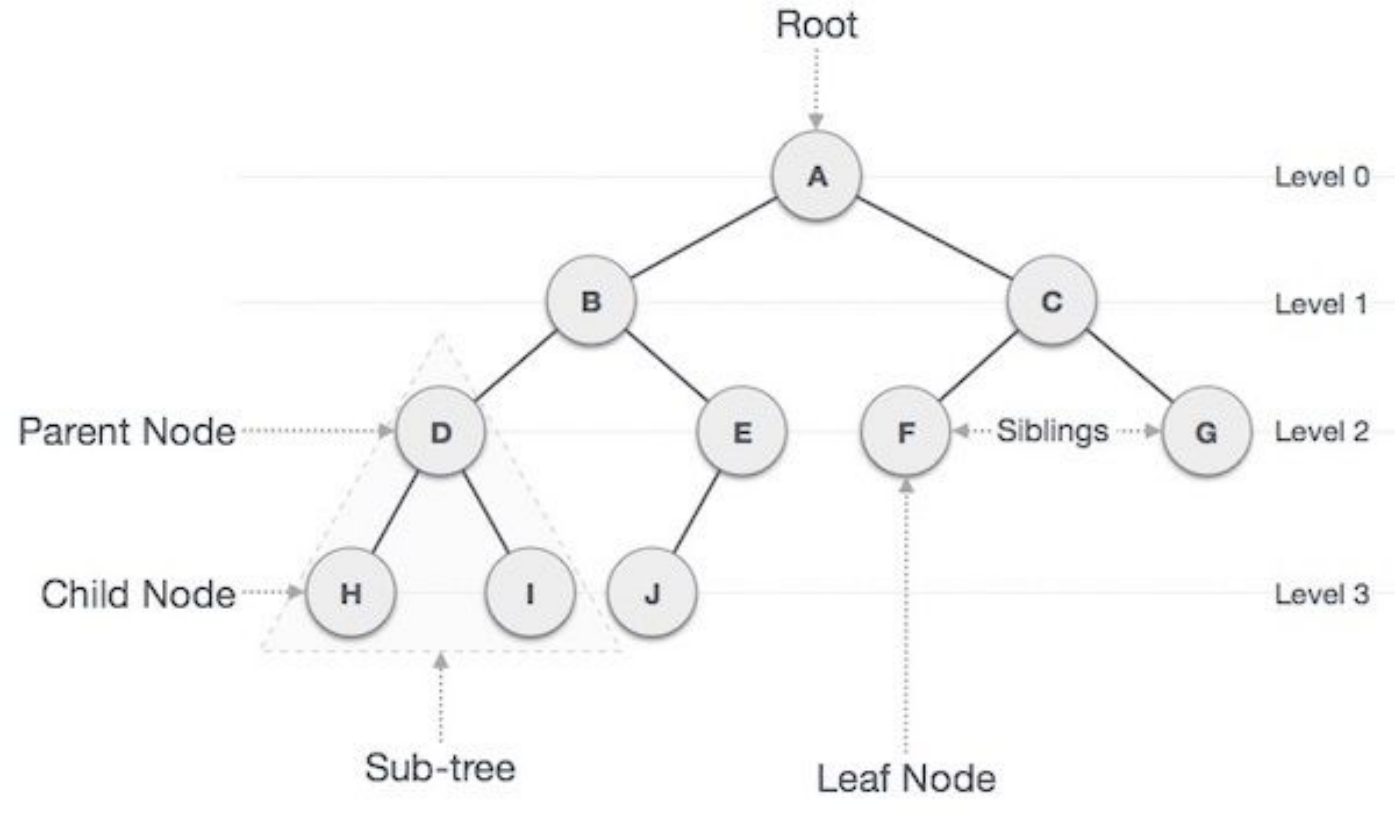
ESTRUCTURA DE DATOS NO LINEALES - Terminologías de Árboles

Terminologías utilizadas en árboles



- **Raíz:** El nodo superior del árbol.
- **Padre**: Nodo con hijos.
- **Hijo**: Nodo descendiente de otro nodo.
- **Hermanos**: Nodos que comparten el mismo padre.
- **Hojas**: Nodos sin hijos.
- **Grado**: Es el número de descendientes directos de un nodo.
- **Longitud**: El camino más largo de un nodo raíz hasta uno terminal.
- **Profundidad**: Es el número de nodos que se encuentran entre él y la raíz.

ESTRUCTURA DE DATOS NO LINEALES - Terminologías de Árboles



ESTRUCTURA DE DATOS NO LINEALES - Estructura de Árboles

Nodo árbol binario

```
record Node {
   data // El dato almacenado en el nodo
   Node NodeIzq   // apunta al primer nodo de la izquierda.
   Node NodeDer   // apunta al último nodo de la derecha.
}
```

Nodo árbol N-ario

```
record Node {
   data // El dato almacenado en el nodo
   Node[...] NodeHijos   // apunta a una lista de nodos hijos.
}
```

Búsqueda en profundidad (DFS o Depth First Search)

- Es un algoritmo que permite recorrer todos los nodos de un grafo o árbol de manera ordenada, pero no uniforme.
- Su funcionamiento consiste en ir expandiendo todos y cada uno de los nodos que va localizando, de forma recurrente, en un camino concreto.
- Cuando ya no quedan más nodos que visitar en dicho camino, regresa (Backtracking), de modo que repite el mismo proceso con cada uno de los hermanos del nodo ya procesado.

ESTRUCTURA DE DATOS NO LINEALES - Tipos de Búsquedas de Árboles

Búsqueda en anchura (BFS - Breadth First Search)

- Es un algoritmo para recorrer o buscar elementos en un árbol el cual comienza en la raíz y se exploran todos los hijos de este nodo.
- A continuación para cada uno de los hijos se exploran sus respectivos hijos, y así hasta que se recorra todo el árbol.
- Formalmente, BFS es un algoritmo de búsqueda sin información, que expande y examina todos los nodos de un árbol sistemáticamente para buscar una solución.
- El algoritmo no usa ninguna estrategia heurística.

ESTRUCTURA DE DATOS NO LINEALES - Tipos de Recorridos de Árboles

Pre-orden

1. Visite la raíz
2. Atraviese el sub-árbol izquierdo
3. Atraviese el sub-árbol derecho

```
preorden(nodo)
  si nodo == nulo entonces retorna
  imprime nodo.valor
  preorden(nodo.izquierda)
  preorden(nodo.derecha)
```

In-orden

1. Atraviese el sub-árbol izquierdo
2. Visite la raíz
3. Atraviese el sub-árbol derecho

```
inorden(nodo)
  si nodo == nulo entonces retorna
  inorden(nodo.izquierda)
  imprime nodo.valor
  inorden(nodo.derecha)
```

Post-orden

1. Atraviese el sub-árbol izquierdo
2. Atraviese el sub-árbol derecho
3. Visite la raíz

```
postorden(nodo)
  si nodo == nulo entonces retorna
  postorden(nodo.izquierda)
  postorden(nodo.derecha)
  imprime nodo.valor
```

ESTRUCTURA DE DATOS NO LINEALES - Tipos de Recorridos de Árboles

Profundidad primero

- Secuencia de recorrido de preorden (raíz, izquierda, derecha)

  F, B, A, D, C, E, G, I, H
- Secuencia de recorrido de inorden (izquierda, raíz, derecha)

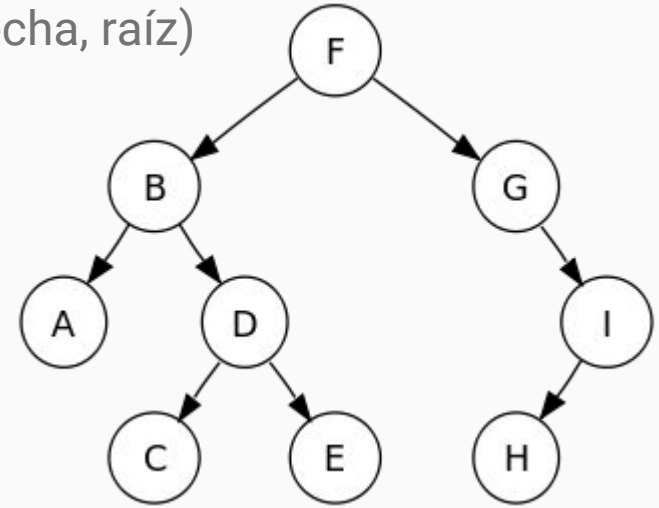
  A, B, C, D, E, F, G, H, I
- Secuencia de recorrido de postorden (izquierda, derecha, raíz)

  A, C, E, D, B, H, I, G, F

Anchura primero

- Secuencia de recorrido de orden por nivel

  F, B, G, A, D, I, C, E, H



ESTRUCTURA DE DATOS NO LINEALES - Insertar Nodos en Árboles

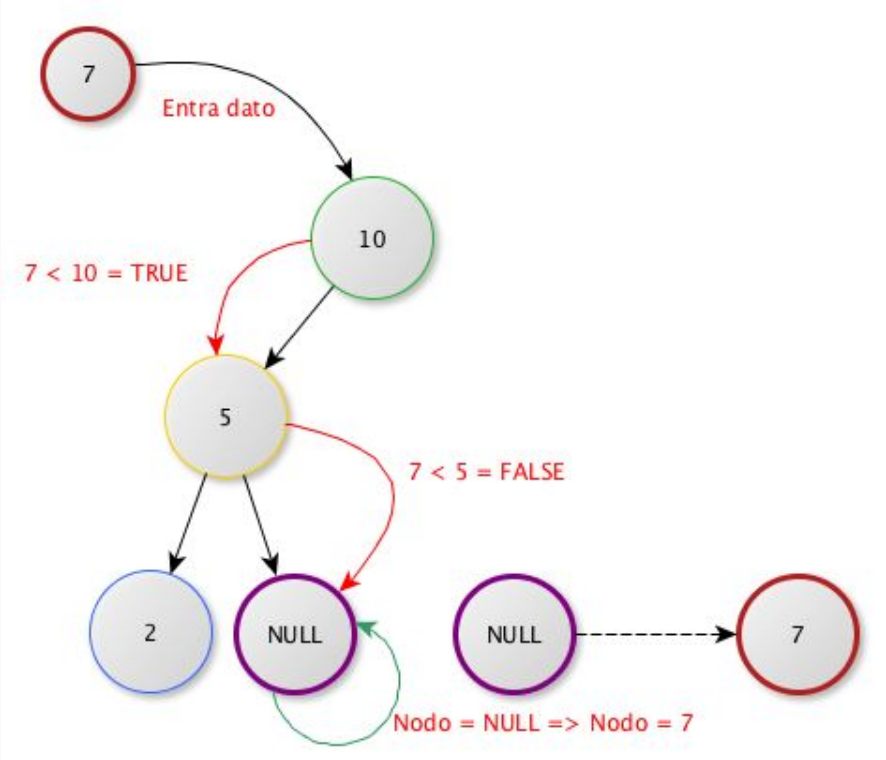
Pasos para insertar nodos

1. Se toma el dato a ingresar X
2. Partiendo de la raíz preguntamos: Nodo == null (o no existe) ?
3. En caso afirmativo X pasa a ocupar el lugar del nodo y ya hemos ingresado nuestro primer dato.
4. En caso negativo preguntamos: X < Nodo
5. En caso de ser menor pasamos al Nodo de la IZQUIERDA del que acabamos de preguntar y repetimos desde el paso 2 partiendo del Nodo al que acabamos de visitar
6. En caso de ser mayor pasamos al Nodo de la DERECHA y tal cual hicimos con el caso anterior repetimos desde el paso 2 partiendo de este nuevo Nodo.

ESTRUCTURA DE DATOS NO LINEALES - Insertar Nodos en Árboles

Es un proceso *RECURSIVO* en el cual al final por más grande que sea el árbol el dato a entrar ocupará un lugar.

Ejemplo: Insertar el valor **7**.



Eliminar Nodos en Árboles

No es un proceso sencillo, debido a que pueden se presentan diferentes situaciones:

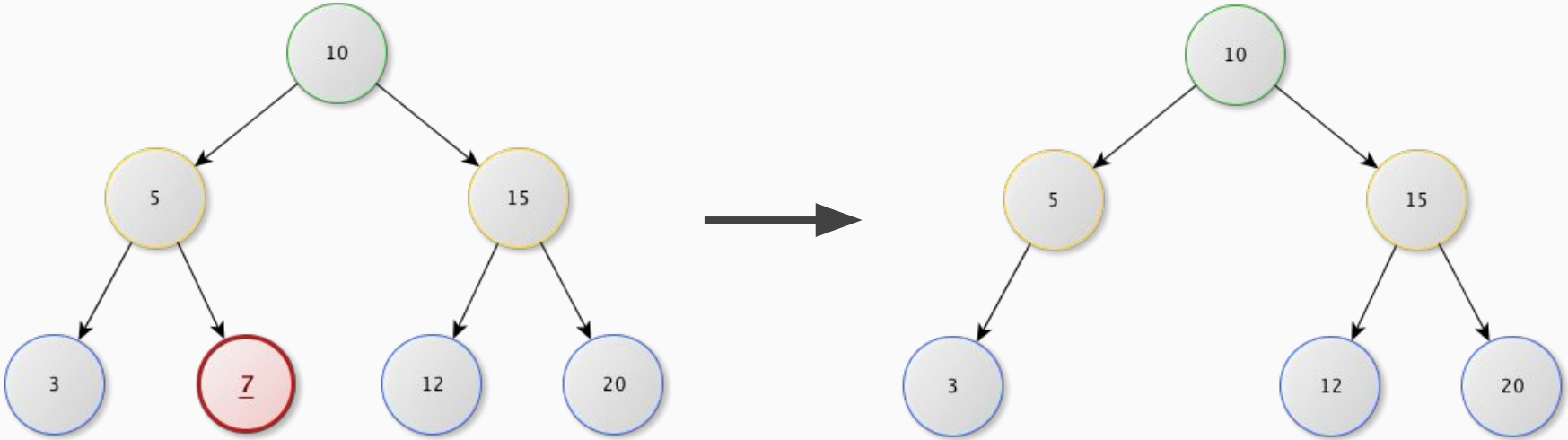
1. Borrar un Nodo sin hijos
2. Borrar un Nodo con un subárbol hijo
3. Borrar un Nodo con dos subárboles hijos

ESTRUCTURA DE DATOS NO LINEALES - Eliminar Nodos en Árboles

Pasos para Borrar un Nodo sin hijos

El caso más sencillo, lo único que hay que hacer es borrar el nodo y establecer el apuntador de su padre a nulo.

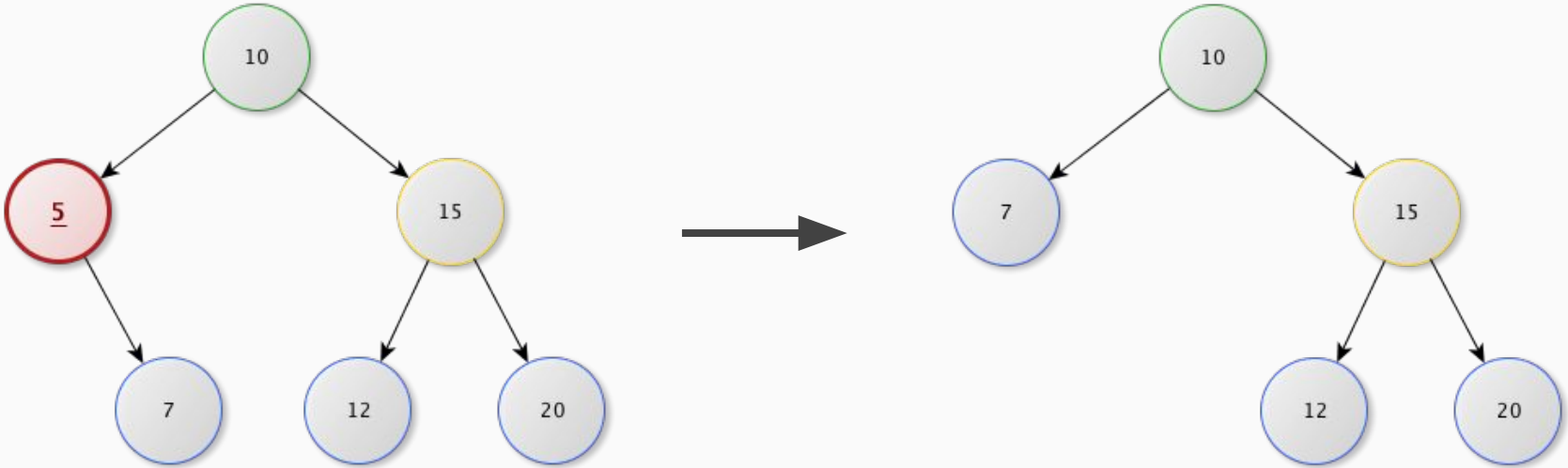
Ejemplo: Eliminar el valor **7**



Pasos para Borrar un Nodo con un subárbol hijo

Este caso tampoco es muy complicado, únicamente tenemos que borrar el Nodo y el subárbol que tenía pasa a ocupar su lugar.

Ejemplo: Eliminar el valor **5**



Pasos para Borrar un Nodo con dos subárboles hijos

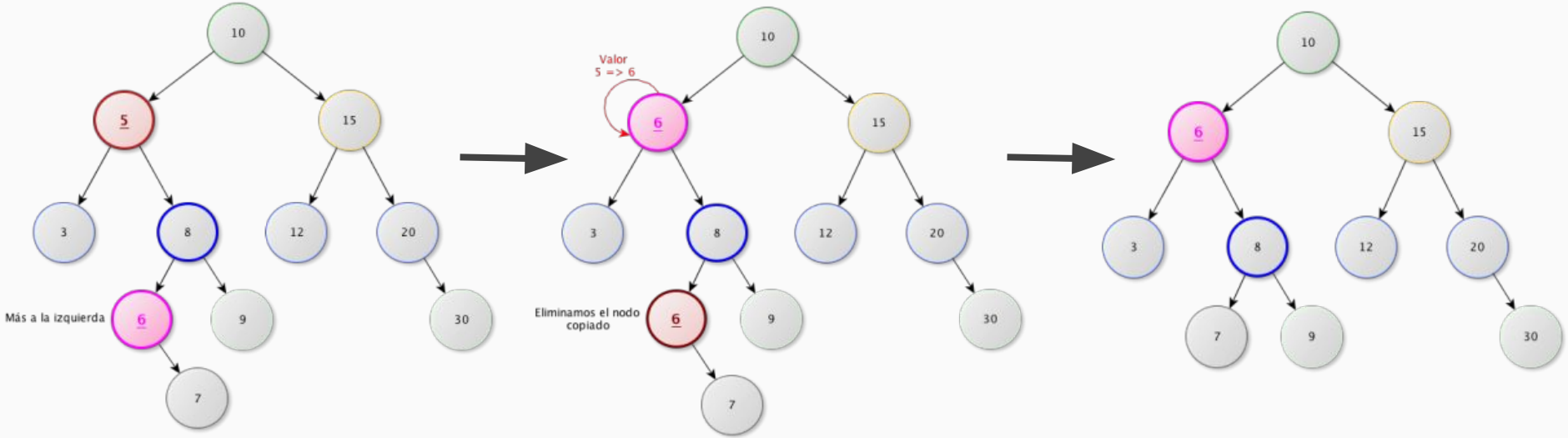
Este es un caso algo complejo:

1. Tenemos que tomar el **hijo derecho** del Nodo que queremos eliminar y recorrer hasta el **hijo más a la izquierda** (hijo izquierdo y si este tiene hijo izquierdo repetir hasta llegar al último nodo a la izquierda)
2. Reemplazamos el valor del nodo que queremos eliminar por el nodo que encontramos (el hijo más a la izquierda), el nodo que encontramos por ser el más a la izquierda es imposible que tenga nodos a su izquierda pero sí que es posible que tenga un subárbol a la derecha
3. Para terminar solo nos queda proceder a eliminar este nodo de las formas que conocemos (caso 1, caso 2) y tendremos la eliminación completa.

ESTRUCTURA DE DATOS NO LINEALES - Eliminar Nodos en Árboles

Pasos para Borrar un Nodo con dos subárboles hijos

Ejemplo: Eliminar el valor **5**

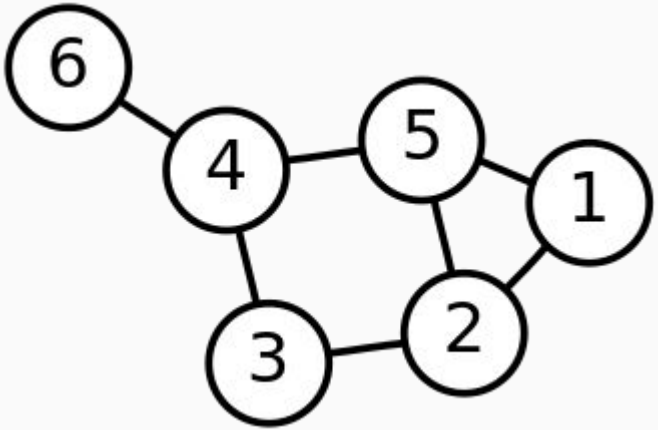


TSP - PROGRAMACION II

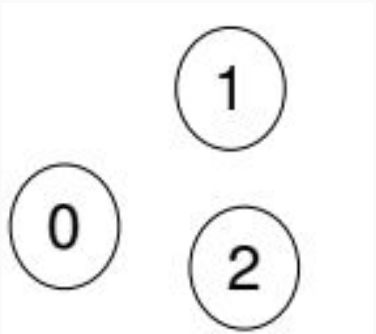
GRAFOS

Grafos

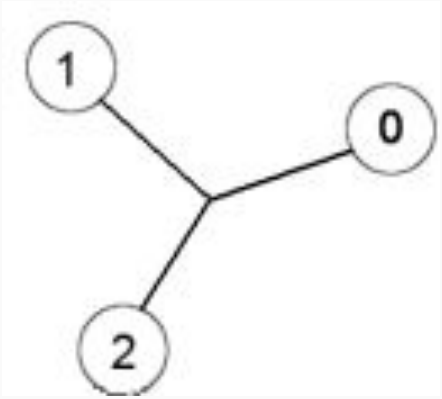
- Un grafo es una representación de un grupo de objetos conectados donde algunos pares están conectados por aristas.
- Es un conjunto de objetos llamados vértices o nodos unidos por enlaces llamados aristas o arcos, que permiten representar relaciones binarias entre elementos de un conjunto.



Esto es un grafo



Esto NO es un grafo



Esto NO es un grafo

ESTRUCTURA DE DATOS NO LINEALES - Grafos - Definiciones

Definición

Un grafo {G} es un par ordenado {G=(V,E)}, donde:

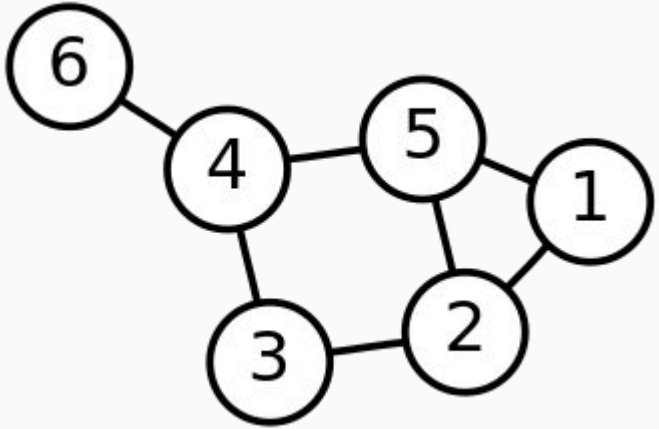
- {V} es un conjunto de vértices o nodos, y
- {E} es un conjunto de aristas o arcos, que relacionan estos nodos.
- {E} se encuentra incluido en V x V.
- Normalmente {V} suele ser finito.

Ejemplo

La imagen es una representación del siguiente grafo:

- V:={1,2,3,4,5,6}
- E:={{1,2},{1,5},{2,3},{2,5},{3,4},{4,5},{4,6}}

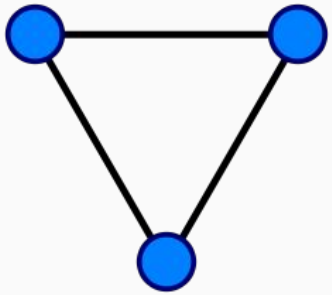
El hecho que el vértice 1 sea adyacente con el vértice 2 puede ser denotado como 1 ~ 2.



ESTRUCTURA DE DATOS NO LINEALES - Grafos - Definiciones

Grafos No Dirigidos

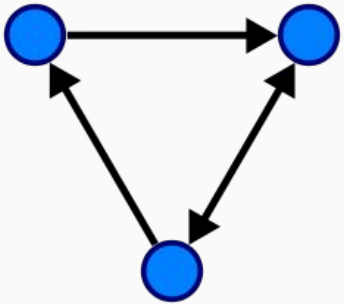
- Es un grafo en donde las aristas no tienen dirección.



(a , b) = (b , a)

Grafos Dirigidos

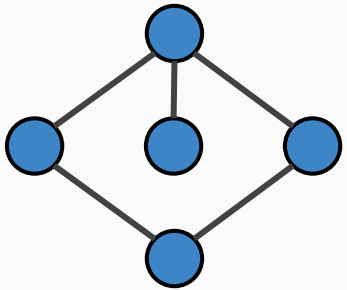
- Es un grafo en donde las aristas tienen dirección.



(a , b) <> (b , a)

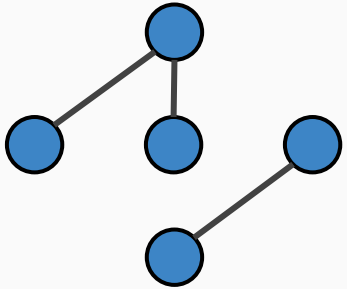
Grafos Conexos

- Es un grafo en donde cada par de vértices están conectados, existiendo una conexión entre todos ellos.

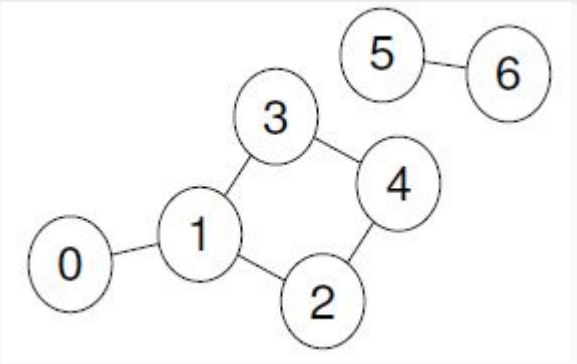


Grafos No Conexos

- Es un grafo en donde existen al menos dos subgrafos conexos.



Terminologías utilizadas en Grafos

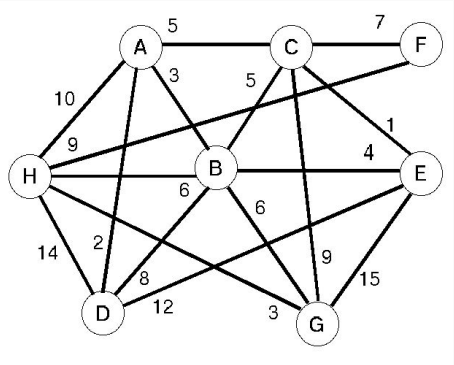


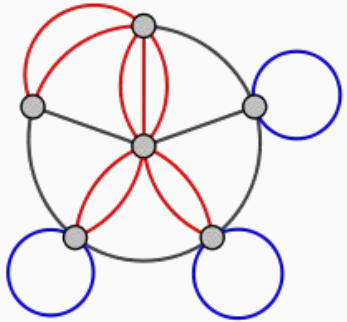
Para un grafo G = {(V, E)}

- **Orden:** Se llama orden del grafo **G** a su número de vértices **V**.
- **Grado:** El grado de un vértice o nodo **V** es igual al número de arcos que lo tienen como extremo.
- **Nodos adyacentes:** Vértices conectados por una arista.
- **Aristas/Arcos adyacentes:** Aristas/Arcos con un nodo en común.
- **Bucle:** es una arista que relaciona al mismo nodo; es decir, una arista donde el nodo inicial y el nodo final coinciden.
- **Camino:** Sucesión de arcos adyacentes tal que el vértice final de cada arco coincide con el inicial del siguiente.
- **Circuito:** Camino que empieza y acaba en el mismo vértice.

Tipos de Grafos

- **Grafo Etiquetado:** Cada arista y/o vértice tiene asociada una etiqueta o valor.
- **Grafo Ponderado o Grafo con pesos:** Grafo etiquetado en el que existe un valor numérico asociado a cada arista o arco.
- **Multigrafo:** Grafo en el que se permite que entre dos vértices exista más de una arista o arco.
- **Árbol:** Grafo conexo que no tiene ciclos.



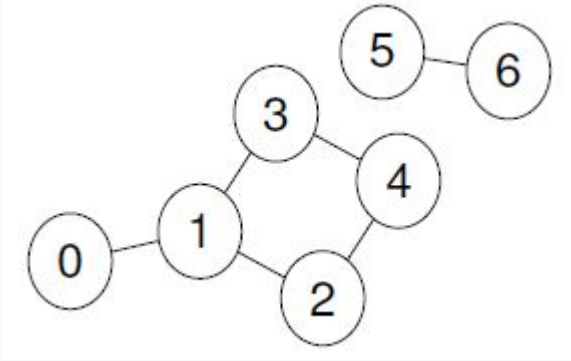


ESTRUCTURA DE DATOS NO LINEALES - Como representar Grafos

Pares de Adyacencia

- Una lista de pares donde cada par representa una arista.
- Se representa usando un arreglo donde cada posición corresponde a una arista que une un par de vértices.

1	0 1
2	1 2
3	2 4
4	3 4
5	1 3
6	5 6



Ventajas:

- Es simple de parsear.

Desventajas:

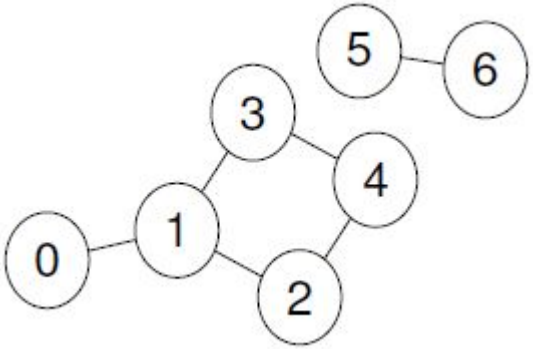
- Hacer cualquier operación es difícil y lento.
- Por lo general solo se usan para la entrada en los problemas de grafos.

ESTRUCTURA DE DATOS NO LINEALES - Como representar Grafos

Matrices de Adyacencia

- Una matriz donde cada el elemento en (i; j) indica si hay una arista entre el nodo i y el nodo j.
- Se representa utilizando una matriz cuadrada
- de V x V.
- Se necesita conocer la cantidad de vértices, luego se completa la matriz.
- Es ideal para cuando el número de vértices |V| es pequeño y hay que encontrar relaciones entre pares de nodos muy seguido, o cuando el grafo es denso.

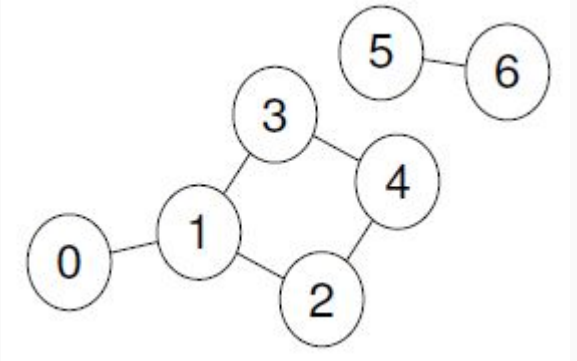
	0	1	2	3	4	5	6
0	0	1	0	0	0	0	0
1	1	0	1	1	0	0	0
2	0	1	0	0	1	0	0
3	0	1	0	0	1	0	0
4	0	0	1	1	0	0	0
5	0	0	0	0	0	0	1
6	0	0	0	0	0	1	0



ESTRUCTURA DE DATOS NO LINEALES - Como representar Grafos

Matrices de Adyacencia

	0	1	2	3	4	5	6
0	0	1	0	0	0	0	0
1	1	0	1	1	0	0	0
2	0	1	0	0	1	0	0
3	0	1	0	0	1	0	0
4	0	0	1	1	0	0	0
5	0	0	0	0	0	0	1
6	0	0	0	0	0	1	0



Ventajas:

- Saber si dos nodos están conectados tiene complejidad simple, una lectura en la matriz.
- Se pueden hacer operaciones sobre esta matriz para encontrar propiedades del grafo.
- Una matriz donde cada el elemento en (i; j) indica si hay una arista entre el nodo i y el nodo j.

Desventajas:

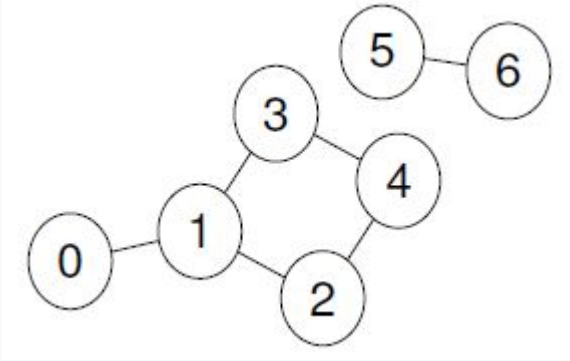
- Buscar los nodos adyacentes a otro nodo tiene mayor complejidad, sin importar cuántos nodos adyacentes tenga el primer nodo.
- Recorrer todo el grafo implica gran complejidad.
- Si la matriz es grande, ocupa mayor memoria.

ESTRUCTURA DE DATOS NO LINEALES - Como representar Grafos

Listas de Adyacencia

- Una lista con |V| elementos, donde el elemento número i tiene la lista de nodos adyacentes al nodo i.
- Para cada nodo de |V| se tiene una lista de aristas que parten de ese nodo.
- Las listas están guardadas en un array de nodos cabecera.
- Si el grafo no es dirigido entonces cada arista será representada dos veces.
- Por lo general se usa en grafos esparsos, cuando |V| es grande, o cuando los multi-ejes son significativos.

0	1
1	2 0 3
2	1 4
3	4 1
4	3 2
5	6
6	5



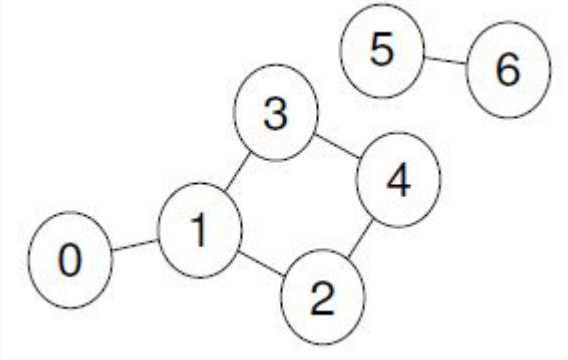
ESTRUCTURA DE DATOS NO LINEALES - Como representar Grafos

Listas de Adyacencia

0	1
1	2 0 3
2	1 4
3	4 1
4	3 2
5	6
6	5

Ventajas:

- Encontrar la cantidad de nodos adyacentes a cierto nodo tiene menor complejidad o es lineal con respecto a las otras opciones.
- Recorrer todo el grafo tiene gran complejidad, ya que requiere recorrer todos los vértices y luego sus nodos adyacentes internos.



Desventajas:

- Saber si dos nodos están conectados es lineal al grado de alguno de los dos nodos.

Búsqueda en profundidad (DFS o Depth First Search)

- Es un algoritmo que permite recorrer todos los nodos de un grafo o árbol de manera ordenada, pero no uniforme.
- Su funcionamiento consiste en ir expandiendo todos y cada uno de los nodos que va localizando, de forma recurrente, en un camino concreto.
- Cuando ya no quedan más nodos que visitar en dicho camino, regresa (Backtracking), de modo que repite el mismo proceso con cada uno de los hermanos del nodo ya procesado.

ESTRUCTURA DE DATOS NO LINEALES - DFS o Depth First Search

```
DFS(grafo G)
   PARA CADA vértice u ∈ V[G] HACER
           estado[u] ← NO_VISITADO
           padre[u] ← NULO
   tiempo ← 0
   PARA CADA vértice u ∈ V[G] HACER
           SI estado[u] = NO_VISITADO ENTONCES
                   DFS_Visitar(u,tiempo)
```

```
DFS_Visitar(nodo u, int tiempo)
   estado[u] ← VISITADO
   tiempo ← tiempo + 1
   d[u] ← tiempo
   PARA CADA v ∈ Vecinos[u] HACER
           SI estado[v] = NO_VISITADO ENTONCES
                   padre[v] ← u
                   DFS_Visitar(v,tiempo)
   estado[u] ← TERMINADO
   tiempo ← tiempo + 1
   f[u] ← tiempo
```

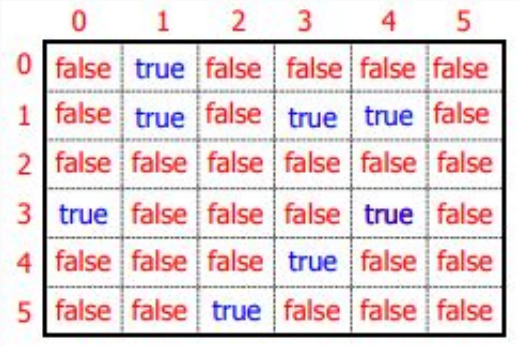
Búsqueda en anchura (BFS - Breadth First Search)

- Es un algoritmo para recorrer o buscar elementos en un grafo el cual comienza eligiendo algún nodo como elemento raíz y se exploran todos los vecinos de este nodo.
- A continuación para cada uno de los vecinos se exploran sus respectivos vecinos adyacentes, y así hasta que se recorra todo el grafo.
- Formalmente, BFS es un algoritmo de búsqueda sin información, que expande y examina todos los nodos de un grafo sistemáticamente para buscar una solución.
- El algoritmo no usa ninguna estrategia heurística.

ESTRUCTURA DE DATOS NO LINEALES - BFS - Breadth First Search

```
BFS(grafo G, nodo_fuente s)
{
   // recorremos todos los vértices del grafo inicializándolos a NO_VISITADO,
   // distancia INFINITA y padre de cada nodo NULL
   for u ∈ V[G] do
   {
      estado[u] = NO_VISITADO;
      distancia[u] = INFINITO; /* distancia infinita si el nodo no es alcanzable */
      padre[u] = NULL;
   }
   estado[s] = VISITADO;
   distancia[s] = 0;
   padre[s] = NULL;
   CrearCola(Q); /* nos aseguramos que la cola está vacía */
   Encolar(Q, s);
   while !vacia(Q) do
   {
      // extraemos el nodo u de la cola Q y exploramos todos sus nodos adyacentes
      u = extraer(Q);
      for  v ∈ adyacencia[u]  do
      {
         if estado[v] == NO_VISITADO then
         {
              estado[v] = VISITADO;
              distancia[v] = distancia[u] + 1;
              padre[v] = u;
              Encolar(Q, v);
         }
      }
   }
}
```

Matriz de Adyacencia

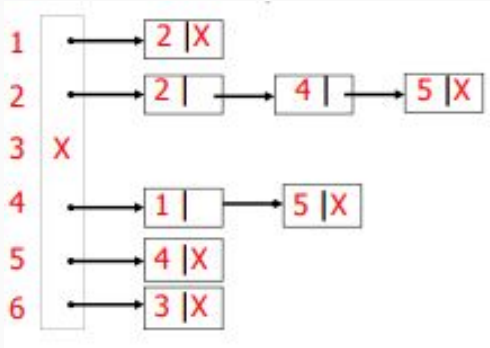


	0	1	2	3	4	5
0	false	true	false	false	false	false
1	false	true	false	true	true	false
2	false	false	false	false	false	false
3	true	false	false	false	true	false
4	false	false	false	true	false	false
5	false	false	true	false	false	false

- Será necesario inicializar la matriz con valores
- Que indiquen que la misma está vacía (0 o false)
- A medida que se indican los pares de vértices adyacentes, se procederá a completar en la matriz, en la fila y columna correspondiente, que existe una arista entre el par de vértices, por ejemplo con un 1 o true.
- En el caso de que el Grafo posea pesos en sus aristas, se podrá insertar dicho valor de forma similar al punto anterior.
- Para eliminar conexiones entre vértices, sólo implica volver a setear el valor en la matriz con 0 o false.

ESTRUCTURA DE DATOS NO LINEALES - Grafos - Insertar y Eliminar Vértices

Listas de Adyacencia



- Será necesario inicializar un arreglo de Listas de Nodos (TAD Lista)
- El arreglo no tendrá valores iniciales (las Listas están vacías)
- Cada posición del arreglo corresponderá a un vértice del grafo, por lo tanto será necesario crear una función de mapeo entre los índices del arreglo y los vértices del grafo. (arreglo[1] => Nodo 1, arreglo[2] => Nodo 2, etc)
- A medida que se indican los pares de vértices adyacentes, se procederá a buscar el índice del arreglo que corresponde con el vértice inicial del par, y luego agregar un nodo en la Lista con el valor del vértice final.
- Para eliminar conexiones entre vértices, será necesario eliminar los nodos de la Lista correspondiente al vértice inicial del par.