ESCUELA DE INGENIERÍA
UCN COQUIMBO

Listas generalizadas y Árboles

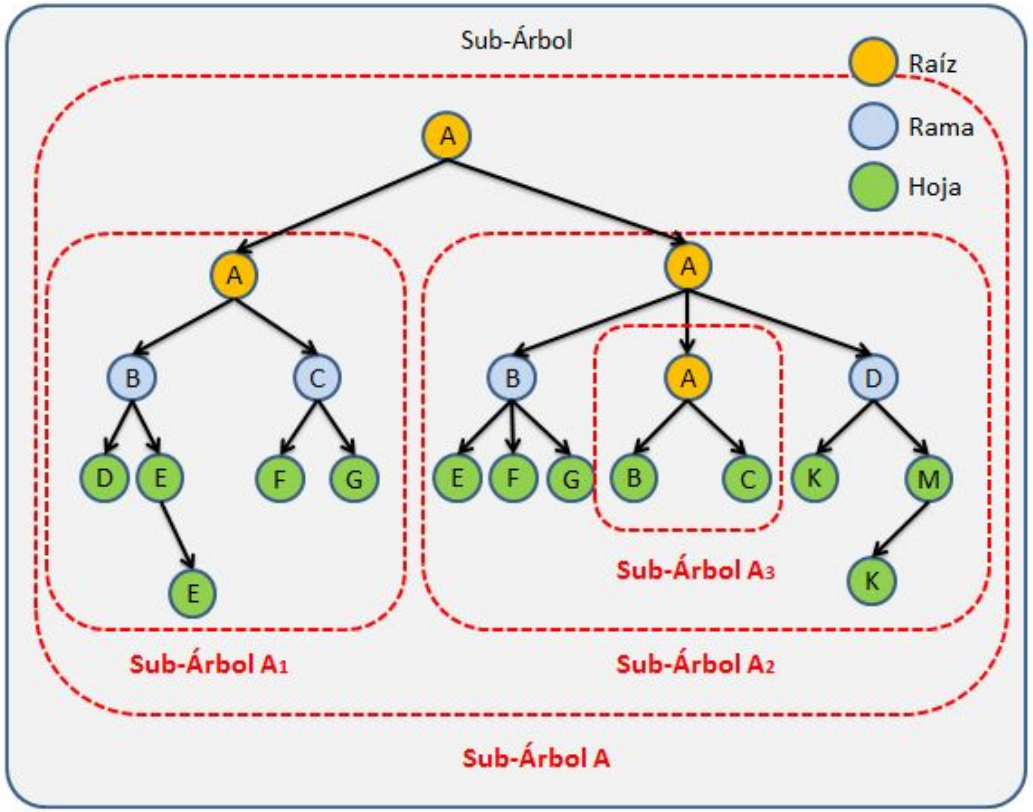
Juan Bekios Calfa

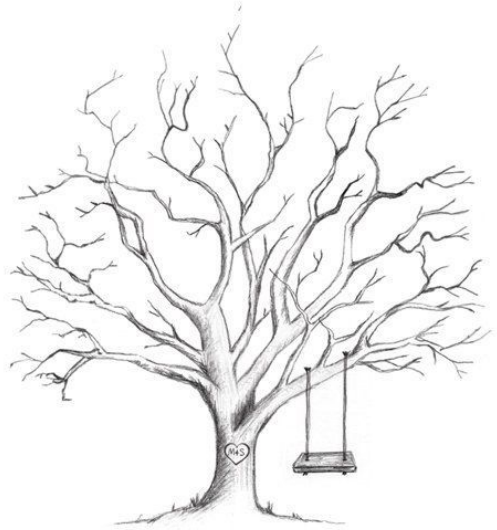
juan.bekios@ucn.cl

http://jbekios.ucn.cl

Contenidos

A. **Listas generalizadas**
B. Conceptos sobre árboles
C. Árboles binarios
D. Recorrido de árboles binarios
E. Árboles binarios enhebrados
F. Heaps
G. Árboles de búsqueda binarios
H. Árboles de búsqueda binarios balanceados
I. Famila de árboles B (B, B*, B+)





Listas Enlazadas Especializadas

Hasta ahora hemos visto el uso de listas enlazadas que guardan un valor como un entero u otro tipo de dato ya definido, pero qué pasa si deseamos crear una “customizable” LinkedList.

Ahora veremos template,los cuales funcionan como los *generics* de Java. Por ejemplo:

```
Vector<String> v = new Vector<String>();
Hashmap<Integer,String> hm = new HashMap<Integer,String>();
```

O Usando #include<vector>, y luego std::vector<int> myVector;

En C++, uno tiene el uso de template es de la siguiente manera template<class X>

Ejemplo de vector

```
#include<iostream>
#include<vector>
using namespace std;

int main(){
  vector<string> v;

  for (int i = 0; i < 10; ++i) {
    v.push_back(to_string(i));
  }

  for (int i = 0; i < 10; ++i) {
    cout<<v[i]<<endl;  //v.at(i)
  }

  return 0;
}
```

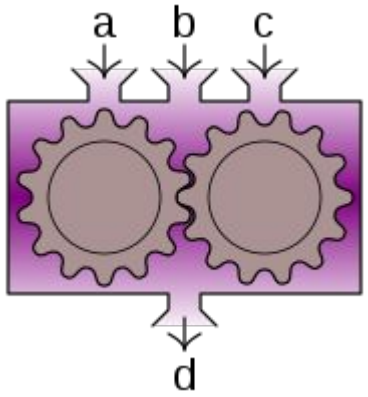
Vector que contiene String

Agregando al final

Se puede tratar como arreglo o objeto

Usando Template

- La función myMax puede ser adaptable a cualquier tipo de dato



Operación Ternaria

```
template <class T>
T myMax(T a, T b) {
    return (a > b)? a: b;
}

int main() {
    int a = myMax(5,6);
    double b = myMax(5.2,4.3);

    cout << a << " " << b << endl;
    return 0;
}
```

https://replit.com/@jbekios/TemplateMax

Ejemplo MyMax

```
#include<iostream>

using namespace std;

template<class T>
T myMax(T a, T b) {
    return (a > b)? a: b;
}

int main() {
  int a = myMax(5,6);
  double b = myMax(5.2, 5.7);
  string c = myMax("alo", "casa");

  cout << a << " " << b << endl;

  return 0;
}
```

¿Que pasa aquí?

Nodo & LinkedList con Template

Nodo

```
template <class T>
class Node {
  public:
  T element;
  Node<T> *next;

  Node(T x) {
    this->element = x;
    this->next = NULL;
  }

  void setNext(Node<T>* next) {
    this->next = next;
  }

  void show() {
    std::cout << this->element;
  }
};
```

Lista Enlazada

```
template <class T>
class LinkedList {
  Node<T> *first;

public:

  LinkedList() {
    first = NULL;
  }

  void push(T x) {
    Node<T> *node = new Node<T>(x);

    if (first == NULL) {
      first = node;
    } else {
      Node<T> *aux = first;
      while (aux->next != NULL) {
        aux = aux -> next;
      }
      aux->next = node;
    }
  }
```

main

```
int main() {
  LinkedList<double> *l = new LinkedList<double>();
  l->push(5);
  l->push(2);
  l->push(8.5);
  l->push(15);
  l->show();
  cout << endl;
  l->show();

  cout << "esta vacia? " << l->empty() << endl;
  delete l;
  return 0;
}
```

Friendships entre Node y LinkedList

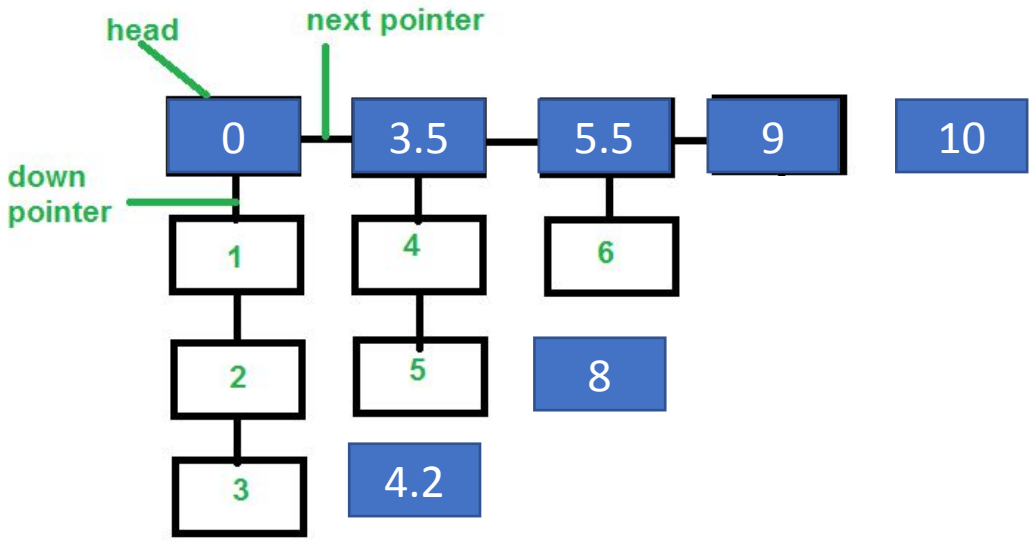
- Claramente Node tiene una relación especial LinkedList

- Por esa razón, para evitar usar muchos métodos, podemos usar `friend class` y así acceder a los métodos privados

```
template <class T> class LinkedList;

template <class T>
class Node {
  private:
    friend class LinkedList<T>;
    T element;
    Node<T> *next;

  public:
    Node(T x) {
      this->element = x;
      this->next = NULL;
    }
    void setNext(Node<T>* next) {
      this->next = next;
    }
    void show() {
      std::cout << this->element;
    }
};
```

“Cortina”: ListaEnlazada con Down

La lista contiene ahora un nodo down, la cual permite
que cada nodo tenga la posibilidad de tener una lista colgando



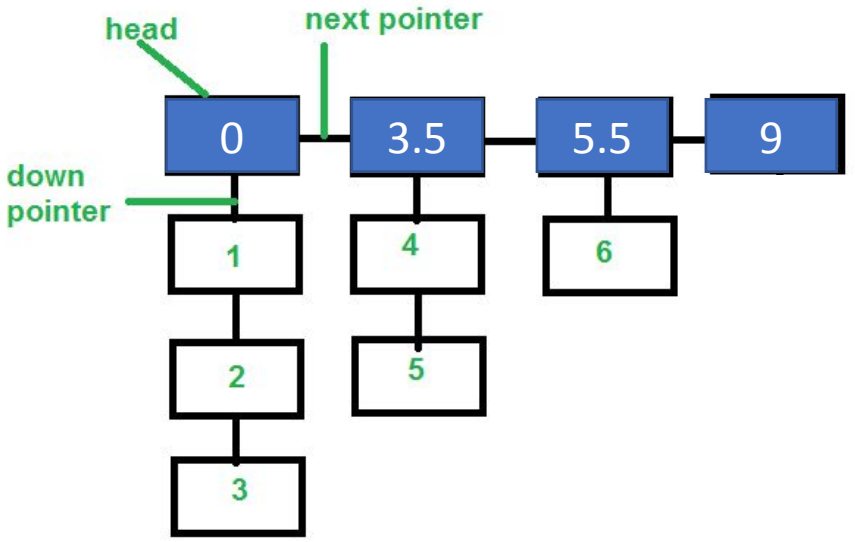
Si agrego el “8”, “4.2”, “10” ¿Dónde queda?

https://repl.it/@PaulLeger/clase93

Invariante de “Cortina”

Al insertar, se agrega en “down” si el elemento es menor al siguiente que se encuentra en la cabecera

Si lo datos son insertados de manera uniforme (**no incremental**), el tiempo de inserción y búsqueda se vuelve más eficiente ¿Por qué? ¿Cuál su peor caso?



Estructura de “Cortina”

```
template <class T>
class Node {
public:
  T element;
  Node<T> *next;

  Node(T x) {
    element = x;
    next = NULL;
  }
  void show() {
    std::cout << element;
  }
};
```

Node de Lista

```
template <class T>
class HeadNode: public Node<T> {
public:
  Node<T> *down;
  HeadNode(T x):Node<T>(x) {
    down = NULL;
  }
};
```

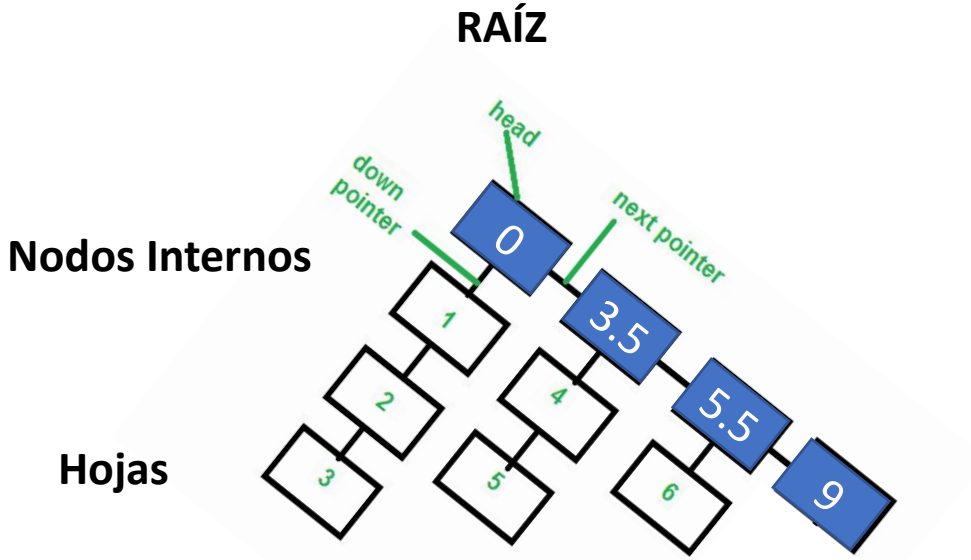
Node de cabecera Lista

```
template <class T>
class LinkedList {
  HeadNode<T> *first;

public:

  LinkedList() {
    first = NULL;
  }
```

Lista tiene un nodo de cabecera

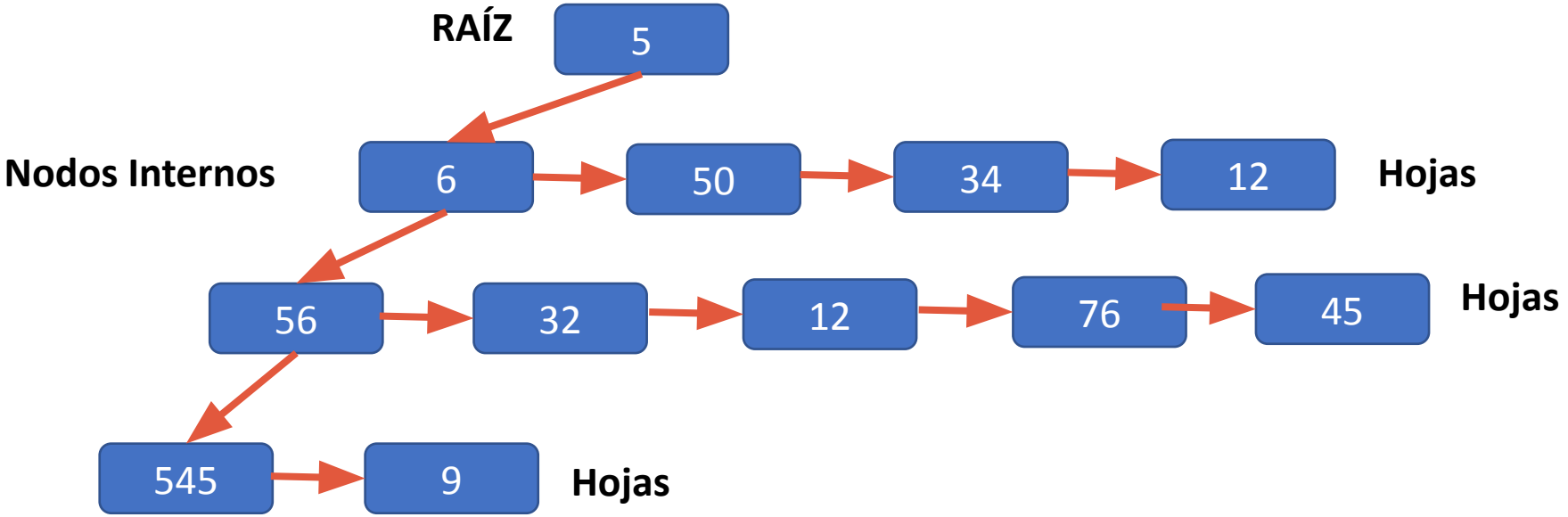
TDAs

Hasta ahora, hemos visto TDAs como Pilas, Cola y Listas con ciertas propiedades. Aparte de las listas, existen otros tipos TDAs muy famosos como árboles



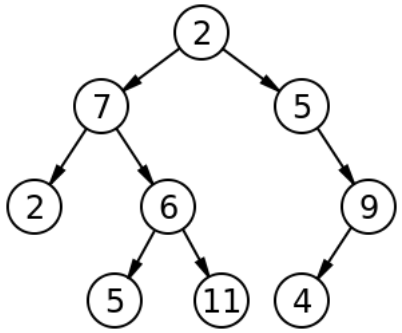
TDAs árboles

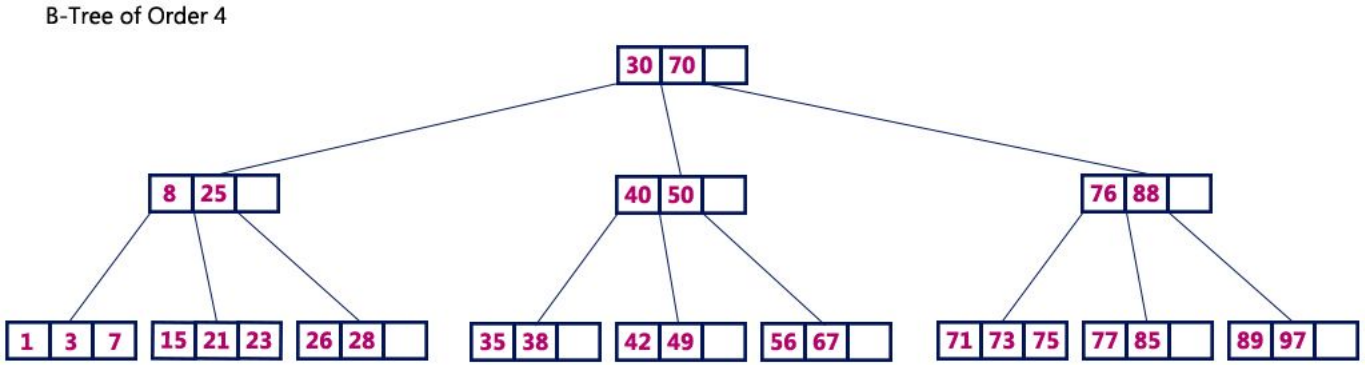
Un árbol tiene una raíz, nodos internos y hojas; que contienen cualquier tipo de datos. Existen muchos tipos de árboles, que incluso pueden tener otros árboles (un bosque).

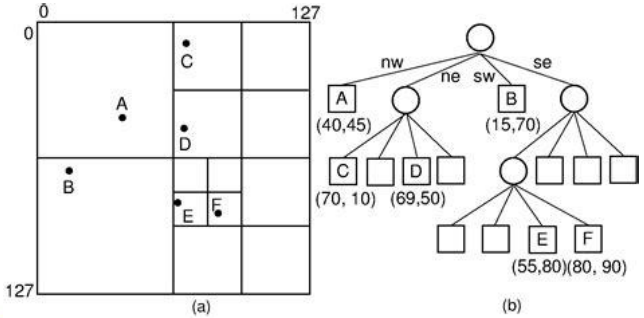


Tipos de árboles

Binary tree, Binary search tree, AVL, Huffman tree, B-Tree, B*-tree, red-black tree, etc



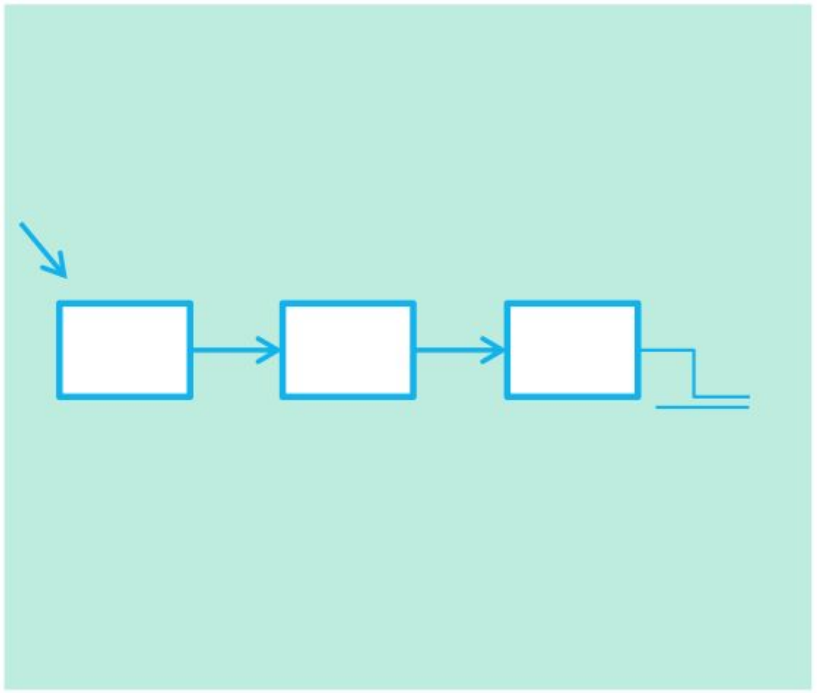




Contenidos

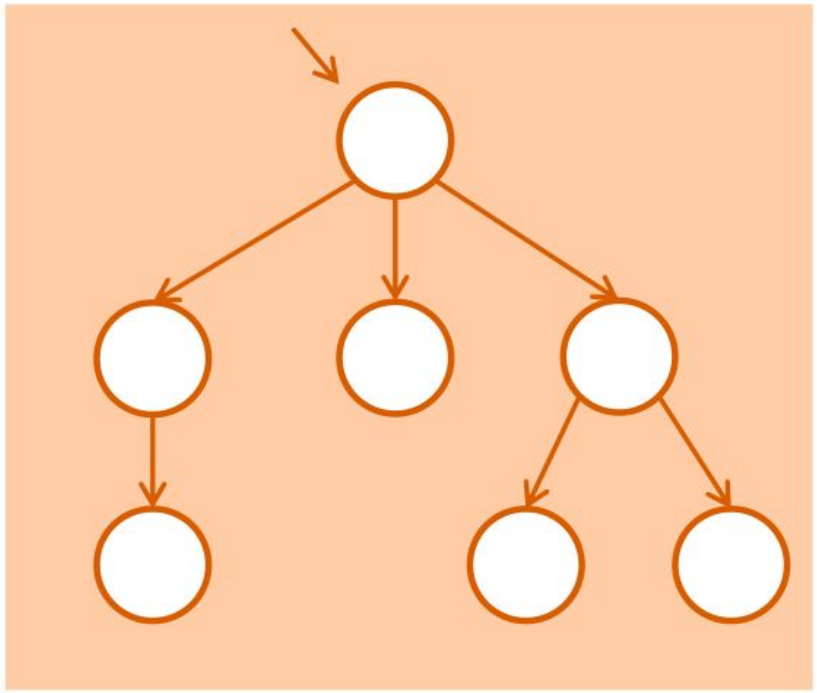
A. Listas generalizadas
B. Conceptos árboles
C. Árboles binarios
D. Recorrido de árboles binarios
E. Árboles binarios enhebrados
F. Heaps
G. Árboles de búsqueda binarios
H. Árboles de búsqueda binarios balanceados
I. Famila de árboles B (B, B*, B+)

Conceptos



Listas lineales

Cada elemento tiene 2 nodos como vecinos: anterior y sucesor

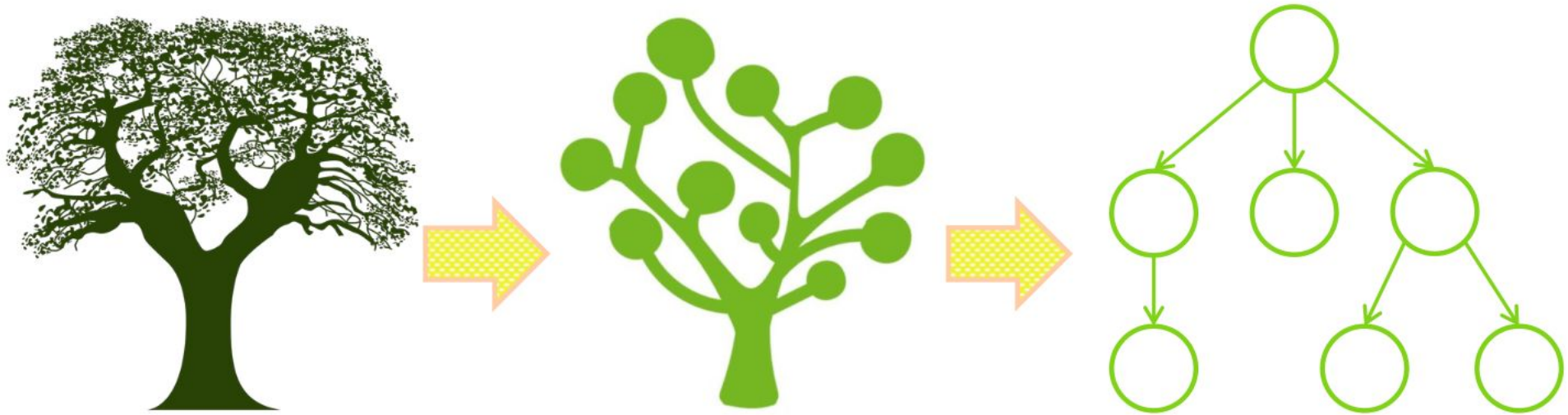


Listas no-lineales

Cada elemento puede tener varios siguientes elementos (ramificaciones)

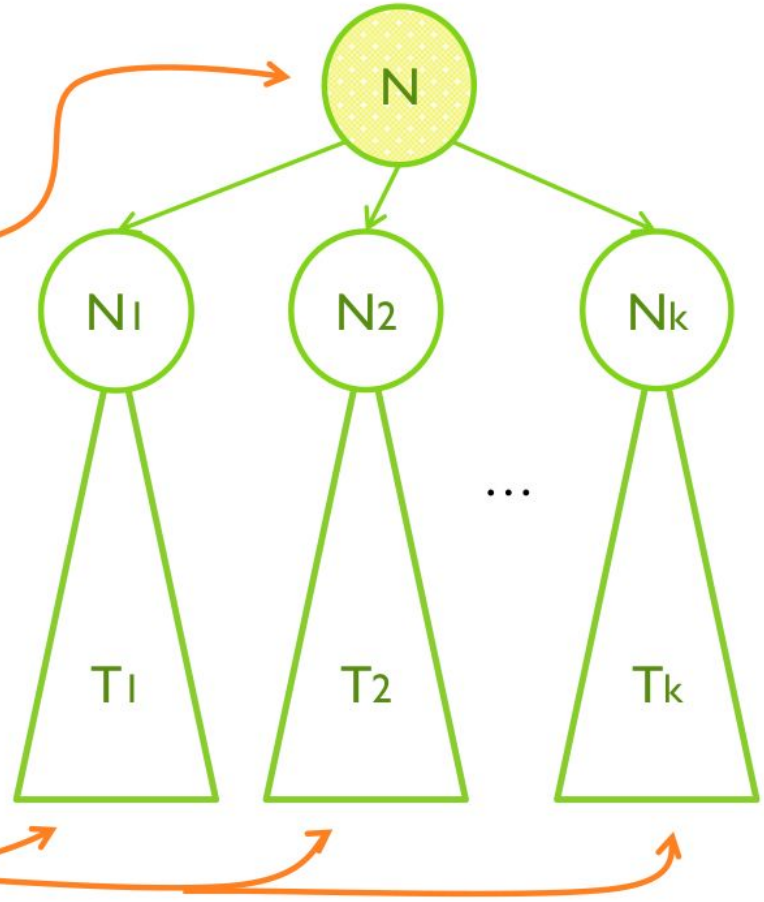
Conceptos

- Árbol:
  - Lista no-lineal que impone una estructura jerárquica sobre una colección de objetos.



Conceptos

- Árbol:
  - Es un conjunto finito de uno o más nodos tal que:
    1. Hay un nodo especial llamado **raíz**.
    2. Los nodos restantes son particionados en $k \geq 0$ conjuntos disjuntos $T_1, T_2, \ldots, T_k$, donde cada uno de estos conjuntos es un árbol. $T_1, T_2, \ldots, T_k$, son llamados "**sub-árboles**" de la raíz.



Conceptos

Grado de un nodo

- Cantidad de sub-árboles del nodo.

Hoja o nodo terminal

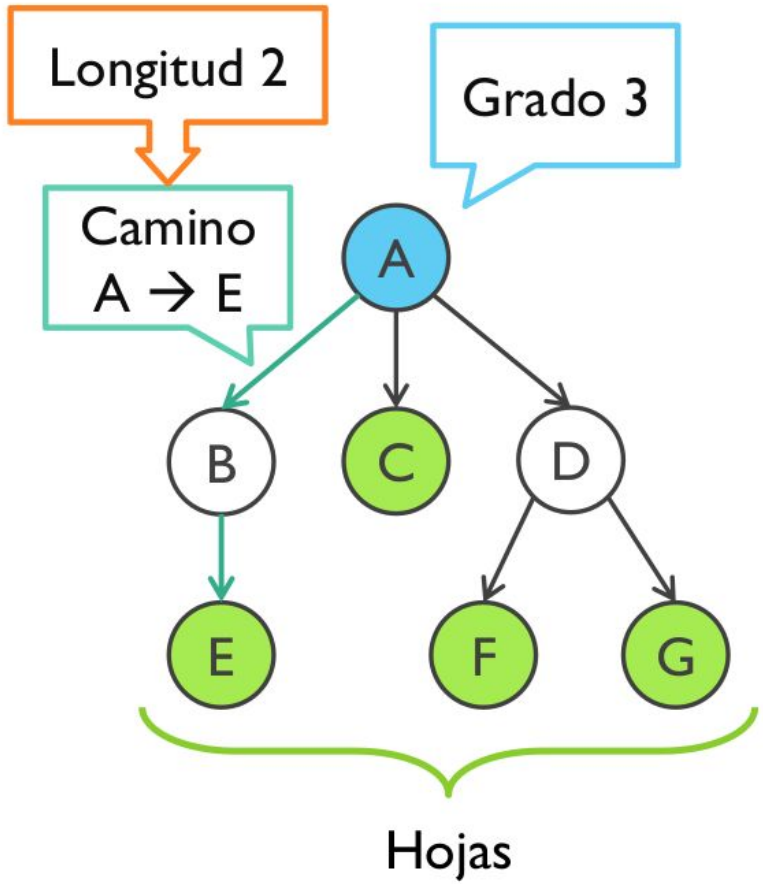
- Nodos con grado=0, es decir, nodo sin sub-árboles.

Camino

- Secuencia de uno o más arcos que conectan 2 nodos.

Longitud de un camino

- N° de arcos que componen el camino.



Conceptos

Hermanos

- Nodos con un mismo padre.

Nivel de un nodo

- La raíz tiene nivel 1. Si un nodo está en el nivel L, sus hijos están en el nivel L+1.

Altura o profundidad de un árbol

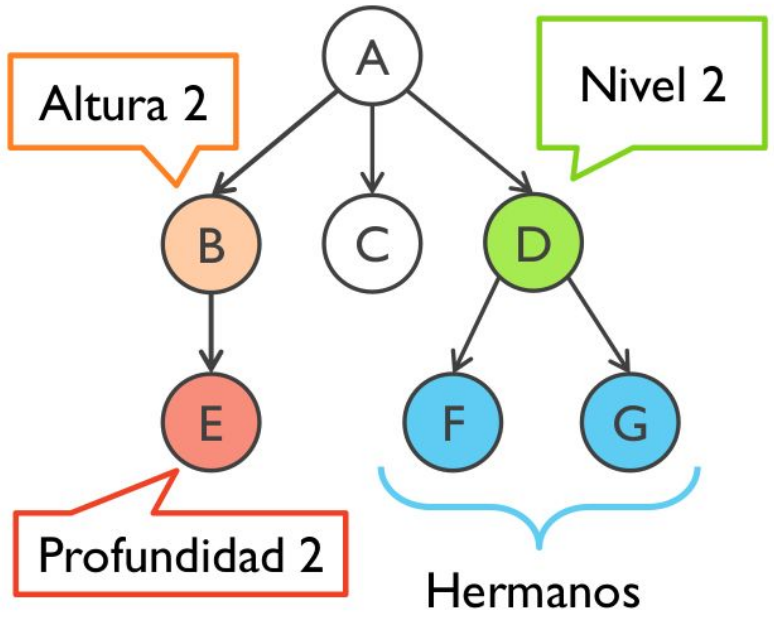
- Máximo nivel de cualquier nodo en el árbol.

Altura de un nodo

- Longitud del camino más largo desde el nodo a una hoja+1.

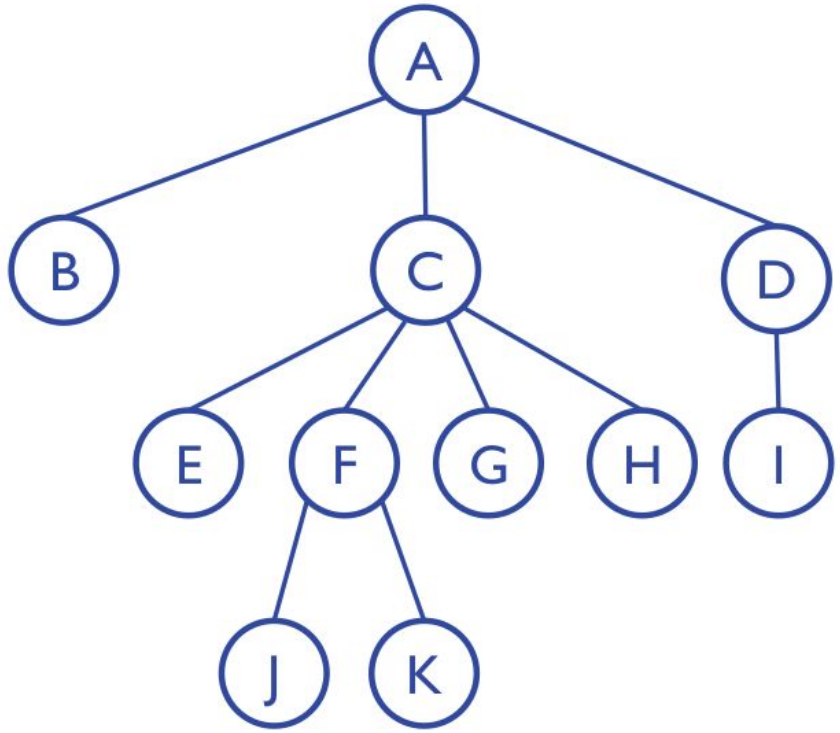
Profundidad de un nodo

- Longitud del único camino desde la raíz al nodo.



Altura del árbol: 3

Conceptos



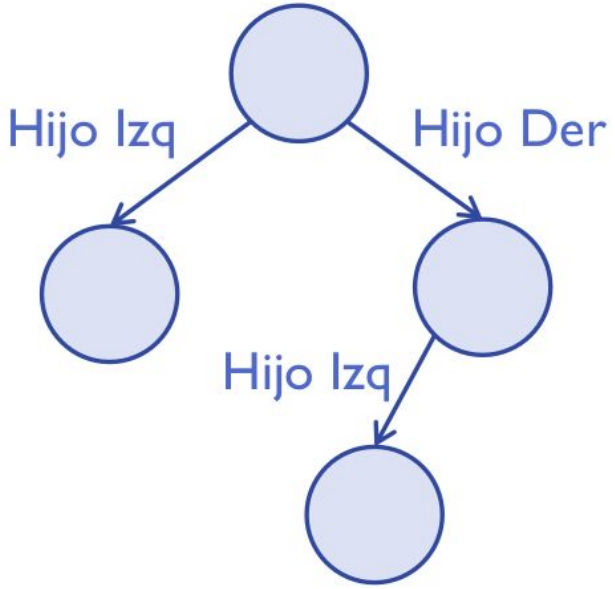
- Grado de C = 4
- Hojas = B, E, J, K, G, H, I
- Hermanos de B = C, D
- Camino de A a J = A→C, C→F, F→J
- Longitud del camino de A a J = 3
- Nivel de A = 1
- Nivel de I = 3
- Altura del árbol = 4
- Altura de C = 3
- Profundidad de C = 1

Contenidos sobre árboles

A. Listas generalizadas
B. Conceptos
C. Árboles binarios
D. Recorrido de árboles binarios
E. Árboles binarios enhebrados
F. Heaps
G. Árboles de búsqueda binarios
H. Árboles de búsqueda binarios balanceados
I. Famila de árboles B (B, B*, B+)

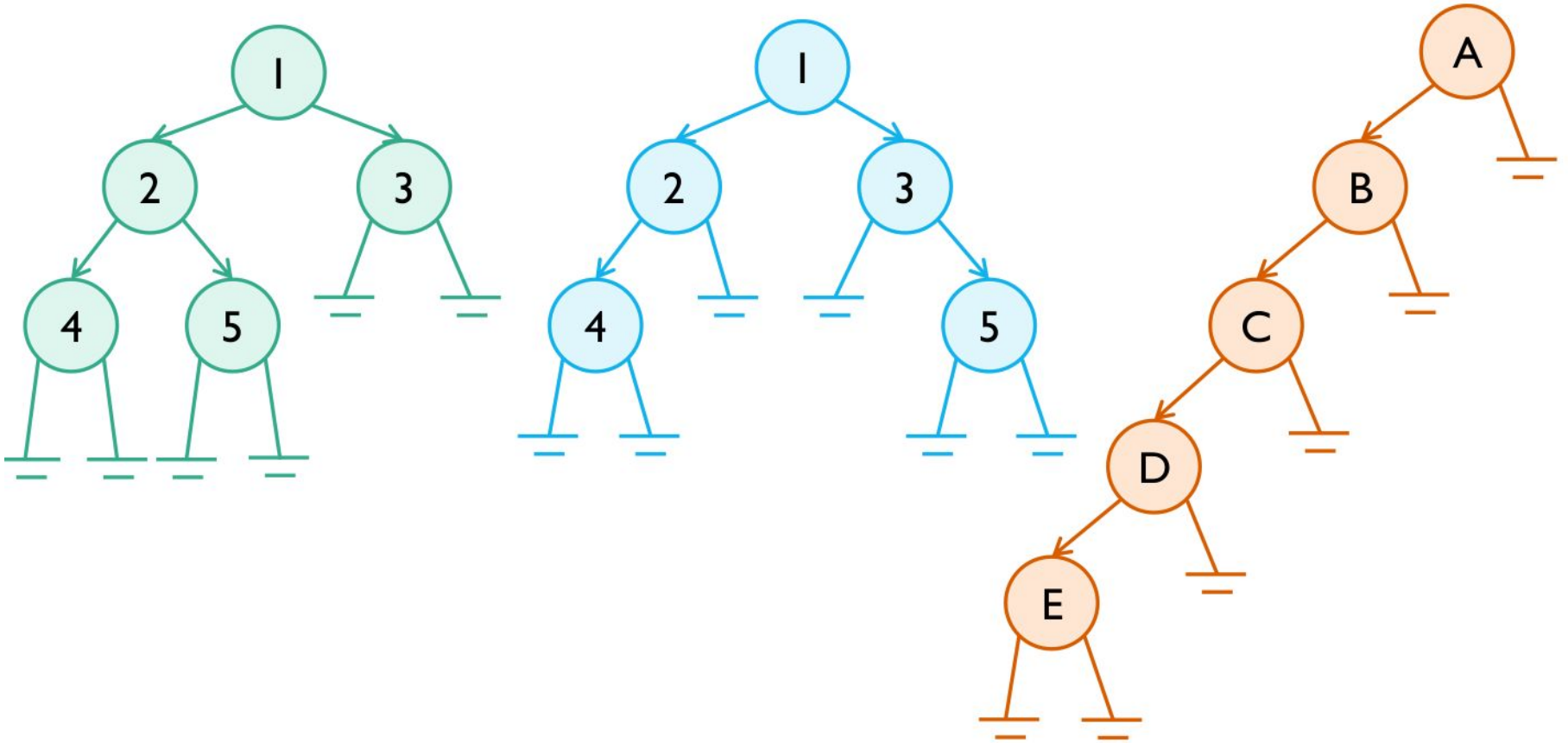
B) Árboles binarios

- Árbol que, en cada nodo, tiene a lo más 2 hijos:
  - Hijos izquierdos.
  - Hijos derechos.



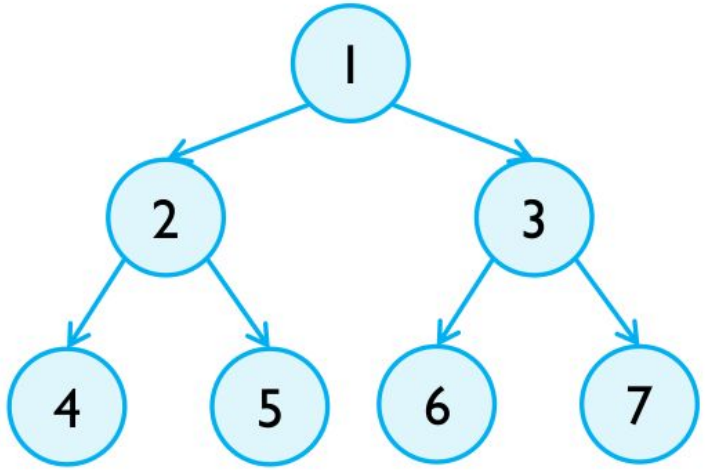
B) Árboles binarios

- Ejemplos de árboles binarios:



B) Árboles binarios

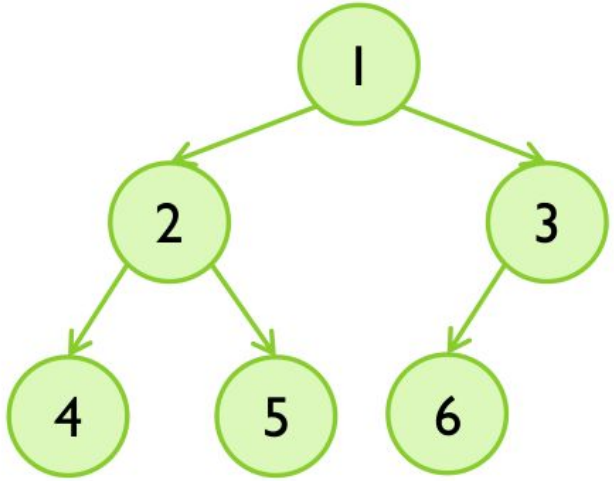
- *Full Binary Tree*:
  - Árbol binario de profundidad k y que tiene $2^k - 1$ nodos.



$$Cantidad\ de\ nodos = 2^k - 1 = 2^3 - 1 = 8 - 1 = 7$$

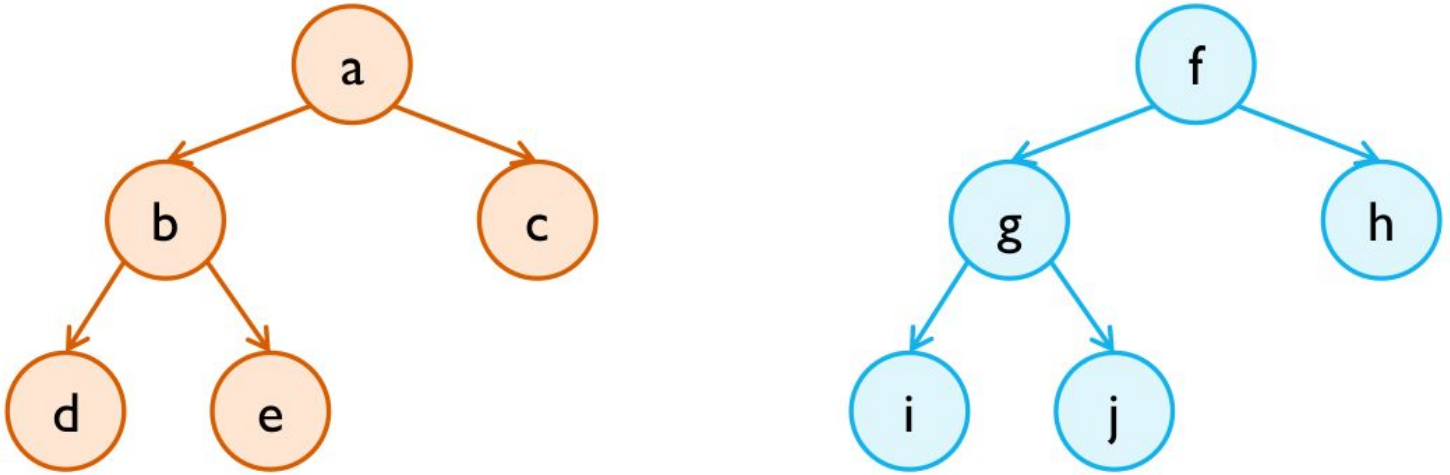
B) Árboles binarios

- *Complete Binary Tree*:
  - Árbol binario con hojas a lo más en 2 niveles adyacentes L-1 y L, en el cual las hojas del nivel L se encuentran en las posiciones de más a la izquierda.



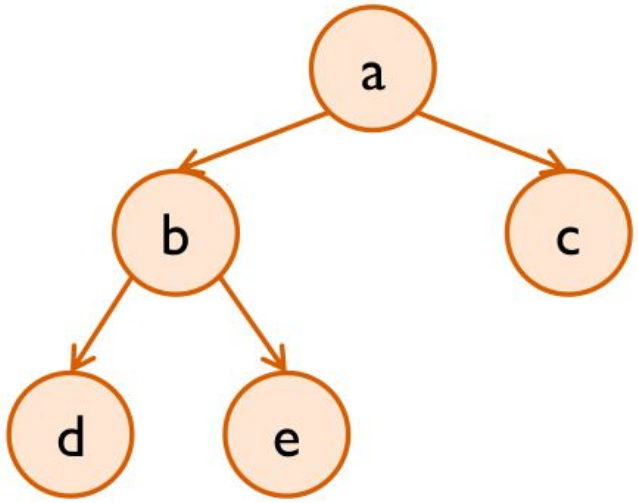
B) Árboles binarios

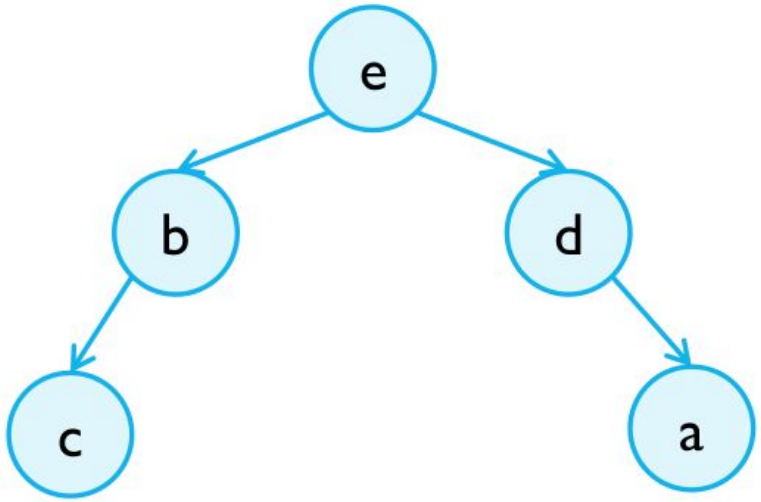
- Árboles isomorfos:
  - Tienen igual forma.



B) Árboles binarios

- Árboles semejantes:
  - Tienen los mismos elementos.

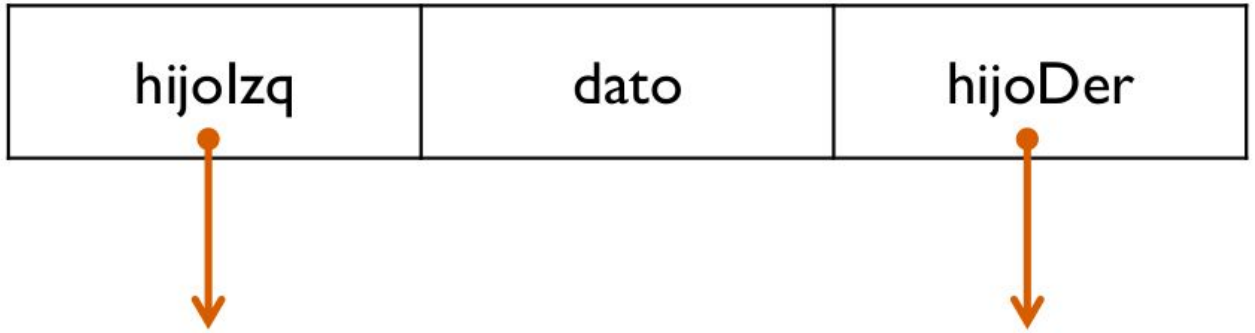




B) Árboles binarios

- Formato **nodo** de un árbol binario:

```
class Nodo{
    TipoElemento dato;
    Nodo* hijoIzq;
    Nodo* hijoDer;
}
```



Contenidos sobre árboles

A. Listas generalizadas
B. Conceptos
C. Árboles binarios
D. Recorrido de árboles binarios
E. Árboles binarios enhebrados
F. Heaps
G. Árboles de búsqueda binarios
H. Árboles de búsqueda binarios balanceados
I. Famila de árboles B (B, B*, B+)

C) Recorrido de árboles binarios

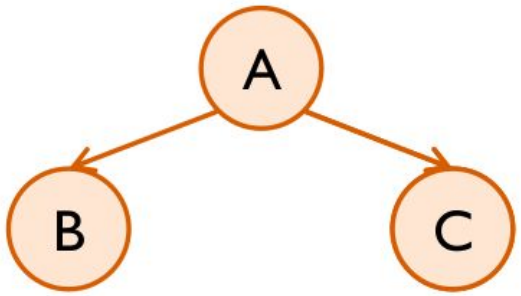
- Es examinar una sola vez cada uno de sus nodos.

- Tipos de recorridos más importantes:

Pre-orden	• **R**ID	Pre-orden inverso	• DIR
In-orden	• I**R**D	In-orden inverso	• DRI
Post-orden	• ID**R**	Post-orden inverso	• RDI
Por nivel	• De izquierda a derecha por nivel		

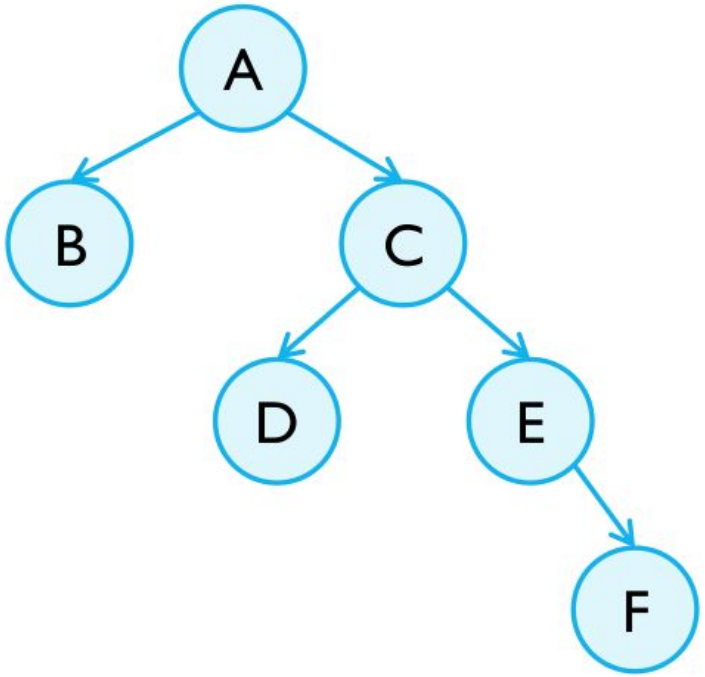
R → Raíz
I → Izquierda
D → Derecha

C) Recorrido de árboles binarios



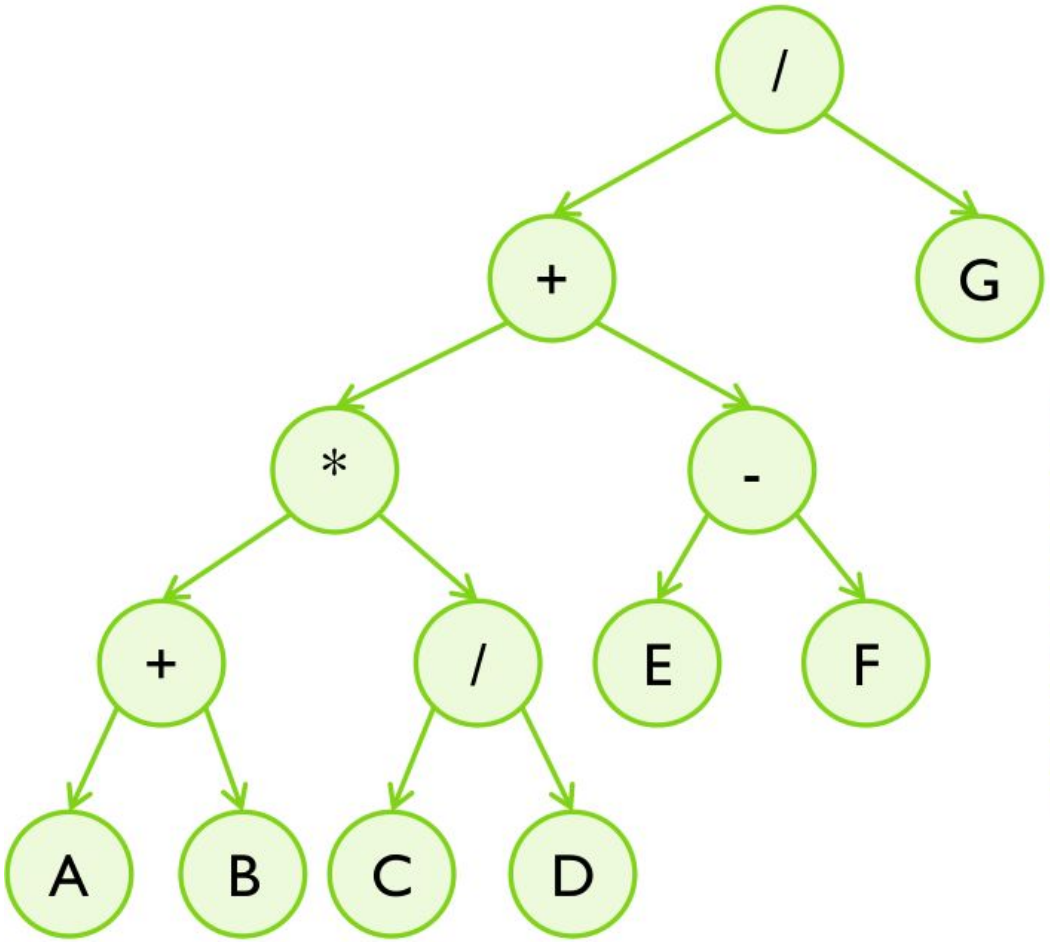
Pre-orden	A B C
↪ Inverso	C B A
In-orden	B A C
↪ Inverso	C A B
Post-orden	B C A
↪ Inverso	A C B
Por nivel	A B C

C) Recorrido de árboles binarios



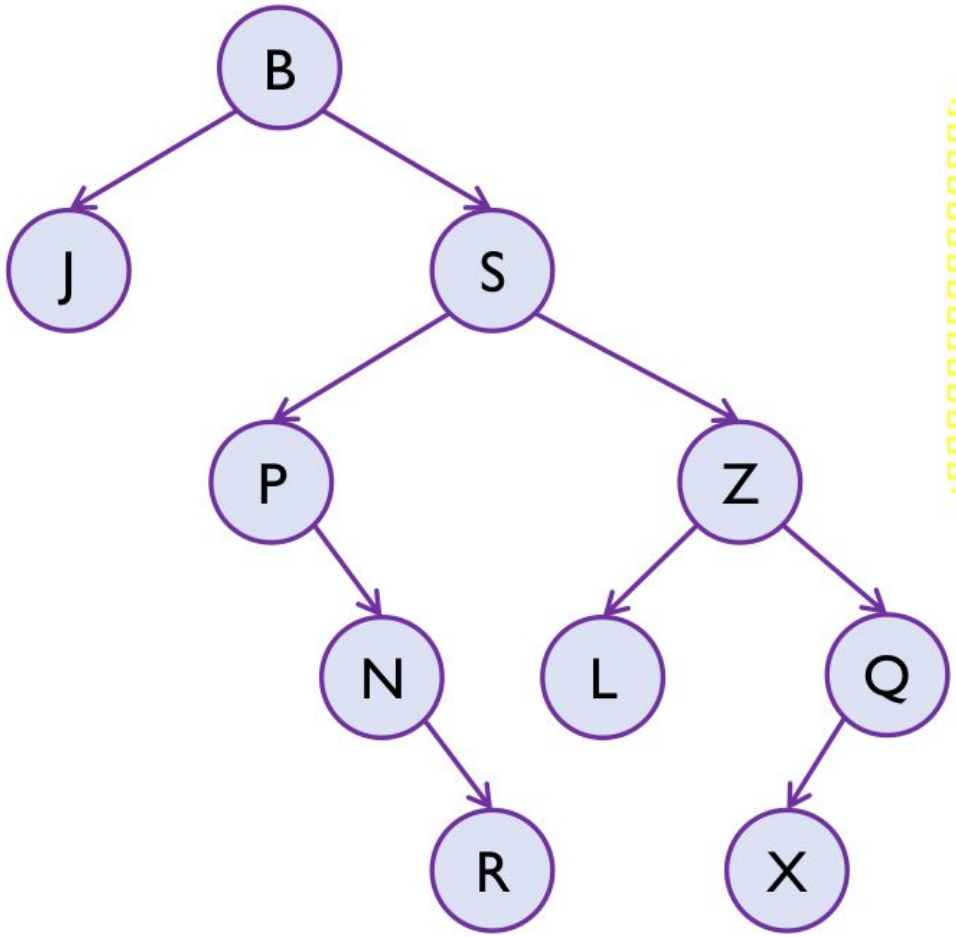
Pre-orden	A B C D E F
↪ Inverso	F E D C B A
In-orden	B A D C E F
↪ Inverso	F E C D A B
Post-orden	B D F E C A
↪ Inverso	A C E F D B
Por nivel	A B C D E F

C) Recorrido de árboles binarios



Pre-orden	/ + * + A B / C D – E F G
In-orden	A + B * C / D + E – F / G
Post-orden	A B + C D / * E F - + G /
Por nivel	/ + G * - + / E F A B C D

C) Recorrido de árboles binarios



Pre-orden	B J S P N R Z L Q X
In-orden	J B P N R S L Z X Q
Post-orden	J R N P L X Q Z S B
Por nivel	B J S P Z N L Q R X

C) Recorrido de árboles binarios

- Recorrido in-orden de un árbol binario:

1. Solución recursiva
2. Solución iterativa (usando *Stack*)
3. Solución con árboles enhebrados
4. Solución iterativa sin *Stack* y sin hebras

C) Recorrido de árboles binarios

- I) Solución recursiva

```
void recorridoInorder(){
    inorder(raiz);
}

void inorder(Nodo p){
    if(p){
        inorder(p->hijoIzq);
        print(p->dato);
        inorder(p->hijoDer);
    }
}
```

Contenidos sobre árboles

A. Listas generalizadas
B. Conceptos
C. Árboles binarios
D. Recorrido de árboles binarios
E. Árboles binarios enhebrados
F. Heaps
G. Árboles de búsqueda binarios
H. Árboles de búsqueda binarios balanceados
I. Famila de árboles B (B, B*, B+)