TEMA 6. Estructuras de datos avanzadas y POO

Estructuras de datos avanzadas y POO. Introducción.

Vectores y matrices, problemas:

- No se adaptan al tamaño del problema.
- Búsquedas complejas.
- Posibles problemas de memoria.(Memoria contigua).
- Manejo complejo: Inserción, borrado...

Aplicación restaurante gestión de objectos de clase pedido. Poblemática

- ¿Para cuantos pedidos como máximo?
- ¿A las 4 de la tarde se tiene los mismos pedidos que a las 9 de la noche?
- ¿Es sencillo buscar un pedido, por ejemplo por hora de pedido, por importe u otras combinaciones?
- Al servir un pedido. ¿Qué sucede con el espacio que ocupa en el array? ¿Si se soluciona, esta solución tiene algún coste?

Estructuras de datos avanzadas y POO. Introducción.

Solución a problemas anteriores: Estructuras dinámicas.

- Crece y decrecen en función de las necesidades.
- El espacio en memoria no es necesario que se encuentre contigua.

Otras características que DEPENDEN de la estructura:

- Velocidad de búsqueda.
- Inserción.
- Existencia de duplicados.
- Borrado.
- Espacio en memoria.

Elección se determinará por el tipo de problema a resolver.

Estructuras de datos avanzadas y POO. Introducción.

Uso de TAD. Tipos abstractos de datos.

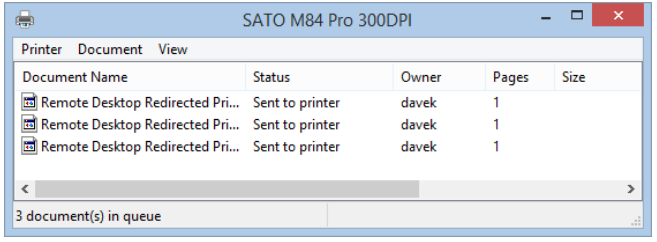
- Conjunto de valores.
- Conjunto de operaciones sobre los valores.

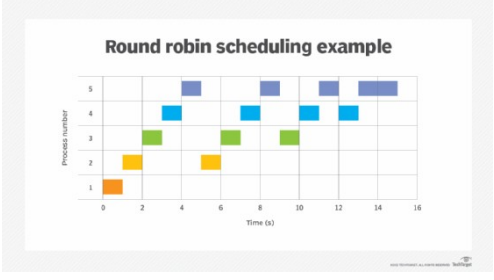
Vídeo UCAM sobre TAD. https://www.youtube.com/watch?v=Kcnp3e17Gq4

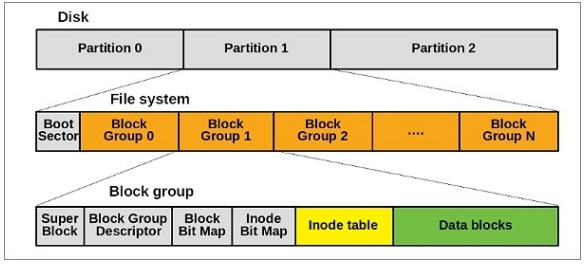
Estructuras de datos avanzadas y POO. Estructuras básicas.

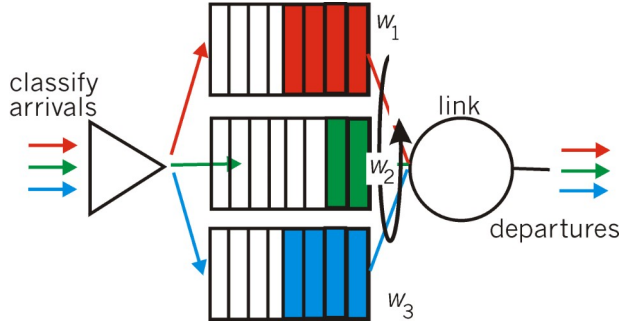
Estructuras básicas usadas en prácticamente todo el sector informático, no sólo programación.

- Sistemas operativos.
- Sistemas de ficheros.
- Tratamiento de paquetes en routers.
- Gestión de trabajos en impresora en red.
- …





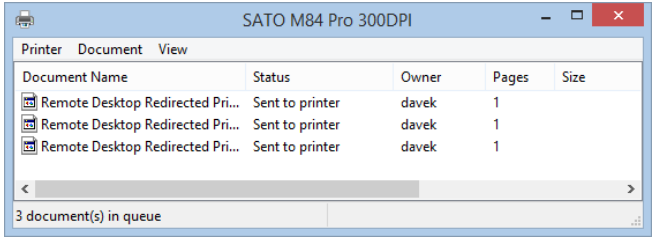


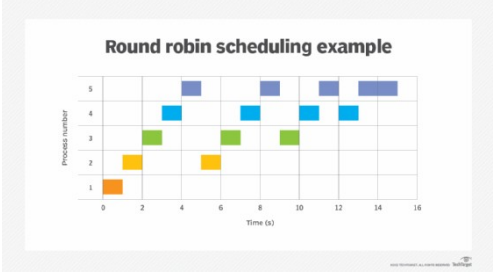


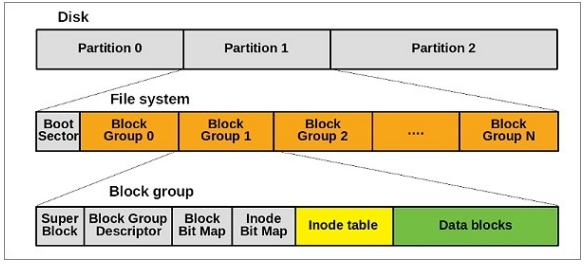
Estructuras de datos avanzadas y POO. Estructuras básicas.

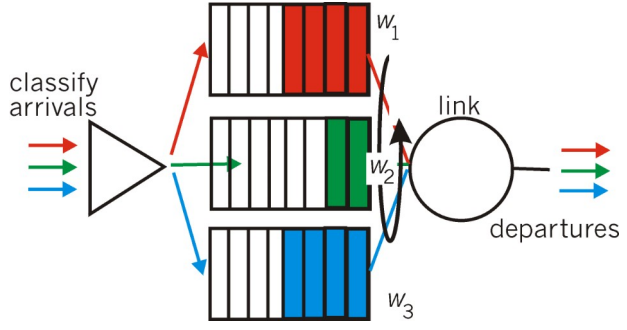
Estructuras básicas usadas en prácticamente todo el sector informático, no sólo programación.

- Sistemas operativos.
- Sistemas de ficheros.
- Tratamiento de paquetes en routers.
- Gestión de trabajos en impresora en red.
- …



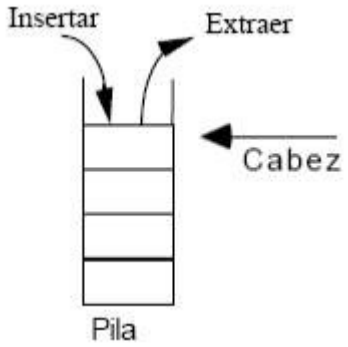






Estructuras de datos avanzadas y POO. Estructuras básicas.

PILAS (Stack).



Conjunto de elementos ordenados en función del momento en que se inserta.

Conocido también como LIFO (Last Input, Fist Output).

Operaciones.

- Create() ; //se crea una pila vacia
- Push(Element e); //apila un elemento en la cima de la pila
- Element Pop(); //devuelve el elemento que se encuentre en la cima de la pila
- Boolean isEmpty(); //indica si la pila se encuentra vacía.

Estructuras de datos avanzadas y POO. Estructuras básicas.

PILAS (Stack). Implementación.

Uso de operador “new” y referencias.

Concepto de puntero o referencia para “enlazar”.

Definir nodo que compone la pila.

Estructura recursiva.

Definir la pila.

```
public class StackNode {
    private String url;
    private StackNode next;
    public StackNode(String url){
        this.next=null;
        this.url=url;
    }
    public StackNode(String
url,StackNode next){
        this.next=next;
        this.url=url;
    }
}
```

PILAS (Stack). Implementación.

```
public class Stack {
    private StackNode top;
    public Stack() {
        this.top = null;
    }….
```

Operación push:

```
public void push(StackNode node) {
   node.setNext(this.top);
   this.top = node;
}
```

Operación está vacia:

```
public boolean isEmpty() {
   return this.top == null;
}
```

Operación pop.

```
public StackNode pop() {
   StackNode tempo = null;
   //si no es el final
   if (this.top != null) {
      tempo = top;
      //pasa al siguiente, si el
siguiente es nulo no pasa nada
      this.top = this.top.getNext();
      tempo.setNext(null);
        }
    return tempo;
}
....
```

PILAS (Stack). Implementación.

```
public static void main(String[] args) {
  Stack pila= new Stack();
  System.out.println("Al construir la pila
contiene:");
  System.out.println(pila);
  pila.push(new StackNode("localhost"));
  pila.push(new StackNode("www.google.com"));
  pila.push(new StackNode("www.facebook.es"));
  pila.push(new
StackNode("www.ieslaencanta.com"));
  System.out.println("Al insertar unos cuentos,
el estado es:");
  System.out.println(pila);
  StackNode tempo=pila.pop();
  System.out.println("Al desapilar se obtiene:");
  System.out.println(tempo.toString());
  System.out.println("El estado de la pila es:");
    System.out.println(pila);
}
```

Salida:

```
Al construir la pila contiene:
empty
Al insertar unos cuentos, el estado es:
www.ieslaencanta.com
www.facebook.es
www.google.com
localhost

Al desapilar se obtiene:
www.ieslaencanta.com
El estado de la pila es:
www.facebook.es
www.google.com
localhost
```

PILAS (Stack). Practicando.

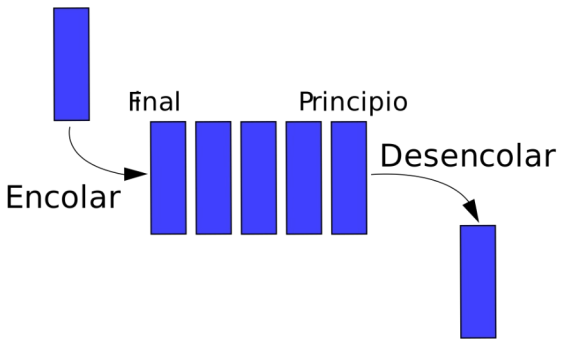
- Implementar usando recursión el método length() que devuelve la longitud de la pila, de igual forma implementar el método toString() que devuelve una cadena con la pila.
- En caso de querer gestionar una pila de objetos de una clase que representa una coordenada ¿Qué cambios introducir?
- Modificar el código para que se pueda añadir una cadena sin necesidad de crear o pasar como parámetro un StackNode, de igual forma para desapilar que devuelva solo la cadena.

Vídeo UCAM sobre Pilas:https://www.youtube.com/watch?v=JRPKWsbjmmI

COLA (Queue).

Conjunto de elementos ordenados en función del momento en que se inserta.

Conocido también como FIFO (Fist Input, Fist Output).



Operaciones.

- Create();
- Enqueue(Element e); //encola el elemento
- Element Dequeue(); //desencola el elemento y lo devuelve
- boolean isEmpty(); //si la cola esta vacia
- Element peek(); //devuelve el primer elemento de la cola, sin desencolar.

COLAS (Queue). Implementación.

Clase pedido:

```
public class Order {
    private String client;
    private Date date;
    private float amount;
    public Order(String client,Date date, float
amount){
        this.client=client;
        this.date=date;
        this.amount=amount;
    }…
 public String toString(){
        ….
```

Clase nodo:

```
public class Node {
    private Order order;
    //referencia al siguiente
    private Node next;
    public Node() {
        this.order = null; this.next = null;
    }
    public Node(Order order) {
        this.order = order; this.next = null;
    }
    public Node(Order order, Node next) {
        this.order = order; this.next = next;
    }
    public Node getNext() {
        return next;
    }
    public void setNext(Node next) {
        this.next = next;
    }….
```

COLAS (Queue). Implementación.

Clase Queue:

```
Public class Queue {
    //inicio de la cola, por donde sale
    private Node top;
    //final de la cola
    private Node end;
    public Queue() {
        this.top = null;
        this.end = null;
    }...
```

Operación es vacia:

```
public boolean isEmpty() {
  return this.top == null && this.end == null;
}
```

Operación encolar:

```
public void enqueue(Order order) {
  Node n = new Node(order);
  //la cola esta vacia
  if (this.top == this.end && this.top == null) {
      this.top = n;
      this.end = n;
  } else {
      // se apunta al anterior
      this.end.setNext(n);
      this.end = n;
   }}
```

Operación desencolar:

```
 public Order dequeue() {
   Order o = null;
   if (this.top !=null) {
     o= this.top.getOrder();
     this.top=this.top.getNext();
   }
   return o; }
```

COLAS (Queue). Implementación.

```
public class QueuePrincipal {
 public static void main(String[] args) {
 Queue cola = new Queue();
 System.out.println("Al construir  contiene:");
 System.out.println(cola);
 cola.enqueue(new Order("Paco", new Date(), 45));
 cola.enqueue(new Order("Paca", new Date(), 9));
 cola.enqueue(new Order("Pepe", new Date(),
234));
 System.out.println("Al insertar unos cuentos, el
estado es:");
 System.out.println(cola);
 Order tempo = cola.dequeue();
 System.out.println("Al desencolar se obtiene:");
 System.out.println(tempo.toString());
 System.out.println("Y el estado de la cola
es:");
 System.out.println(cola);
    }
}
```

Salida:

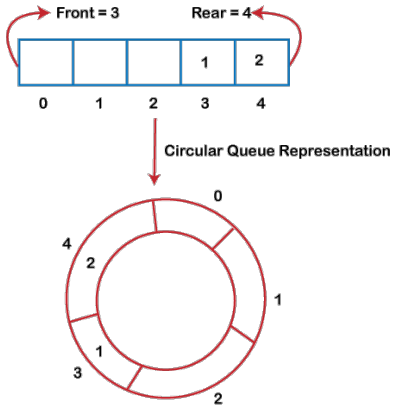
```
Al construir  contiene:
empty
Al insertar unos cuentos, el estado es:
Client: Paco Date:22/02/2022 Amount:45.0
Client: Paca Date:22/02/2022 Amount:9.0
Client: Pepe Date:22/02/2022 Amount:234.0
Al desencolar se obtiene:
Client: Paco Date:22/02/2022 Amount:45.0
Y el estado de la cola es:
Client: Paca Date:22/02/2022 Amount:9.0
Client: Pepe Date:22/02/2022 Amount:234.0
```

COLAS (Queue). Practicando.

- Implementar usando recursión el método length() que devuelve la longitud , de igual forma implementar el método toString() que devuelve una cadena con la cola.
- Añadir un método boolean Exists(Order o), que indique si una orden existe enla cola.

Otros tipos de colas:

- Colas dobles. Referencia/puntero siguiente-anterior.
- Colas circulares.

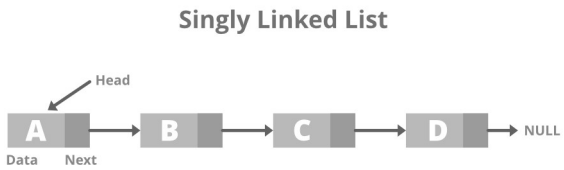


Estructuras de datos avanzadas y POO. Estructuras básicas.

LISTAS.

Las COLAS Y PILAS caso particular de las listas.

Similar a un vector.



Características:

- Acceso a cualquier posición que exista en la lista.
- Insertar y borrar elementos en cualquier posición.
- Es posible concatenar (unir) y dividir listas a través de sublistas.

LISTAS.

Operaciones básicas:

- Insert(Element e, int i); //inserta un elemento en la posición i-ésima
- Append(Element e); //añade el elemento e al final de la lista.
- getByIndex(int i); //devuelve el elemento que se encuentra en la posición i-ésima.
- Delete(int i); //elimina el elemento de la posición i-esima
- int Lenght(); //devuelve la longitud de la lista
- void Join(List l); //une dos listas
- List l subList(int from, int to); //extrae una sublista indicando los índices
- boolean Exists(Element e); //indica si un elemento se encuentra en la lista.

Estructuras de datos avanzadas y POO. Estructuras básicas.

LISTAS. Implementación.

Lista de órdenes. Nodo igual a la cola.

```
public class StackNode {
    private String url;
    private StackNode next;
    public StackNode(String url){
        this.next=null;
        this.url=url;
    }
    public StackNode(String
url,StackNode next){
        this.next=next;
        this.url=url;
    }
}
```

LISTAS. Implementación.

```
public class List {
    private Node start;
    public List() {
        this.start = null;
    }….
```

Operación getByIndex(int i):

```
public Order getByIndex(int index) {
  Order o = null;
  Node tempo = this.start;
  //mientras no llegue al index ono fin
  for (int i=0;i<=index && tempo!=
null; i++) {
    if (i == index) {
       o = tempo.getOrder();
    }
    tempo = tempo.getNext();
  }
  return o;
}
```

Operación insert.

```
public void insert(Order o) {
  Node now = this.start;Node last = now;
  Node new_node = new Node(o);
  new_node.setNext(null);
  if (this.start == null) { this.start = new_node; }
  else {
    while (now != null && now.getOrder().compareTo(o) < 0) {
        last = now;
        now = now.getNext();
    }
    if (now == null) { last.setNext(new_node); }
    else {
      if (now == start) {  this.start = new_node;
          new_node.setNext(last);
      }
      else {last.setNext(new_node);new_node.setNext(now);}
    }
  }
}
```

LISTA VACIA

SE BUSCA PUNTO INSERTAR

ÚLTIMO

PRIMERO

INTERMEDIO

LISTAS. Implementación.

Operación remove:

```
public Order remove(int index) {
  Order o = null;
  //se necesita dos temporales, el
actual y el anterior al actual
  Node now = this.start;
  Node last = now;
  if (this.start != null) {
     //si es el primero
     if (index == 0) {
       o=this.start.getOrder();
       this.start=this.start.getNext();
     } else {
     //se busca el punto a borrar
     for (int i = 0; i < index && now != null; i++) {
       last = now;
       now = now.getNext();
     }
     //si no es nulo se obtiene, el anterior,ahora
apunta al siguiente del actual (saltandolo)
     if (now != null) {
       last.setNext(now.getNext());
       o = now.getOrder();
       now = null;
     }
   }
  }
  return o;
}
```

LISTAS. Practicando.

- Se ha utilizado el método compareTo de la clase Order, ¿Cómo sería la implementación de ese método? ¿Existe algún otra clase que la utilice, por ejemplo ordenar elementos? ¿Qué peculiaridad tiene?
- ¿Cuál es el coste temporal de la búsqueda? ¿Es factible usar listas para almacenar registros en una base de datos, por ejemplo una tabla con un millón de registroso, o para gestionar el sistema de ficheros de un sistema operativo?. Debatir en clase posibles soluciones para hacer la búsqueda más rápida.
- Crear los métodos getByName y removeByName, en el que se utiliza el nombre para obtener un pedido y para borrar un pedido.

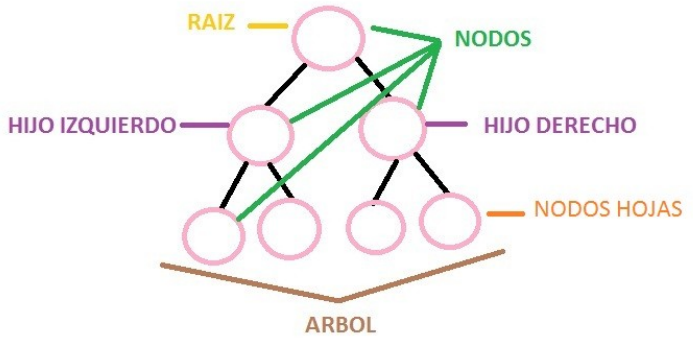
Vídeo UPV sobre listas, pias y colas: https://www.youtube.com/watch?v=-Shr2s0gYao

ÁRBOLES.

Estructuras anteriores problema con la búsqueda→ O(n).

Mejorar la búsqueda. O(log n).

Definición:



- Los árboles se componen de nodos.
- Un sólo nodo es por si solo un árbol.
- Al primer nodo de un árbol se le denomina raíz.
- Un nodo puede tener uno o más descendientes (las listas son árboles con nodos con un único descendiente), a los nodos descenciente se les denomina hijos y al antecesor de un nodo se le denomina padre.
- Los nodos terminales reciben el nombre de hoja.

ÁRBOLES.

Conceptos:

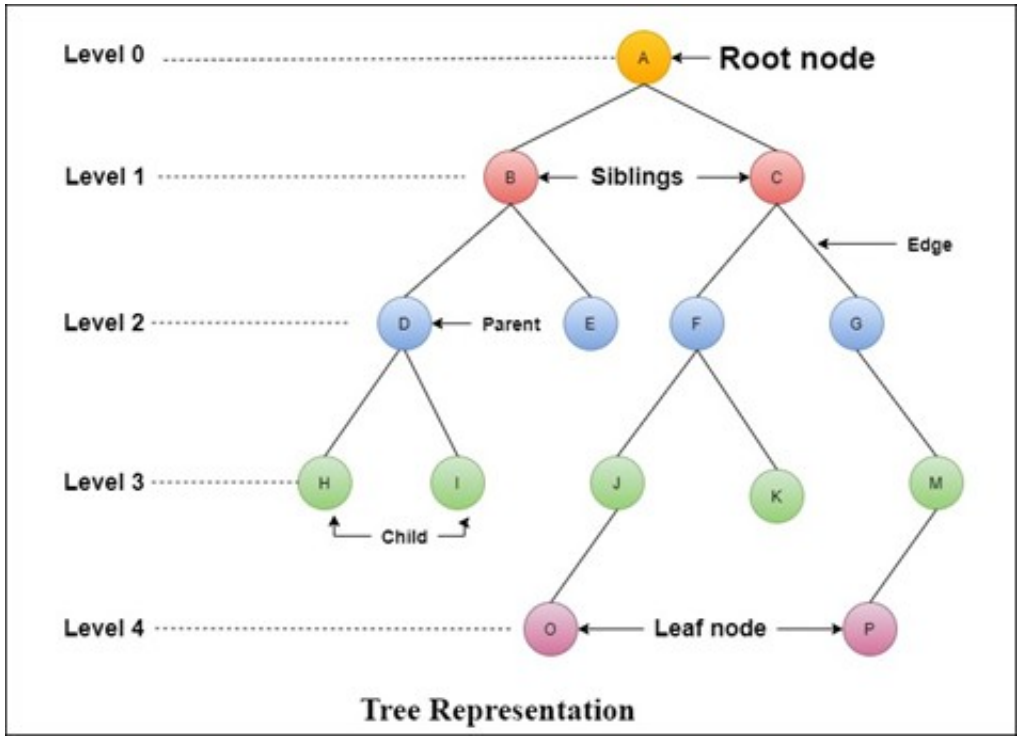
- Los nodos con el mismo padre se denomina **hermanos.**
- Los nodos se unen mediante **aristas**.
- Se define un **camino** en el árbol como la secuencias de aristas para llegar de un nodo A a un nodo B.
- La **profundidad** de un nodo es la longitud del camino de la raíz a ese nodo.
- La **altura** de un árbol es el camino más largo desde la ráiz a su hoja más profunda.
- Los árboles cuyos nodos pueden tener n hijos se denominan **árbones n-arios**.
- Una lista es un arbol unario. Los árboles binarios son aquellos cuyos

ÁRBOLES.

Conceptos:

Los árboles pueden estar ordenados o desordenados, en caso de ser ordenados suele hacerse de izquierda a derecha, es decir el subárbol de la izquierda es menor que el subárbol de la derecha.

UCAM. Árboles.
https://www.youtube.com/watch?v=kzQ49lgWd68

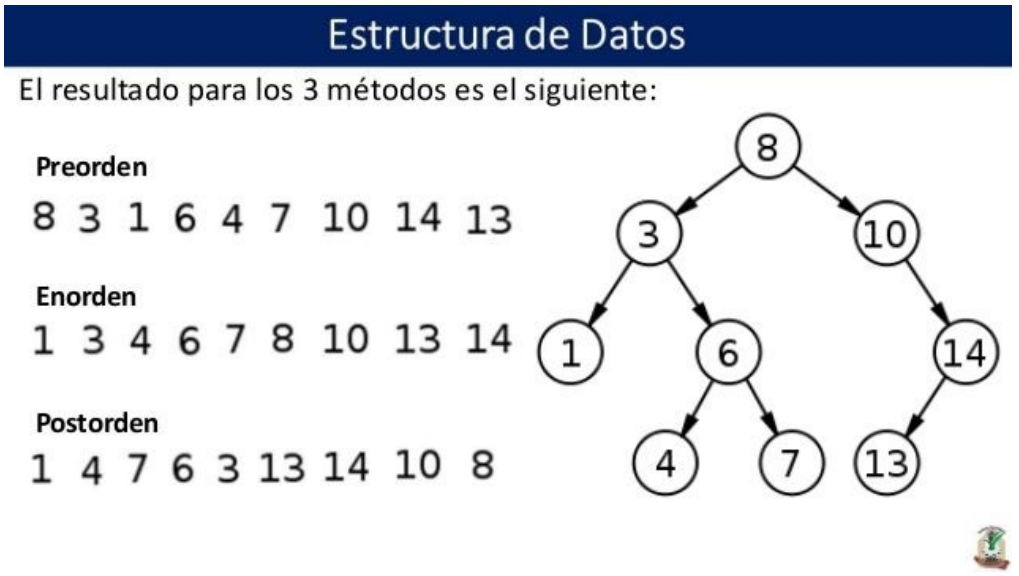


ÁRBOLES.

Conceptos:

Recorrido (recursividad).

- Preorden: raíz, árbol de la izquierda, árbol de la derecha.
- Inórden: árbol de la izquierda, raíz, árbol de la derecha.
- Postorden: Centro, árbol de la izuierda, árbol de la derecha, raíz.



ÁRBOLES.

Conceptos:

Operaciones básicas.

- Node Parent(Node n); //Devuelve el padre del nodo.
- Node Root(); //Devuelve la raíz del árbol.
- Node Search(parameters); //busca a partir de los parámetros un nodo devolviéndolo si lo encuentra.
- Insert(Element e); //inserta el elemento en el árbol
- Delete(Element e); //borra un elemento del nodo
- Preorden(); //Recorre el árbol en preorden
- Inorden();
- Posrorden();

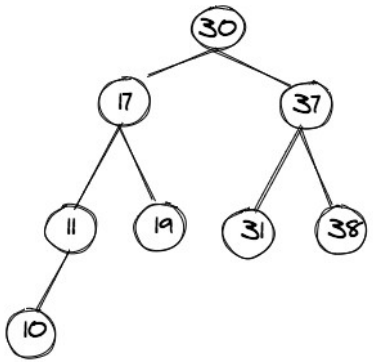
Estructuras de datos avanzadas y POO. Estructuras básicas.

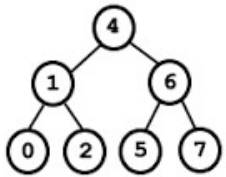
ÁRBOLES.

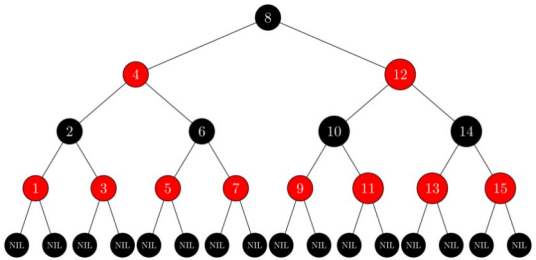
Conceptos:

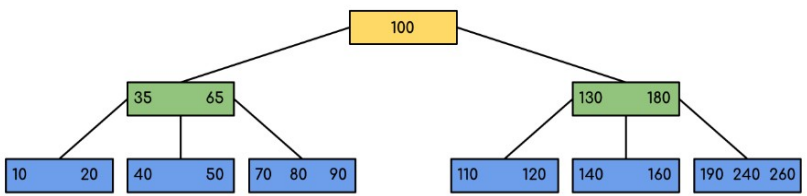
Tipos de árboles.

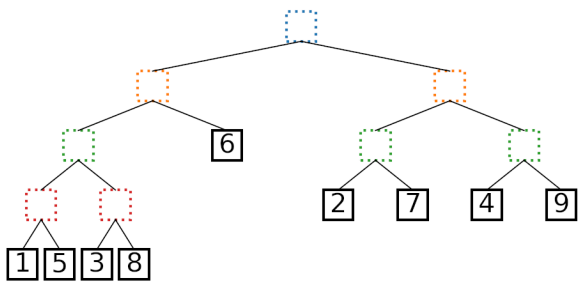
- Árboles binarios.
- Árboles AVL. (Balanceado)
- Árboles rojo-negro.
- Árboles B,B+,B*.
- AABB.
-

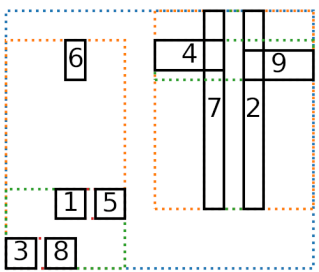












ÁRBOLES. Implementación. Ejemplo binario de búsqueda.

Ordenados de izquierda a derecha, tiempo O(log2 n).

```
public class Node {
    private Order order;
    //referencia al siguiente
    private Node left;
    private Node right;
    public Node() {
        this.order = null;
        this.left = null;
        this.right=null;
    }
    public Node(Order order) {
        this.order = order;
        this.left = null;
        this.right=null;
    }
    ….
```

```
public class SortBinaryTree {

    private Node root;

    public SortBinaryTree() {
        this.root = null;
    }

    public SortBinaryTree(Order o) {
        this.root = new Node();
        this.root.setOrder(o);

    }
….
```

ÁRBOLES. Implementación. Ejemplo binario de búsqueda.

Búsqueda.

```
public Order search(int amount) {
   Order vuelta = null; Node actual = null;
   if (this.root != null) {
     actual = this.root;
    while (actual != null && actual.getOrder().getAmount() != amount) {
      if (actual.getOrder().getAmount() > amount) {
         actual = actual.getLeft();
      } else if (actual.getOrder().getAmount() < amount)
          actual = actual.getRight();
    }
    if (actual!=null && actual.getOrder().getAmount() == amount)
      vuelta=actual.getOrder();
  }
  return vuelta;
}
```

ÁRBOLES. Implementación. Ejemplo binario de búsqueda.

Recorrido.

```
public String preorden() {
        if (this.root != null)
            return this.preorden(this.root);
        else
            return "empty";
    }
private String preorden(Node n) {
        StringBuilder vuelta = new StringBuilder();
        if (n.getOrder() != null)
            vuelta.append(n.getOrder());
        if (n.getLeft() != null)
            vuelta.append(this.preorden(n.getLeft()));
        if (n.getRight() != null)
            vuelta.append(this.preorden(n.getRight()));
        return vuelta.toString();
    }
```

ÁRBOLES. Implementación. Ejemplo binario de búsqueda.

Operaciones inserción.

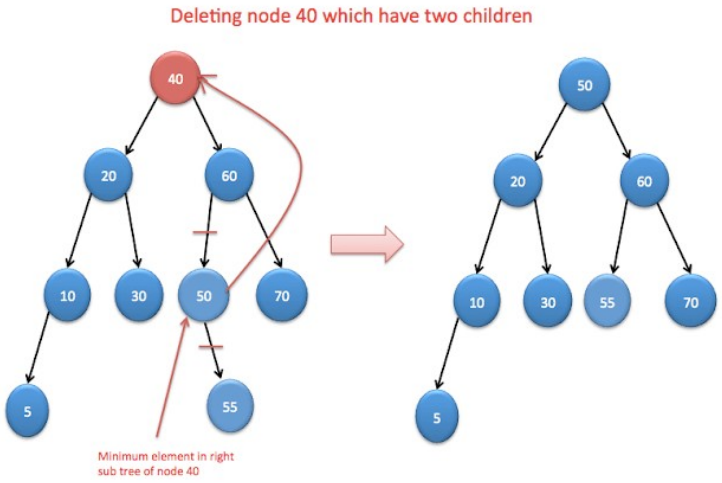
Buscar hoja por el árbol.

¿ Rama izquierda o derecha?

Operación borrado, más compleja.

Dos alternativas:

- Buscar el menor a la derecha y sustituir.
- Buscar el mayor a la izquierda y sustituir.



UPV. Insercción en árbol binario de búsqueda. https://www.youtube.com/watch?v=Yy_0EqTpFPs

ÁRBOLES. Practicando.

- Implementar un método que devuelva el número de elementos del árbol binario de búsqueda.
- Dibujar el árbol binario de búsqueda con clave el id, resultado del siguiente código:

```
SortBinaryTree tree;
tree = new SortBinaryTree();
tree.insert(new Order("Paco", new Date(), 1));
tree.insert(new Order("Paca", new Date(), 2));
tree.insert(new Order("Pepe", new Date(), 3));
tree.insert(new Order("Pepa", new Date(), 4));
tree.insert(new Order("Pedro", new Date(), 5));
tree.insert(new Order("Petra", new Date(), 6));
tree.insert(new Order("Pilar", new Date(), 7));
```

- ¿Existe algún problema con el árbol anterior? ¿Alguna idea para solucionarlo?

ÁRBOLES. Otros árboles.

Árboles AVL.

Cada rama se encuentra balanceada, la diferencia entre las profundidades de cada rama de un nodo es como mucho 1. La búsqueda de cualquier nodo es del orden log2(n), siendo n el tamaño del árbol.

Árboles B.

Árboles n-arios, cada nodo puede tener como máximo 2n hijos y como mínimo n, definiendo n al crear el árbol. Garantizando una ocupación de un 50%.
Se utilizan principalmente en los sistemas de ficheros (NTFS,EXT…) ya que es posible cargar en memoria secciones del árbol. La información se almacena en los nodos intermedios y en las hojas.

Árboles B+.

Similar al árbol B, pero en este caso la información se almacena únicamente en las hojas. Los intermedios solo almacena referencias y claves de búsqueda. Suelen tener referencias a la siguiente hoja aunque no formen parte de la misma rama.

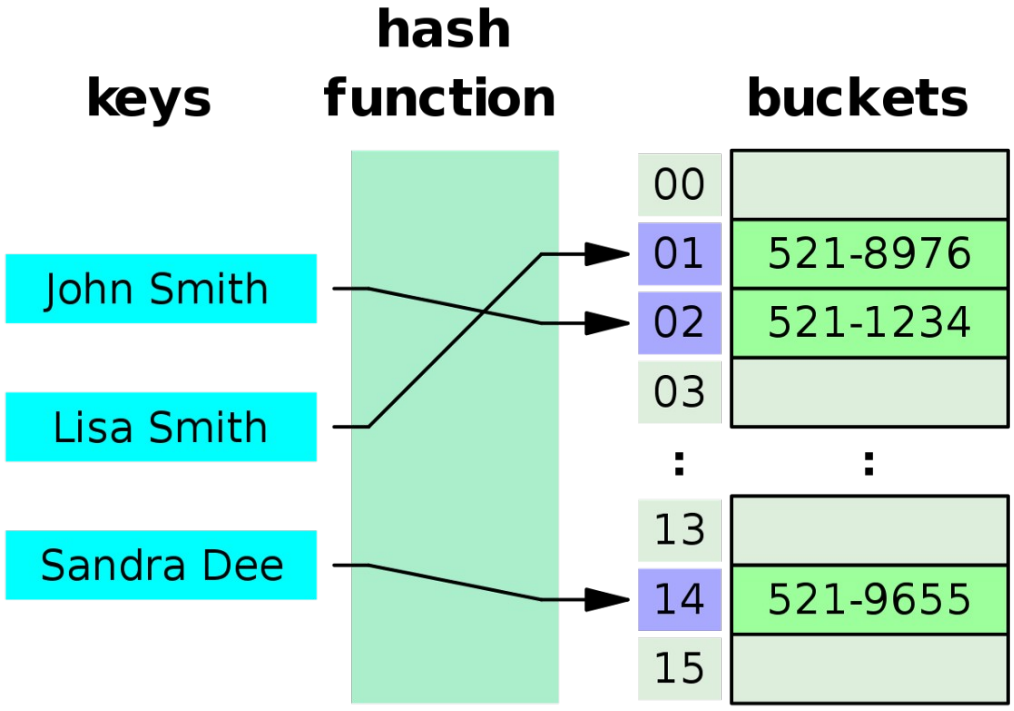
Estructuras de datos avanzadas y POO. Estructuras básicas.

Tablas HASH.

- También conocidas como DICCIONARIOS.
- Combinación de vector de un tamaño dado con el cálculo de su índice a partir de una llave (key).
- Acceso al elemento con la llave.
- Operaciones con un tiempo constante O(C), independiente del tamaño, en especial la búsqueda.
- Las inserciones pueden ser problematicas (colisiones con llaves que al transformarlas devuelven el mismo índice).

Tablas HASH.

- Tabla de 16 elementos.
- Almacena cadenas (Strings).
- Clave el nombre (String).
- Transformación de nombre a entero (índice).
- ¿Qué método de una clase puede ser una buena función Hash?



Tablas HASH.

- Función Hash de John Smith y “Juan Palomo” es la misma.
  - Colisión.
  - Diversas soluciones:
    - Común cada celda a su vez una lista.
    - Cambiar tamaño vector.
    - Cambiar función Hash.

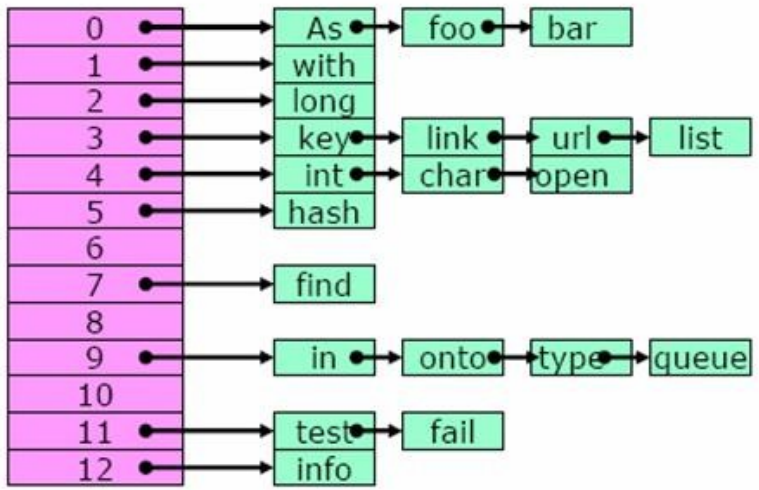


Tabla Hash UPV. https://www.youtube.com/watch?v=WnLdu8OHA3Q

Grafos.

- Un árbol, una pila, una lista o una cola son un tipo de grafo.
- Conjunto de nodos y aristas o arcos(conexión entre 2 nodos)
- Definición formal: G=(V,E), conjunto V de nodos (vértices) y un conjunto E de aristas (arcos). Cada arista es un par (v,w), siendo v y w un par de nodos pertenecientes al conjunto V de nodos.
- Se distinguen entre grafos dirigidos (indica si se puede ir de uno a otro traduciéndose la arista en una flecha que va desde el nodo v al nodo w (dirección navegación), o no dirigido donde los nodos están unidos mediante líneas sin indicación de dirección.

Grafos.

- Se puede asignar un coste a cada arco (ponderado).

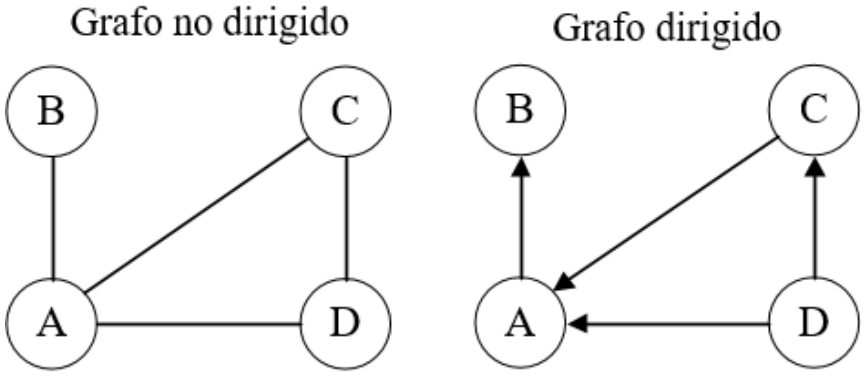
Terminología:

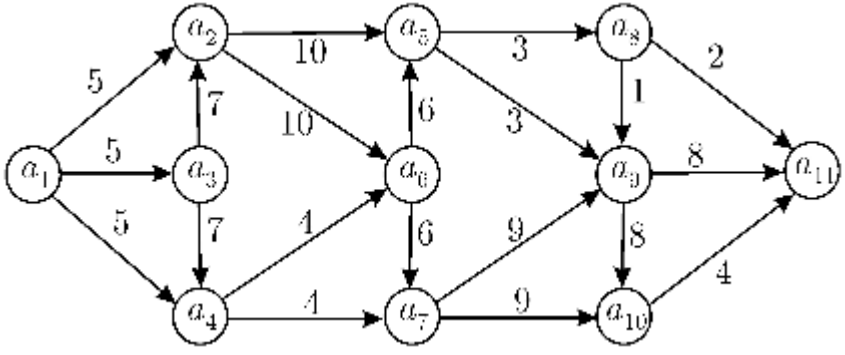
Vértices adyacentes: Están unidos por un arco.

Camino entre vértices:secuencia de vértices tal que dos vértices consecutivos son adyacentes.

Camino simple: No se repiten vértices.

Ciclo: Camino donde el primer vértices y último son el mismo.

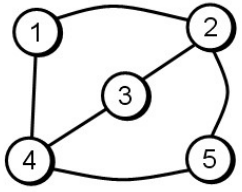




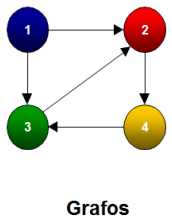
Estructuras de datos avanzadas y POO. Estructuras básicas.

Grafos.

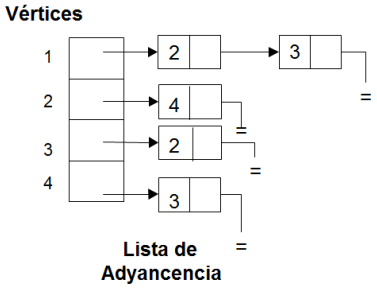
- Implementación:
  - Matrices de adyacencia.
  - Listas de adyacencia.
  - Matrices disperas.



M	1	2	3	4	5
1	0	1	0	1	0
2	1	0	1	0	1
3	0	1	0	1	0
4	1	0	1	0	1
5	0	1	0	1	0

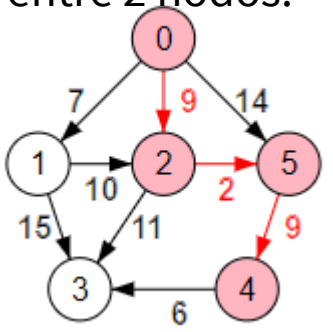


Grafos



Lista de Adyancencia

- Algoritmo de Dijkstra, calcula el camino mínimo de un grafo ponderado entre 2 nodos.
  - Tablas de rutas de enrutadores.
  - GPS.
  -



BettaTech: https://www.youtube.com/watch?v=5k2DWMRTXMM